Memorias adaptativas para el problema de ruteo de vehículos con múltiples viajes

Alfredo Olivera

Tesis de Maestría en Informática PEDECIBA

Orientador y supervisor: Omar Viera

Departamento de Investigación Operativa
Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Agosto de 2005

Resumen

En este trabajo se estudia el Problema de Ruteo de Vehículos con Múltiples Viajes o *Vehicle Routing Problem with Multiple Trips* (VRPMT). En esta variante del problema de ruteo de vehículos, se dispone de una flota limitada y se supone que cada uno de ellos puede recorrer más de una ruta en el mismo período de planificación. Se busca obtener, además de un conjunto de rutas de costo mínimo para visitar a los clientes, una asignación de esas rutas a los vehículos. Existe un horizonte de tiempo prefijado que limita las posibles asignaciones.

La inclusión de estas características, que en la mayor parte de los modelos clásicos no son tenidas en cuenta, hace que el VRPMT sea un problema de relevancia práctica. Además reviste un alto grado de complejidad, pues encontrar una solución factible es un problema NP-Completo. Su importancia práctica y su complejidad computacional son dos aspectos que motivan el estudio de esta variante del problema.

Se propone un algoritmo de resolución basado en la técnica conocida como *Adaptive Memory Procedure*. Esta técnica consiste en utilizar una estructura de memoria para generar soluciones iniciales sobre las que se ejecuta una heurística de búsqueda local. Las soluciones obtenidas por dicha heurística se utilizan, a su vez, para modificar la información almacenada en la memoria. En este trabajo se utiliza un algoritmo de búsqueda local basado en la metaheurística *Tabu Search*. Se experimenta con dos estrategias para reducir el tamaño de las vecindades y el tiempo de ejecución.

Los algoritmos fueron evaluados sobre un conjunto de instancias de prueba, obteniendo soluciones que pueden considerarse de buena calidad. En particular, se resolvieron instancias para las que los trabajos anteriores únicamente reportaban soluciones no factibles.

Palabras clave: problemas de ruteo de vehículos, múltiples viajes, adaptive memory procedure, tabu search.

Agradecimientos

Al Departamento de Investigación Operativa y al PEDECIBA, por haberme brindado apoyo económico mediante becas en diferentes etapas del proceso. A mis compañeros del departamento, con quienes tuve la oportunidad de intercambiar opiniones e ideas durante el desarrollo de este trabajo. A Héctor Cancela, María Urquhart y Omar Viera, por el interés que muestran en mi formación y mi desarrollo académico.

A mis padres, por haberme dado la posibilidad de realizar mis estudios.

Y a Virginia, por su estímulo, apoyo y paciencia.

Indice

Capítulo 1

Introducción

1.1. Contexto

El transporte de objetos entre sitios dispersos geográficamente constituye un problema clave de la economía moderna y trae consigo costos considerables. En algunos estudios se estima que el costo del transporte representa entre el 10 % y el 20 % del costo final de los bienes [126]. La adecuada gestión de la distribución puede significar grandes ahorros, lo que justifica en gran medida el rol de la Investigación Operativa para facilitar la planificación del transporte y la logística.

En ese sentido, las últimas cuatro décadas han visto un enorme esfuerzo por formular y resolver los llamados problemas de ruteo de vehículos. Luego de la publicación del algoritmo de ahorros [29] el área ha experimentado un crecimiento explosivo. Por un lado, hacia modelos que incorporen cada vez más características de la realidad, y, por otro lado, en la búsqueda de algoritmos que permitan resolver los modelos de manera eficiente. El ruteo de vehículos suele mencionarse como un caso de éxito de la Investigación Operativa.

Este éxito se debe, en buena parte, a la evolución de los sistemas informáticos. El incremento del poder de cómputo y la baja en sus costos, ha permitido el desarrollo de algoritmos cada vez más potentes y rápidos. Esto posibilita la resolución de modelos complejos, que introducen gran cantidad de características de los sistemas reales y hace viable, además, el desarrollo de métodos para la planificación en tiempo real y la toma semi-automática de decisiones. Por otro lado, el desarrollo de los Sistemas de Información Geográfica es fundamental para lograr una adecuada interacción de los modelos y algoritmos con las personas encargadas de realizar la planificación y llevarla adelante.

En un problema de ruteo de vehículos existen tres elementos principales: clientes, vehículos y depósitos. Los clientes representan sitios que demandan ser visitados, los vehículos son quienes se mueven para realizar dichas visitas y los depósitos constituyen los lugares de partida y regreso de los vehículos. Suelen modelarse como problemas de optimización combinatoria, en los cuales existe un conjunto de restricciones y una función

objetivo. Las restricciones se utilizan para modelar las diferentes características y limitaciones asociadas a los clientes, vehículos y depósitos, mientras que la función objetivo representa una medida del costo de cada una de las potenciales soluciones. Resolver el problema consiste en encontrar un conjunto de rutas para los vehículos, que respeten todas las restricciones y de modo que se minimice el valor de la función objetivo.

Los algoritmos para resolver este tipo de problemas pueden clasificarse en dos grandes grupos: algoritmos exactos y heurísticas (o algoritmos aproximados). Los algoritmos exactos ofrecen la garantía de que al final de la ejecución se obtiene la mejor solución posible, y suelen requerir tiempos de ejecución elevados (esto es, exponenciales en el tamaño de la entrada). Las heurísticas, en cambio, requieren tiempos de ejecución más moderados (o al menos controlables) pero no garantizan obtener la mejor solución.

La mayoría de las variantes de estos problemas, aún las que aparentan ser más simples, pertenecen a la clase NP-Hard [57, 91]. Esto implica que si la conjetura usual P $\neq$ NP es cierta, no será posible construir un algoritmo exacto cuyo tiempo de ejecución en el peor caso esté acotado por un polinomio en el tamaño de la entrada. Diseñar algoritmos que permitan resolver problemas de la clase NP-Hard de manera eficiente es un desafío situado en las fronteras de la computación.

1.2. Motivación del trabajo

En muchos de los problemas de ruteo de vehículos estudiados en la literatura académica suele hacerse dos simplificaciones importantes: suponer que se dispone de una cantidad no acotada de vehículos y suponer que cada vehículo recorre una sola ruta durante el período planificado. En un contexto de planificación estratégica, en el cual no se desee obtener una solución detallada sino, por ejemplo, una estimación de los costos, estas hipótesis pueden ser válidas. Pero a nivel operativo, si se pretende determinar un conjunto de rutas para llevar a la práctica, las simplificaciones mencionadas pueden quitarle utilidad práctica al modelo alejándolo demasiado de la realidad.

En este trabajo se estudia una variante del problema de ruteo de vehículos en la cual se dispone de una cantidad fija de vehículos y cada uno de estos puede recorrer más de una ruta en el mismo período de planificación. Este problema es conocido como el Problema de Ruteo de Vehículos con Múltiples Viajes (VRPMT por *Vehicle Routing Problem with Multiple Trips*). Esta variante ha sido relativamente poco estudiada en la literatura.

Encontrar una solución que respete todas las restricciones del problema es NP-Hard. Esto incrementa el grado de dificultad de resolución respecto a otros problemas de ruteo de vehículos (para los cuales una solución factible puede conseguirse fácilmente), pues hace casi inevitable el tener que manipular soluciones no factibles durante la ejecución del algoritmo.

Proponemos dos algoritmos aproximados para resolver el VRPMT. Ambos están basados en la estrategia conocida como *Adaptive Memory Programming* (AMP) [112]. Este método consiste en realizar diversas ejecuciones de un algoritmo de búsqueda local partiendo de diferentes soluciones iniciales. Dichas soluciones se generan combinando componentes de las mejores soluciones encontradas durante la ejecución del AMP.

Los resultados obtenidos mediante la ejecución de los algoritmos sobre un conjunto de instancias de prueba sugieren que el método propuesto logra encontrar soluciones de buena calidad. Se encontraron soluciones para instancias del problema para las que los intentos anteriores no habían podido encontrar ninguna solución factible.

La motivación de este trabajo tiene dos raíces principales: el estudio de un problema de ruteo de vehículos que incorpore características importantes de la realidad y el desarrollo de técnicas para la resolución de problemas complejos desde el punto de vista computacional.

1.3. Estructura del documento

En el Capítulo 2 se realiza una reseña de algunos de los problemas de ruteo de vehículos que han sido más estudiados. Se dan formulaciones de estos problemas como problemas de programación entera. Asimismo, se resumen las ideas principales que han sido utilizadas en el diseño de algoritmos para su resolución, tanto a nivel de métodos exactos como de heurísticas y metaheurísticas.

En el Capítulo 3 se profundiza en el estudio de las metaheurísticas conocidas como *Tabu Search* y *Adaptive Memory Procedure*, que son la base del algoritmo de solución propuesto en este trabajo. Estas técnicas son introducidas mediante casos concretos de su aplicación para resolver versiones más simples del problema de ruteo de vehículos. Se enfatiza además sobre las principales decisiones de diseño que deben tomarse a la hora de adaptar estos métodos para resolver un problema concreto. También se realiza un resumen de aplicaciones de estos algoritmos para la resolución aproximada de otros problemas de optimización combinatoria.

El problema central de este trabajo, el VRPMT, es formulado en el Capítulo 4. Se define la notación utilizada, se dan dos formulaciones como problema de programación entera y los detalles sobre su complejidad computacional. Se presentan cotas para el valor de las soluciones del problema, así como algunas medidas alternativas de la calidad de las soluciones. Finalmente, se resumen los antecedentes sobre la resolución del problema encontrados en la literatura.

En el Capítulo 5 se presentan dos algoritmos basados en el AMP para resolver el problema. Además de dar una explicación detallada de los métodos empleados, se procura explicar cuáles son los características del problema que dificultan el diseño de métodos de solución eficientes. Finalmente, se marcan algunas diferencias importantes entre los

algoritmos diseñados y la propuesta original del AMP.

Los resultados obtenidos por los algoritmos se reportan en el Capítulo 6. El análisis realizado busca cuantificar experimentalmente el desempeño de los algoritmos en términos de la calidad de las soluciones encontradas y el tiempo de ejecución. Se realizan comparaciones con los resultados reportados por otros autores y también entre los dos algoritmos propuestos en este trabajo.

Finalmente, en el Capítulo 7 se presentan las conclusiones finales del trabajo y algunas ideas en las cuales se podría profundizar en el futuro.

Capítulo 2

Problemas de ruteo de vehículos

2.1. Introducción

En 1959, Dantzig y Ramser [38] realizaron por primera vez una formulación de un problema de ruteo de vehículos para una aplicación de distribución de combustible. Cinco años más tarde, Clarke y Wright [29] propusieron el primer algoritmo que resultó efectivo para su resolución: el popular Algoritmo de Ahorros. Desde entonces, el trabajo en el área ha sido intenso. Se han propuesto diversos modelos buscando capturar aspectos relevantes de los problemas reales. Asimismo, un gran número de algoritmos han sido diseñados para resolver esos modelos. Dada la complejidad computacional de los problemas de ruteo de vehículos, resulta muy difícil considerar a alguno de los algoritmos propuestos como el mejor, y es de esperar que el crecimiento del área continúe.

En este capítulo se realiza un relevamiento de los modelos y los algoritmos que han resultado más significativos en la literatura académica.

2.2. Modelos

2.2.1. Descripción de los problemas

Un problema de ruteo de vehículos consta de tres elementos principales: vehículos, clientes y depósitos. A grandes rasgos, el problema consiste en determinar un conjunto de rutas que comiencen y terminen en los depósitos, para que los vehículos visiten a los clientes. Existe una medida de costo (usualmente la distancia recorrida, el tiempo o una combinación de ambos) que es la que se busca minimizar. Las características de los vehículos, clientes y depósitos dan lugar a diferentes restricciones que, combinadas entre sí, generan diferentes variantes del problema.

Los Vehículos

Los vehículos suelen tener una capacidad limitada. Dicha capacidad puede tener una o varias dimensiones (por ejemplo, peso y volumen). Cada vehículo puede tener asociados tanto un costo fijo como un costo variable proporcional a la distancia que recorra. Los problemas en que los que todos los vehículos tienen los mismos atributos (capacidad, costo, etc.) se denominan de *flota homogénea*, y, en caso de haber diferencias, se trata de problemas de *flota heterogénea*.

Regulaciones legales podrían limitar el tiempo que un vehículo puede estar en circulación e incluso prohibir el pasaje de ciertos vehículos por ciertas zonas de la red vial. En algunos casos puede requerirse que la cantidad de trabajo realizado por los diferentes vehículos (por ejemplo, la cantidad de clientes visitados o el tiempo de viaje) no sea muy dispar.

La cantidad de vehículos disponibles podría ser un dato de entrada o una variable de decisión. En general se supone que cada vehículo recorre una sola ruta en el período de planificación, pero últimamente se han estudiado modelos en los que un mismo vehículo puede recorrer más de una ruta.

Los Clientes

Cada cliente tiene una demanda que debe ser satisfecha por algún vehículo. En muchos casos la demanda es un bien que ocupa lugar en los vehículos. El requerimiento puede ser distribuir la mercadería entre los clientes, o recolectar la mercadería ubicada en los clientes y transportarla hacia el depósito. También podría ocurrir que la mercadería deba ser transportada a los clientes pero no esté inicialmente en el depósito, sino distribuída en ciertos sitios proveedores. En este caso, los proveedores deben ser visitados antes que los clientes.

En otros casos el objeto de la demanda no es un bien sino un servicio: cada cliente simplemente debe ser visitado por el vehículo. Un mismo vehículo podría, potencialmente, visitar a todos los clientes. En otra variante del problema, cada cliente tiene una ubicación de origen y desea ser transportado hacia un sitio de destino. Aquí la capacidad del vehículo impone una cota sobre la cantidad de clientes que puede alojar simultáneamente.

Usualmente se exige que cada cliente sea visitado exactamente una vez. Sin embargo, en ciertos casos se acepta que la demanda de un cliente sea satisfecha en momentos diferentes y por vehículos diferentes.

Los clientes podrían tener restricciones relativas su horario de servicio. Usualmente estas restricciones se expresan en forma de intervalos de tiempo (llamados *ventanas de tiempo*) en los que un vehículo puede arribar al cliente.

En problemas con varios vehículos diferentes podría existir restricciones de compati-

bilidad entre éstos y los clientes. En estos casos, cada cliente sólo puede ser visitado por algunos de los vehículos (por ejemplo, algunos vehículos muy pesados no pueden ingresar en ciertas localidades).

Los Depósitos

Tanto los vehículos como las mercaderías a distribuir (si las hubiera) suelen estar ubicados inicialmente en depósitos. Es común exigir que cada ruta comience y finalice en un mismo depósito, aunque este podría no ser el caso en algunas aplicaciones (por ejemplo, podría ocurrir que el viaje deba comenzar o finalizar en el domicilio del conductor).

En los problemas con múltiples depósitos, cada uno podría tener diferentes características, como su ubicación y capacidad máxima de producción. Puede ocurrir, además, que cada depósito tenga una flota de vehículos asignada *a priori* o que dicha asignación sea parte de lo que se desea determinar. Suele exigirse que cada ruta comience y termine en el mismo depósito, aunque también puede permitirse rutas entre depósitos diferentes.

Los depósitos, al igual que los clientes, podrían tener ventanas de tiempo asociadas. En algunos casos debe considerarse el tiempo necesario para cargar o preparar un vehículo antes de que comience su ruta, o el tiempo invertido en su limpieza al regresar. Incluso, por limitaciones de los propios depósitos, puede ser necesario evitar que demasiados vehículos estén operando en un mismo depósito a la vez.

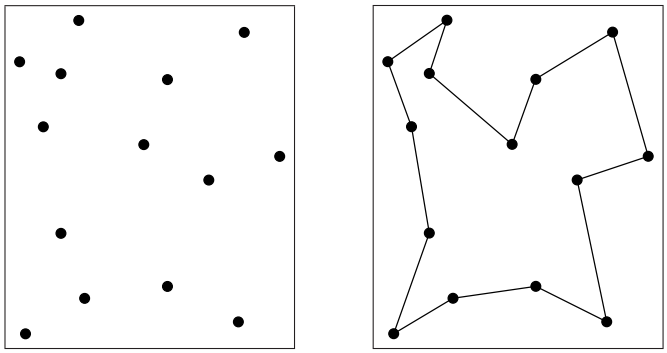
2.2.2. Formulación de algunos problemas clásicos

En esta sección se dan las modelos matemáticos asociados a algunos problemas clásicos de ruteo de vehículos (TSP, VRP, VRPTW, FSMVRP y VRPHF).

La componente geográfica de los problemas se modela a través de un grafo conexo $G = (V, E)$. El conjunto de nodos $V = \{0, 1, \ldots, n\}$ representa los sitios que participan en el problema, es decir, clientes y depósitos. La existencia de un arco $(i, j) \in E$ indica que es posible transportarse desde el sitio representado por i al sitio representado por j. Es usual que a cada arco $(i, j) \in E$ se le asocie un costo c_{ij} que indica la manera más económica de transportarse de i a j. En algunos casos se hace necesario asociar un tiempo de viaje t_{ij} a cada arco. Dado un nodo i se definen los conjuntos $\Delta^+(i) = \{j : (i, j) \in E\}$ y $\Delta^-(i) = \{j : (j, i) \in E\}$.

Una ruta es un ciclo simple en G con origen y destino en el depósito (en caso de que haya algún depósito en la formulación del problema), que representa la secuencia de visitas realizadas por el vehículo que recorre la ruta. El costo y el tiempo de una ruta se obtienen sumando los costos y tiempos de los arcos que forman el ciclo.

En la mayor parte de los casos G será un grafo completo, pues en una red de transporte real dados dos sitios cualesquiera existe una manera de transportarse de uno al otro. No obstante, modelar la red de transporte mediante un grafo permite codificar ciertas



(a) Una instancia del TSP (b) Una solución factible

Figura 2.1: Un TSP de ejemplo y una solución.

características o restricciones directamente en los datos del problema. Por ejemplo, si el uso de un vehículo tiene un costo fijo, puede sumarse dicho valor al costo de todos los arcos de la forma $(0, i)$ donde i es un cliente y 0 es el depósito.

Por simplicidad, los ejemplos serán presentados sobre *instancias euclídeas*. Es decir, sobre grafos completos en los cuales los nodos se corresponden con puntos en un plano y los costos están definidos por la distancia euclídea. Dada la simetría de ese tipo de costos, los grafos de los ejemplos serán no dirigidos y no se indicará explícitamente un sentido para los ciclos que representan las rutas. Debe tenerse en cuenta que dichas hipótesis son sólo a los efectos de simplificar la presentación de los ejemplos y los dibujos asociados.

El TSP

El *Travelling Salesman Problem* (TSP) consiste en encontrar un ciclo simple que visite todos los nodos del problema y cuyo costo total sea mínimo. En este problema no hay demandas y se cuenta con un solo vehículo. Tampoco existe un depósito (o, si existiera, no se distingue del resto de los nodos). En la Figura 2.1 se presenta una instancia del problema y una solución factible para la misma.

La siguiente formulación del TSP como problema de programación entera binaria fue propuesta por Dantzig, Fulkerson y Johnson [37] en 1954:

$$\text{(TSP)} \min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{2.1}$$

$$\text{s.a.} \sum_{j\in\Delta^+(i)} x_{ij} = 1 \qquad \forall\, i \in V \tag{2.2}$$

$$\sum_{i\in\Delta^-(j)} x_{ij} = 1 \qquad \forall\, j \in V \tag{2.3}$$

$$\sum_{i\in S, j\in\Delta^+(i)\setminus S} x_{ij} \geq 1 \quad \forall\, S \subset V, 2 \leq |S| \leq n-1 \tag{2.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in E$$

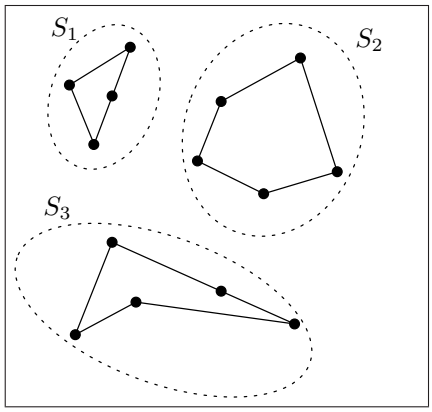


Figura 2.2: Una solución para la instancia de la Figura 2.1(a) formada por 3 sub-tours.

Las variables binarias x_{ij} indican si el arco (i, j) es utilizado en la solución ($x_{ij} = 1$) o no ($x_{ij} = 0$). La función objetivo (2.1) establece que el costo total de la solución es la suma de los costos de los arcos utilizados. Las restricciones (2.2) y (2.3) indican que la ruta debe llegar y abandonar cada nodo exactamente una vez. Finalmente, las restricciones (2.4) son llamadas *restricciones de eliminación de sub-tours* e imponen que todo subconjunto de nodos S debe ser abandonado al menos una vez. Si no se impusieran estas restricciones se estaría admitiendo soluciones que constan de más de un ciclo, como la que se muestra en la Figura 2.2. Esta solución está formada por tres sub-tours y viola las restricciones (2.4) para los conjuntos S_1, S_2 y S_3. La cantidad de restricciones de eliminación de sub-tours de la forma de (2.4) crece de manera exponencial con la cantidad de nodos del problema. Además de las restricciones propuestas en esta formulación, existen otras alternativas para eliminar los sub-tours [89].

Este es un problema de la clase NP-Hard, pues el problema de determinar si existe un ciclo hamiltoniano en un grafo (que es NP-Completo) puede reducirse polinomialmente al TSP [57]. Es NP-Hard inclusive si solamente se consideran instancias euclídeas.

El TSP es uno de los problemas de optimización combinatoria más difundidos y estudiados. Surge naturalmente en las áreas de transporte y logística, pero también encuentra campos de aplicación en otros contextos como la planificación de órbitas de satélites [7] y el estudio del genoma humano [4]. Además, suele utilizarse para el testeo y la comparación de métodos generales de solución para problemas de optimización, por ser de formulación simple y a la vez albergar gran complejidad computacional.

El VRP

En el *Vehicle Routing Problem* (VRP) el nodo 0 representa un depósito y los nodos $1, \ldots, n$ representan clientes. Cada cliente i tiene asociada una demanda $d_i > 0$. Se dispone de una flota de vehículos idénticos, cada uno de los cuales tiene capacidad $Q > 0$. En algunos casos se supone la existencia de infinitos vehículos, mientras que en otros, dicha

cantidad está acotada superiormente por un valor fijo m. El objetivo del problema es diseñar un conjunto de rutas de costo mínimo, de modo que:

(i) cada ruta comience y termine en el depósito,

(ii) cada cliente sea visitado por exactamente una ruta,

(iii) la demanda de los clientes visitados en una misma ruta no supere la capacidad Q.

En la formulación conocida con el nombre de *flujo de vehículos de dos índices*, se utilizan las variables binarias x_{ij} para determinar si el arco (i, j) se utiliza o no en la solución. Dado un conjunto de clientes S, $d(S) = \sum_{i \in S} d_i$ denota su demanda total y $r(S)$ indica la mínima cantidad de vehículos necesarios para visitarlos a todos respetando las restricciones del problema. El problema se formula de la siguiente manera [126]:

$$\text{(VRP1)} \min \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{2.5}$$

$$\text{s.a.} \sum_{j \in \Delta^+(i)} x_{ij} = 1 \qquad \forall\, i \in V \setminus \{0\} \tag{2.6}$$

$$\sum_{i \in \Delta^-(j)} x_{ij} = 1 \qquad \forall\, j \in V \setminus \{0\} \tag{2.7}$$

$$\sum_{i \in \Delta^+(0)} x_{0i} - \sum_{j \in \Delta^-(0)} x_{j0} = 0 \tag{2.8}$$

$$\sum_{i \in S, j \in \Delta^+(i) \setminus S} x_{ij} \geq r(S) \qquad \forall\, S \subset V \setminus \{0\}, S \neq \emptyset \tag{2.9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in E \tag{2.10}$$

Esta formulación surge como una extensión natural del modelo dado para el TSP. La función objetivo (2.5) mide el costo total de la solución. Las restricciones (2.6) y (2.7) aseguran que todo cliente es un nodo intermedio de alguna ruta. En (2.8) se impone que la cantidad de vehículos que sale del depósito coincida con la cantidad de vehículos que regresa. Finalmente, las restricciones (2.9) exigen que la cantidad de vehículos que abandonan un subconjunto de clientes S no sea menor que la cantidad de vehículos necesarios para transportar la demanda de los clientes en S. Con esto se logra eliminar los sub-tours y a la vez imponer que ninguna ruta sobrepase la capacidad de los vehículos.

En esta formulación se supone que la cantidad de vehículos disponibles no está acotada. Si hubiera una flota finita de m vehículos, se debe agregar la restricción

$$\sum_{i \in \Delta^+(0)} x_{0i} \leq m \tag{2.11}$$

que impone que no salgan más de m vehículos del depósito.

El valor de $r(S)$, utilizado como parámetro del problema en (2.9), puede obtenerse resolviendo un *Bin Packing Problem* (BPP) (ver Apéndice B.3) con un item de peso d_i por cada $i \in S$ y recipientes de capacidad Q. Esto resulta poco práctico desde el punto de vista de la formulación del problema, puesto que *a priori* se necesita una cantidad exponencial es estas restricciones y el BPP es un problema NP-Hard. De todos modos, la formulación anterior es válida aún cuando se sustituye $r(S)$ por la cota inferior dada por $\left\lceil \frac{d(s)}{Q} \right\rceil$, que el valor óptimo de la relajación lineal del BPP [126].

En otra alternativa para formular el VRP, propuesta por Balinsky y Quandt [9], se considera el conjunto R formado por todas las rutas que comienzan y terminan en el depósito y cuya demanda no excede la capacidad del vehículo. Para cada $r \in R$, c_r denota su costo y a_{ir} es un parámetro binario que indica si la ruta visita al cliente i o no. El problema se reduce a determinar $R' \subseteq R$ de modo que cada cliente sea visitado por exactamente una ruta de R', es decir, un *Set Partitioning Problem* (SPP) (ver Apéndice B.2):

$$\text{(VRP2)} \min \sum_{r \in R} c_r x_r \tag{2.12}$$

$$\text{s.a.} \sum_{r \in R} a_{ir} x_r = 1 \quad \forall\, i \in V \setminus \{0\} \tag{2.13}$$

$$x_r \in \{0,1\} \qquad \forall\, r \in R \tag{2.14}$$

Nuevamente, en esta formulación se supone que existe una cantidad infinita de vehículos. Para considerar el caso de un flota de m vehículos, debe exigirse que no se seleccionen más de m rutas, agregando la siguiente restricción al modelo

$$\sum_{r \in R} x_r \leq m.$$

El VRP es un problema NP-Hard [91]. En efecto, la versión con una cantidad fija de vehículos tiene al TSP como caso particular. Dada una instancia del TSP, se puede construir una instancia del VRP con cantidad fija de vehículos, considerando a uno cualquiera de los nodos como depósito, fijando todas las demandas en 0 y la cantidad de vehículos en 1. Cualquier solución de este problema es también una solución del TSP original. Como el TSP es NP-Hard, el VRP con una cantidad fija de vehículos también lo es. Finalmente, el VRP con una cantidad infinita de vehículos también lo es (ya que es equivalente al VRP con una cantidad fija de vehículos igual a n).

El VRPTW

En la variante del problema conocida como *Vehicle Routing Problem with Time Windows* (VRPTW), además de haber demandas y capacidades, cada cliente i tiene asociada una ventana de tiempo $[e_i, l_i]$ que establece un horario de servicio permitido para que un

vehículo arribe a él y un tiempo de servicio o demora s_i. Si t_i es la hora de arribo al cliente i, las ventanas de tiempo imponen que $t_i \leq l_i$, es decir, que no se arribe al cliente más allá del fin del horario de servicio. Además, si $t_i < e_i$ (o sea, si el vehículo llega antes de que comience el horario de servicio), debe esperar ocioso hasta la fecha e_i. Se supone que los vehículos parten del depósito en la hora 0.

Utilizando los nodos 0 y $n+1$ para representar al depósito y un conjunto K con los índices de los vehículos, el problema puede formularse de la siguiente manera [31]:

$$\text{(VRPTW)} \min \sum_{k \in K} c_{ij} \sum_{(i,j) \in E} x_{ij}^k \tag{2.15}$$

$$\text{s.a.} \sum_{k \in K} \sum_{j \in \Delta^-(i)} x_{ij}^k = 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{2.16}$$

$$\sum_{j \in \Delta^+(0)} x_{0j}^k = 1 \qquad \forall k \in K \tag{2.17}$$

$$\sum_{j \in \Delta^+(i)} x_{ij}^k - \sum_{j \in \Delta^-(i)} x_{ji}^k = 0 \qquad \forall k \in K,\ i \in V \setminus \{0, n+1\} \tag{2.18}$$

$$\sum_{i \in V \setminus \{0,n+1\}} d_i \sum_{j \in \Delta^+(i)} x_{ij}^k \leq Q \qquad \forall k \in K \tag{2.19}$$

$$y_j^k - y_i^k \geq s_i + t_{ij} - M(1 - x_{ij}^k) \qquad \forall k \in K, \forall (i,j) \in E \tag{2.20}$$

$$e_i \leq y_i^k \leq l_i \qquad \forall k \in K, \forall i \in V \tag{2.21}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall k \in K, \forall (i,j) \in E$$

$$y_i^k \geq 0 \qquad \forall k \in K, \forall i \in V$$

Esta formulación recibe el nombre de *flujo de vehículos de tres índices*. Las variables x_{ij}^k indican si el arco (i,j) es recorrido por el vehículo k. Las variables y_i^k indican la hora de comienzo de la atención en el cliente i cuando es visitado por el vehículo k (si el cliente no es visitado por dicho vehículo el valor de la variable no tiene significado). La función objetivo (2.15) es el costo total de las rutas. En (2.16) se impone que cada cliente debe ser visitado por exactamente un vehículo. Las restricciones (2.17) y (2.18) determinan que cada vehículo $k \in K$ recorre un camino de 0 a $n+1$. La capacidad de cada vehículo es impuesta en (2.19). Siendo M una constante lo suficientemente grande, la restricción (2.20) asegura que si un vehículo k viaja de i a j, no puede llegar a j antes que $y_i^k + s_i + t_{ij}$. Finalmente, los límites de las ventanas de tiempo son impuestos en (2.21).

El VRPTW es una generalización del VRP. En efecto, cualquier instancia del VRP puede transformarse en un VRPTW en el que todos los clientes tienen una ventana de tiempo lo suficientemente grande como para que todas las rutas sean factibles, por ejemplo, $[0, \sum_{i \in V \setminus \{0\}} s_i + \sum_{(i,j) \in E} t_{ij}]$. Dado que el VRP es un problema de la clase NP-Hard, el VRPTW es también NP-Hard.

El FSMVRP

El *Fleet Size and Mix Vehicle Routing Problema* (FSMVRP) es una extensión del VRP, en la que se elimina la hipótesis de que los vehículos son idénticos. En particular, los costos y capacidades de los vehículos varían, existiendo un conjunto $K = \{1, \ldots, |K|\}$ de tipos de vehículo. La capacidad de los vehículos de tipo $k \in K$ es q^k y su costo fijo es f^k. Los costos y tiempos de viaje para cada tipo de vehículo son c_{ij}^k y t_{ij}^k respectivamente. Se supone que los índices de los tipos de vehículo están ordenados en forma creciente por capacidad (es decir, $q^{k_1} \leq q^{k_2}$ si $k_1, k_2 \in K, k_1 < k_2$).

La siguiente es una adaptación de la formulación de flujo de vehículos de tres índices:

$$\text{(FSMVRP)} \quad \min \sum_{k \in K} f^k \sum_{j \in \Delta^+(0)} x_{0j}^k + \sum_{k \in K} \sum_{(i,j) \in E} c_{ij}^k x_{ij}^k \tag{2.22}$$

$$\text{s.a.} \sum_{k \in K} \sum_{i \in \Delta^+(j)} x_{ij}^k = 1 \qquad \forall\, j \in V \setminus \{0\} \tag{2.23}$$

$$\sum_{j \in \Delta^+(i)} x_{ij}^k - \sum_{j \in \Delta^-(i)} x_{ji}^k = 0 \qquad \forall\, i \in V, \forall\, k \in K \tag{2.24}$$

$$r_0 = 0 \tag{2.25}$$

$$r_j - r_i \geq (d_j + q^{|K|}) \sum_{k \in K} x_{ij}^k - q^{|K|} \qquad \forall (i,j) \in E, i \neq 0 \tag{2.26}$$

$$r_j \leq \sum_{k \in K} q^k \sum_{i \in \Delta^-(j)} x_{ij}^k \qquad \forall\, j \in V \setminus \{0\} \tag{2.27}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall\, (i,j) \in E, \forall\, k \in K$$

$$r_i \geq 0 \qquad \forall\, i \in V \setminus \{0\}$$

Las variables binarias x_{ij}^k indican si el arco (i,j) es utilizado por algún vehículo de tipo k y las variables r_i positivas indican la carga acumulada en la ruta correspondiente hasta el nodo i (inclusive). La función objetivo (2.22) mide el costo total de la solución incluyendo costos fijos y variables. Las restricciones (2.23) establecen que todo cliente debe ser visitado por exactamente un vehículo. En (2.24) se indica que si un vehículo de tipo k visita al nodo i, entonces un vehículo del mismo tipo debe abandonarlo. Las restricciones (2.25) y (2.26) fijan los valores de las variables r_i y actúan como restricciones de eliminación de subtours, mientras que la capacidad de los vehículos se impone en (2.27).

En esta formulación se supone que la cantidad de vehículos de cada tipo es ilimitada. No solo se debe decidir las rutas, sino la composición de la flota de vehículos a utilizar (es decir, cuántos vehículos de cada tipo). Usualmente, al tratar con problemas de flota heterogénea, se opta por utilizar este modelo aunque en algunos casos no refleja la realidad.

Si la cantidad de vehículos disponibles de cada tipo k fuera m_k, conocida de antemano,

debería agregarse las siguientes restricciones al modelo anterior

$$\sum_{j \in \Delta^+(0)} x_{0j}^k \leq m_k \quad \forall\, k \in K.$$

Esta versión se conoce como *Vehicle Routing Problem with Heterogeneous Fleet* (VRPHF).

Dado que el VRP puede modelarse como un problema con flota heterogénea en el cual hay un solo tipo de vehículos, tanto el FSMVRP como el VRPHF son problemas NP-Hard.

2.3. Métodos de solución

Desde que los problemas de ruteo de vehículos fueron formulados por primera vez, decenas de métodos han sido propuestos para su resolución. Estos métodos pueden separarse en dos grandes categorías: algoritmos exactos y algoritmos aproximados (o heurísticas). Los algoritmos exactos encuentran siempre una solución óptima y su principal desventaja es que, dado que se trata de problemas NP-Hard, suelen requerir tiempos de ejecución muy elevados. Esto dificulta su aplicación en la práctica. Las heurísticas, en cambio, devuelven una solución que no necesariamente es óptima, pero en tiempos de ejecución más moderados (o al menos más controlables).

En lo que queda de este capítulo se realiza un resumen de las técnicas principales para resolver problemas de ruteo de vehículos. A no ser que se indique lo contrario, los algoritmos se presentan para solucionar el VRP. Se pretende dar un panorama general de las diferentes ideas, más que un estudio sistemático de cada una de ellas. Existen compendios más completos y profundos en la literatura [28, 15, 89, 73, 85, 62, 35, 31, 126, 33, 32]. En el diagrama de la Figura 2.3 se muestran los métodos de solución analizados en esta sección.

2.4. Métodos exactos

Los métodos exactos tienen la propiedad de que devuelven una solución óptima siempre que se los ejecute hasta su terminación. El inconveniente suele ser que el tiempo necesario para que estos algoritmos finalicen puede ser prohibitivo en la práctica. Sin embargo, algunos de estos algoritmos pueden obtener una solución factible (no necesariamente óptima) en algún paso intermedio de su ejecución. En estos casos, la ejecución del algoritmo puede finalizarse antes de tiempo y aún así contar con una solución, comportándose el algoritmo como una heurística.

La mayor parte de los métodos exactos propuestos para problemas de ruteo de vehículos pueden clasificarse en tres categorías: métodos arborescentes, programación dinámica y generación de columnas. Relevamientos de métodos exactos pueden encontrarse en los trabajos de Laporte y Norbert [89], Laporte [85] y Toth y Vigo [125].

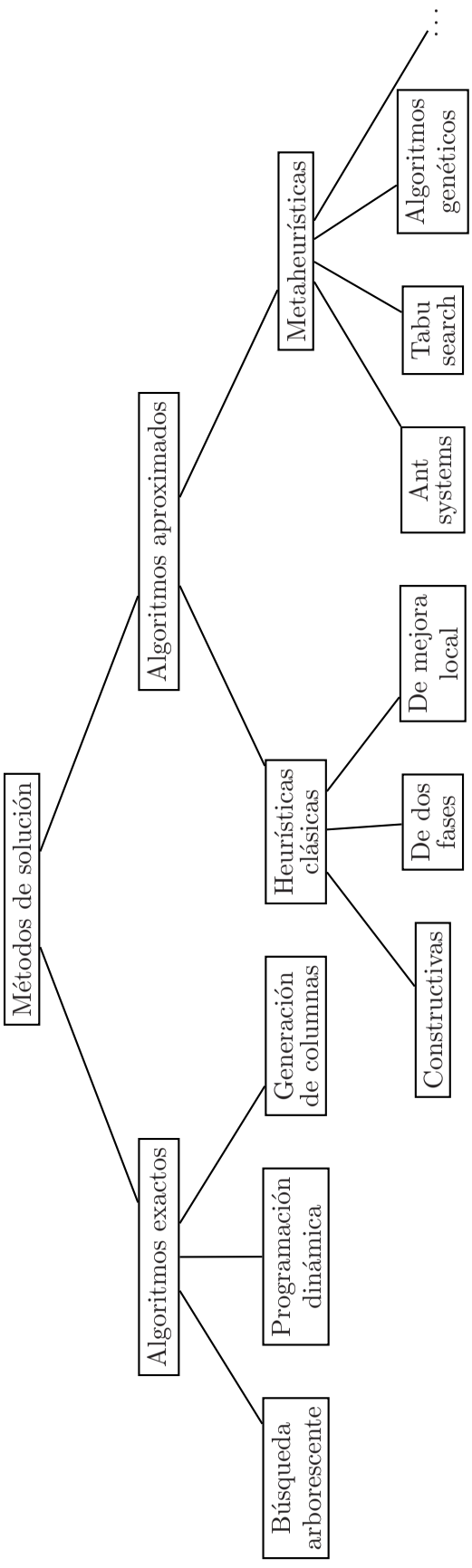


Figura 2.3: Una clasificación de los métodos de solución.

Paso 1 (inicialización).
Hacer $k \leftarrow 0$.
Agregar el problema original P_0 al conjunto de pendientes $\mathcal{L} \leftarrow \{P_0\}$.
Si se conoce alguna solución factible s_0 (de valor z_0) para el problema original, hacer $s^{\text{best}} \leftarrow s_0$ y $z^{\text{best}} \leftarrow z_0$; si no, dejar indefinida a s^{best} y hacer $z^{\text{best}} \leftarrow \infty$.

Paso 2 (terminación).
Si $\mathcal{L} = \emptyset$, devolver s^{best} y terminar.
Si no, hacer $k \leftarrow k + 1$, extraer un problema $P_k \in \mathcal{L}$ y hacer $\mathcal{L} \leftarrow \mathcal{L} \setminus \{P_k\}$.

Paso 3 (relajación del subproblema).
Hallar una solución óptima de una relajación de P_k.
Sea $\bar{s_k}$ dicha solución y $\bar{z_k}$ su valor.

Paso 4 (poda).
Si $\bar{z_k} \geq z^{\text{best}}$, ir al paso 2 (la solución de P_k no es mejor que s^{best}).

Paso 5 (ramificación).
Determinar si $\bar{s_k}$ es factible para P_k.
Si lo es, hacer $z^{\text{best}} \leftarrow \bar{z_k}$ y $s^{\text{best}} \leftarrow \bar{s_k}$.
Si no, particionar P_k en subproblemas e insertarlos en $\mathcal{L}$.
En cualquier caso, ir al paso 2.

Figura 2.4: Algoritmo *branch and bound*.

2.4.1. Métodos arborescentes

Este tipo de métodos están basados en la estrategia de ramificación y acotamiento o *branch and bound* [100]. Se resuelve una relajación del problema (obteniendo una cota inferior para el valor óptimo) y en caso de que la solución encontrada no sea factible para el problema original, el problema se particiona en subproblemas, cada uno de los cuales se resuelve de la misma manera. Las cotas inferiores obtenidas son utilizadas para *podar* la búsqueda, es decir, no resolver subproblemas que no mejorarán la mejor solución obtenida hasta cierto momento. Un esquema general de estos algoritmos se muestra en la Figura 2.4.

La ejecución de este tipo de algoritmos puede pensarse como una recorrida por un árbol enraizado en el cual cada subproblema es un nodo y los subproblemas en los que este es particionado son sus hijos. En la Figura 2.5 se ilustra este aspecto. En este ejemplo, al particionar cada problema se generan tres subproblemas. En el problema P_3 se cumple el criterio de poda en el paso 4 del algoritmo, y por lo tanto no tiene hijos en el árbol.

Un aspecto importante para el buen desempeño de estos algoritmos, es que las relajaciones que se resuelven den cotas ajustadas para el valor óptimo de cada subproblema, pues si eso ocurre, es de esperar que se cumpla repetidas veces el criterio de poda del paso 4. Por otro lado, es deseable que los subproblemas puedan resolverse de manera relativamente eficiente. Estos dos aspectos suelen representar objetivos encontrados y lo difícil es encontrar una buena solución de compromiso. Obtener cotas inferiores ajustadas para el

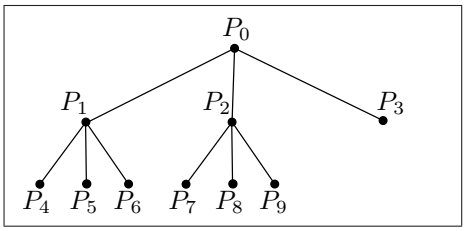


Figura 2.5: Árbol de ejecución de un algoritmo branch and bound.

VRP es un problema complejo. Cabe mencionar las basadas en *k-degree center trees* y en *q-routes* propuestas por Christofides [26].

Little et al. [93] propusieron un algoritmo de este estilo para resolver el TSP. El algoritmo se basa en la formulación dada en la Sección 2.2.2, la cual admite al *Generalized Assignment Problem* (GAP) [100] y al *Minimum Spanning Tree Problem* (MST) [100] como relajaciones. Las particiones en subproblemas se determinan fijando la presencia y la ausencia de cierto arco. Es decir, para cierto arco $(i, j) \in E$, en un subproblema se fija $x_{ij} = 1$ y en el otro $x_{ij} = 0$, generando dos subproblemas en cada ramificación. Christofides y Eilon [25] adaptaron dicha formulación al VRP y aplicaron un algoritmo similar.

Fisher y Jaikumar [50] propusieron un algoritmo basado en la formulación de tres índices similar a la dada en (VRPTW) para un VRP con restricciones adicionales. El algoritmo utiliza una descomposición de Benders [95], en la que se asignan clientes a los vehículos por medio de un GAP y luego se calculan las rutas para cada vehículo resolviendo un TSP con ventanas de tiempo.

Laporte et al. [88] propusieron un algoritmo para problemas con costos simétricos, que admiten formulaciones más compactas. En el algoritmo se relajan las restricciones de eliminación de sub-tours y se utiliza un método de planos de corte [100] dentro de un esquema de branch and bound.

Christofides [24] propuso otro tipo de algoritmo de búsqueda arborescente en el cual en cada nivel del árbol de búsqueda se tiene una solución parcial en la que ciertos clientes no pertenecen a ninguna ruta. Para pasar de un nivel a otro se considera cada cliente que no esté en ninguna ruta, junto con todas las posibles rutas en las que éste podría ser insertado de manera factible (además de la creación de una nueva ruta para el cliente). Además, se da un conjunto de criterios para podar la búsqueda.

Entre las diversas propuestas de este tipo para problemas de ruteo de vehículos se encuentran las de Laporte et al. [86, 87], Fisher [49], Fischetti et al. [48] y Miller [97].

2.4.2. Programación dinámica

Eilon et al. [46] formularon el VRP como un problema de programación dinámica. Dado un conjunto de clientes S, se define $c(S)$ como el costo de la mejor ruta que visita a los clientes de S y al depósito (si la demanda supera la capacidad del vehículo $c(S) = \infty$).

El mínimo costo para visitar al conjunto de clientes S con a lo sumo k vehículos puede calcularse mediante la siguiente ecuación de recurrencia:

$$f_k(S) = \begin{cases} c(S) & \text{si } k = 1 \\ \min_{S^* \subseteq S} \left(f_{k-1}(S \setminus S^*) + c(S^*) \right) & \text{si } k > 1 \end{cases}$$

Esta formulación general no resulta eficiente en la práctica, pues la cantidad de estados a evaluar crece en forma exponencial con el tamaño del problema. En efecto, existen $2^{|S|}$ alternativas para S^* a explorar en cada nivel de la recurrencia. Christofides et al. [27] utilizan esta formulación junto con una relajación del espacio de estados para obtener cotas inferiores del valor óptimo. Si bien con este método no se resuelve el problema, puede utilizarse dentro de un método de búsqueda arborescente para obtener las cotas inferiores.

2.4.3. Métodos de generación de columnas

Los métodos de generación de columnas se utilizan para resolver problemas de programación lineal con una gran cantidad de variables continuas:

$$\begin{aligned} \text{(LP)} \min \quad & c^T x \\ \text{s.a.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

donde $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ y $c \in \mathbb{R}^n$ son los parámetros y $x \in \mathbb{R}^n$ son las variables del problema. Cuando la cantidad de variables crece en forma exponencial con el tamaño del problema (de manera similar a lo que ocurre con en la formulación VRP2), la cantidad de columnas de la matriz A puede hacer que formular explícitamente el problema no sea viable.

En los métodos de generación de columnas se trabaja sobre dos problemas: el problema maestro restringido y el subproblema. El problema maestro restringido se construye a partir del problema original (LP) considerando un subconjunto de sus columnas. Dada una solución factible del problema maestro restringido (que es también factible en el problema original), el subproblema consiste en encontrar una columna de LP cuyo costo reducido sea mínimo. Si el subproblema encuentra una columna de costo reducido no negativo, la última solución obtenida del problema maestro restringido es óptima para el problema original. En caso contrario, la columna obtenida se agrega al problema maestro y éste se vuelve a resolver. Un esquema del método de generación de columnas se muestra en la Figura 2.6.

Si se utiliza un método iterativo para resolver los problemas maestro restringido (como el método Simplex [95]), suele ser ventajoso comenzar la resolución del problema M_{k+1}

Paso 1 (inicialización).
Inicializar el problema maestro restringido M_0 con un subconjunto de las columnas de la matriz A. Hacer $k \leftarrow 0$.

Paso 2 (problema maestro).
Resolver el problema maestro restringido M_k. Sea $\bar{x_k}$ la solución obtenida.

Paso 3 (generación de una columna).
Resolver el subproblema. Sea $\bar{a_k}$ la columna óptima y $\bar{\pi_k}$ su costo reducido.

Paso 4 (terminación).
Si $\bar{\pi_k} \geq 0$, terminar ($\bar{x_k}$ es la solución óptima del problema original).
Si no, agregar la columns a_k al problema M_k (sea M_{k+1} el nuevo problema), hacer $k \leftarrow k + 1$ e ir al paso 2.

Figura 2.6: Método de generación de columnas.

partiendo de $\bar{x_k}$, es decir, la solución óptima del problema anterior. El intercambio de información entre el problema maestro restringido y el subproblema se ilustra en la Figura 2.7. Para resolver problemas de programación entera, esta estrategia suele incorporarse a un método de búsqueda arborescente.

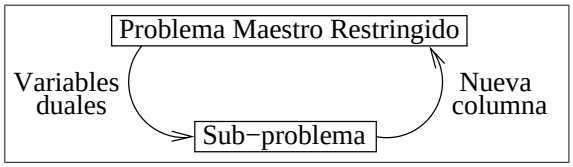


Figura 2.7: Pasaje de información en los métodos de generación de columnas.

Agarwal [3] propuso un algoritmo para resolver la formulación VRP2, en el cual el problema maestro es un SPP y cada subproblema se resuelve como un problema de la mochila. En los trabajos que consideran ventanas de tiempo [102, 43, 41] el subproblema consiste en hallar un camino más corto respetando ventanas de tiempo.

2.5. Algoritmos aproximados

Dados los elevados tiempos necesarios para ejecutar los métodos exactos, los algoritmos aproximados o heurísticas surgen como una alternativa más práctica para resolver los problemas. Estas técnicas se basan en diversas ideas sobre cómo buscar buenas soluciones. Algunas se inspiran en sistemas naturales y otras se basan en la experiencia e intuición de sus creadores sobre cómo hacer más efectiva la búsqueda. Mediante el uso de heurísticas se ha conseguido resolver instancias con miles de nodos e incorporar de manera sencilla diversos tipos de restricciones. El precio que se paga por obtener tiempos de ejecución más bajos es la falta de una garantía de optimalidad para las soluciones encontradas.

Los algoritmos aproximados que han sido propuestas para resolver problemas de ruteo de vehículos pueden clasificarse en dos categorías: heurísticas clásicas y metaheurísticas.

Heurísticas clásicas. Estos métodos realizan una exploración muy limitada del espacio de soluciones. En general son procedimientos constructivos de una pasada o métodos simples de búsqueda local. Requieren tiempos de ejecución extremadamente bajos.

Metaheurísticas. En esta categoría se encuentran métodos que buscan realizar una exploración más efectiva del espacio de soluciones. Este tipo de algoritmos suelen utilizar procedimientos constructivos, algoritmos de búsqueda local, estrategias de combinación de soluciones y estructuras memoria, entre otros aspectos. La necesidad de recursos computacionales es más intensa en este tipo de procedimientos que en las heurísticas clásicas.

Gendreau et al. [63] sugieren un diagrama similar al de la Figura 2.8 para dar una idea del compromiso entre performance y tiempo de ejecución obtenidos por cada una de estas clases de heurísticas. Las heurísticas clásicas logran obtener resultados relativamente buenos en tiempos de ejecución muy bajos. Los mejores resultados conocidos son obtenidos, en general, por las metaheurísticas, incurriendo en altos tiempos de ejecución.

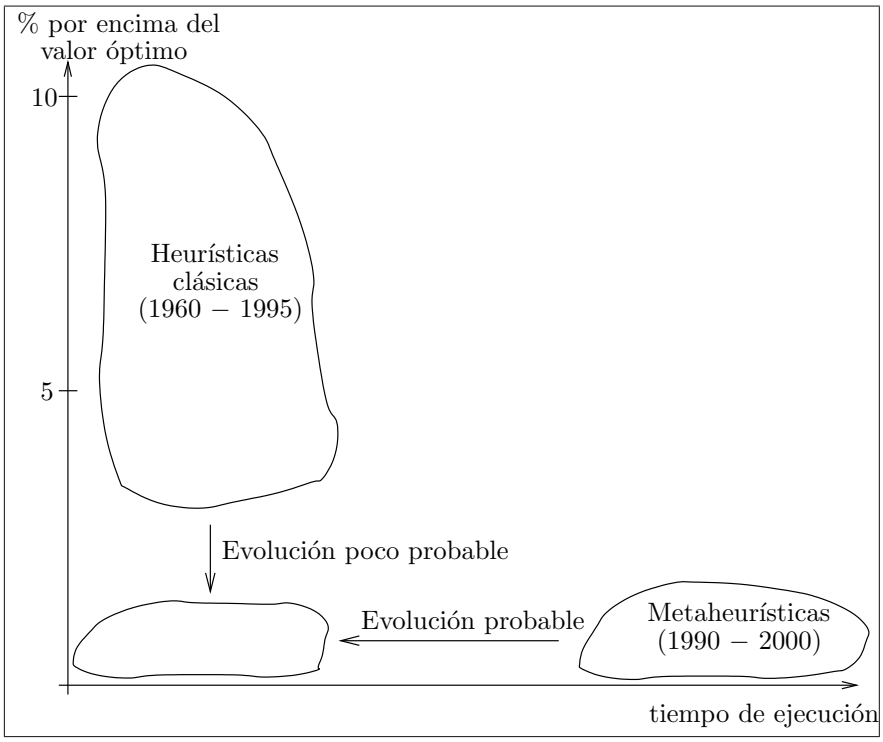


Figura 2.8: Diagrama comparativo propuesto por Gendreau et al. [63].

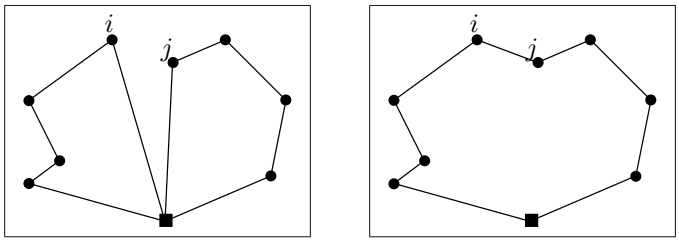


Figura 2.9: Dos rutas antes y después de ser unidas.

2.5.1. Heurísticas clásicas

Dentro de las heurísticas clásicas, existen a su vez tres grandes grupos de algoritmos: los procedimientos constructivos, los de dos fases y los de mejora local. En los trabajos de Bodin et al. [15] y Laporte y Semet [90] se da un completo resumen de este tipo de métodos.

Procedimientos constructivos

En los procedimientos constructivos se crea una solución de manera gradual. Suele trabajarse sobre una solución parcial (por ejemplo, un conjunto de rutas que no visita a todos los clientes) que es aumentada en cada iteración según una regla ávida.

El más popular de los algoritmos para resolver el VRP, el *algoritmo de ahorros* propuesto por Clarke y Wright [29], pertenece a esta clase. Si en una solución dos rutas diferentes $(0, \ldots, i, 0)$ y $(0, j, \ldots, 0)$ pueden ser combinadas formando una nueva ruta $(0, \ldots, i, j, \ldots, 0)$ como se muestra en la Figura 2.9, el ahorro obtenido por realizar dicha unión es

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}$$

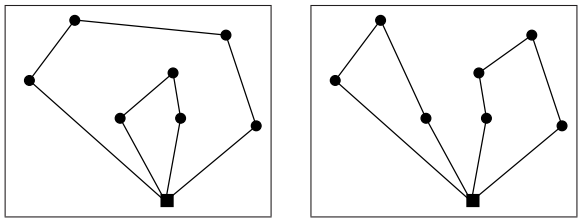
pues en la nueva solución los arcos $(i, 0)$ y $(0, j)$ no serán utilizados y se agregará el arco (i, j).

Partiendo de una solución inicial formada por rutas unitarias de la forma $(0, i, 0)$, se realizan las uniones de rutas que den mayores ahorros siempre que se satisfagan las restricciones del problema. Existe una versión paralela en la que se trabaja sobre todas las rutas simultáneamente, y otra secuencial que construye las rutas de a una por vez.

Se ha observado que utilizando la definición original de s_{ij} suele generarse algunas rutas circulares (ver Figura 2.10) lo cual puede ser negativo en algunos casos. Para solucionar este problema algunos autores [136, 72, 58] proponen redefinir el ahorro como

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$$

donde λ es un parámetro que penaliza la unión de rutas con clientes lejanos (llamado parámetro de forma o *shape parameter*). Dicho parámetro puede utilizarse también para



(a) Rutas Circulares (b) Rutas Radiales

Figura 2.10: Un ejemplo de rutas circulares y radiales.

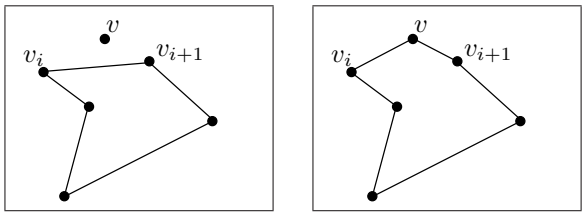


Figura 2.11: Inserción del cliente v entre dos nodos consecutivos v_i y v_{i+1}.

generar un conjunto de soluciones diferentes mediante la ejecución repetida del algoritmo con diferentes valores de λ.

Existen variantes de este algoritmo, como las basadas en *matching* [42, 6, 132] y las adaptaciones para resolver el VRPTW [114] y el FSMVRP [74].

Otro tipo de métodos constructivos son las llamadas *heurísticas de inserción*. En cada iteración se tiene una solución parcial cuyas rutas sólo visitan un subconjunto de los clientes y se selecciona uno de los clientes no visitados, para insertar en dicha solución. Para disminuir la cantidad de alternativas (y, por ende, el tiempo de ejecución) suele considerarse sólamente la inserción de clientes entre nodos consecutivos, como se muestra en la Figura 2.11.

En las heurísticas de inserción secuencial [99, 114, 45] sólo se considera insertar clientes en la última ruta creada. Cuando ninguna inserción en esa ruta es factible, se crea una nueva. La principal desventaja de este enfoque es que los últimos clientes no visitados tienden a estar dispersos y por lo tanto las últimas rutas construídas son de costo muy elevado. Las heurísticas de inserción en paralelo [28, 108] surgen para remediar esta deficiencia. Al comienzo se crea un conjunto de rutas unitarias y luego se consideran las inserciones de los clientes no visitados en cualquiera de las rutas que componen la solución.

Las *inserciones generalizadas* [59, 61] (GENI, por *GENeralized Insertions*) surgen dentro de un método de solución del TSP y tienen como principal característica que permiten insertar clientes entre nodos no consecutivos de una misma ruta. Se proponen dos maneras para insertar un cliente v entre los nodos v_i y v_j (no necesariamente consecutivos). Ambas inserciones se ilustran en las Figuras 2.12 y 2.13.

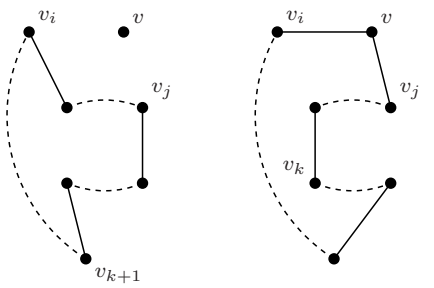


Figura 2.12: Inserción GENI de Tipo I.

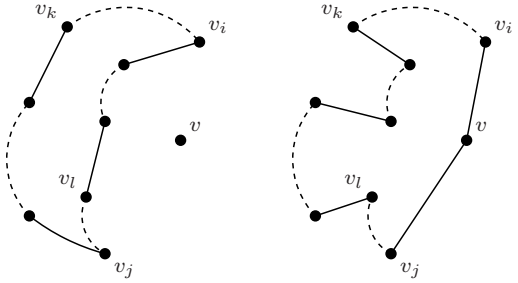


Figura 2.13: Inserción GENI de Tipo II.

Procedimientos de dos fases

En los procedimientos de dos fases, se descompone el problema en 2 subproblemas que se resuelven de manera secuencial.

En los métodos *asignar primero y rutear después* primero se busca particionar el conjunto de clientes en subconjuntos, también llamados *clusters*, que estarán en una misma ruta en la solución final. Luego, para cada cluster se crea una ruta que visite a todos sus clientes. Las restricciones de capacidad son consideradas en la primera etapa, asegurando que la demanda total de cada cluster no supere la capacidad del vehículo. Por lo tanto, construir las rutas para cada cluster implica resolver un TSP que, dependiendo de la cantidad de clientes, se puede resolver el forma exacta o aproximada. En la *heurística de barrido* [133, 134, 64], los clusters se forman girando una semirrecta con origen en el depósito e incorporando los clientes "barridos" por dicha semirrecta hasta que se viole la restricción de capacidad. Solomon [114] propuso una extensión de este algoritmo para resolver el VRPTW. Fisher y Jaikumar [50] proponen generar los clusters resolviendo un problema de asignación, mientras que Bramel y Simchi-Levi [17] los generan resolviendo un problema de localización con capacidades.

La estrategia conocida como *rutear primero y asignar después* [10] propone en una primera fase, calcular una única ruta que visite a todos los clientes resolviendo un TSP. Es de esperar que esa ruta no respete la restricción de capacidad. Por lo tanto, se busca particionarla en varias rutas de modo que cada una de ellas sí sea factible. El problema de particionar la ruta se puede modelar como un problema de hallar el camino más corto en un grafo. Golden et al. [74] dan una extensión de este método para resolver el FSMVRP.

Los *algoritmos de pétalos* [53, 113, 111] explotan la formulación VRP2 dada en la

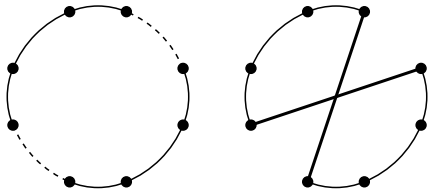


Figura 2.14: El único 2-intercambio posible para los arcos marcados.

Sección 2.2.2. En una primera fase se genera un conjunto con gran cantidad de rutas, cada una de las cuales satisface la restricción de capacidad (es posible que un mismo cliente sea visitado por varias rutas). Luego, se busca seleccionar un subconjunto de rutas de modo que cada cliente sea visitado exactamente una vez, lo cual puede lograrse resolviendo un SPP. Si las rutas se generan como en el algoritmo de barrido el SPP resultante puede resolverse en tiempo polinomial en la cantidad de rutas generadas [14].

Procedimientos de mejora local

Estos algoritmos se basan en la definición de modificaciones simples (también llamadas *movidas*) que, dada una solución, podrían resultar en otra de menor costo. Usualmente las movidas implican cambiar la ubicación de algunos nodos en las rutas y la variación ocasionada en el costo de la solución a causa de aplicar la movida requiere realizar unos pocos cálculos sencillos (y no evaluar completamente la función objetivo). Dada una solución inicial, se busca una movida que disminuya su costo. Si se encuentra, se realiza y se continúa la búsqueda partiendo de esa nueva solución. Si todas las movidas incrementan el costo, la solución es localmente óptima.

Uno de los operadores de búsqueda local más conocidos es el λ-intercambio definido por Lin [92]. Un λ-intercambio consiste en eliminar λ arcos de la solución (donde $\lambda > 1$) y reconectar los λ segmentos resultantes. Cada movida modifica una única ruta de la solución. En las Figuras 2.14 y 2.15 se muestran los 2-intercambios y 3-intercambios posibles para una ruta, respectivamente. Estas movidas pueden invertir el orden de algunas visitas[1]. Suponiendo que esas inversiones no afectan la factibilidad de las rutas y que los costos son simétricos, puede buscarse movidas que mejoren el costo de una ruta sin necesidad de explorar y evaluar todas las posibilidades [83]. Un algoritmo que realiza λ-intercambios suele denominarse λ-opt.

Renaud, Boctor y Laporte [110] propusieron una versión simplificada de 4-opt que explora un subconjunto de los 4-intercambios, reduciendo la cantidad de movidas a explorar y el tiempo de ejecución.

Una versión reducida del algoritmo 3-opt es el algoritmo Or-opt [101], que consiste en

[1] Es decir, $(\ldots, v_i, v_{i+1}, \ldots)$ pasa a ser $(\ldots, v_{i+1}, v_i, \ldots)$.

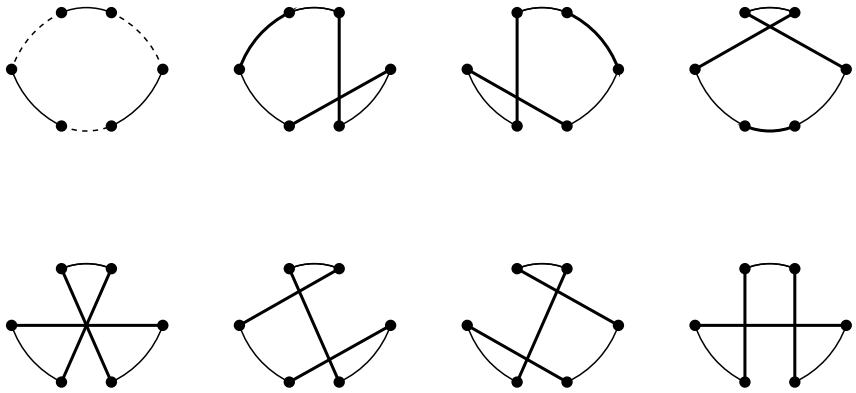


Figura 2.15: Todos los 3-intercambios posibles para los arcos marcados.

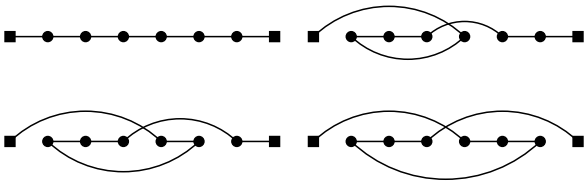


Figura 2.16: Movidas para reubicar los 3 primeros clientes de una ruta.

eliminar una secuencia de k clientes consecutivos de la ruta y colocarlos en otra posición de la ruta, de modo que permanezcan consecutivos y en el mismo orden. Primero se realizan las movidas con $k = 3$, luego con $k = 2$ y finalmente con $k = 1$. En la Figura 2.16 se muestra una ruta y todas las posibles maneras de reubicar los 3 primeros clientes a la manera del algoritmo Or-opt.

El procedimiento GENIUS (*GENI Unstringing and Stringing*) [59] consiste en realizar alternadamente eliminaciones e inserciones generalizadas. Se comienza considerando el primer cliente de la ruta. Se efectúa la eliminación del estilo GENI para este cliente que reduce más el costo de la ruta. Luego se realiza la mejor inserción generalizada de ese cliente en la ruta resultante. Si la ruta resultante es de menor costo que la original, se vuelve a comenzar con el primer cliente de la nueva ruta. Si no es mejorada, se repite el proceso con el segundo cliente de la nueva ruta. El proceso termina luego de eliminar e insertar el último cliente de la ruta. El proceso de eliminar clientes e insertarlos nuevamente se denomina *Unstringing and Stringing* (US).

Van Breedam [128] propuso dos movidas que intercambian clientes entre dos rutas eventualmente diferentes (a diferencia de todas las propuestas anteriores). En las movidas *string relocation* (SR), una secuencia de nodos es transferida de una ruta a la otra manteniendo el orden que tenían en la ruta original. En las movidas *string exchange* (SE), ambas rutas intercambian los clientes. En las Figuras 2.17(a)-(b) se muestra una movida

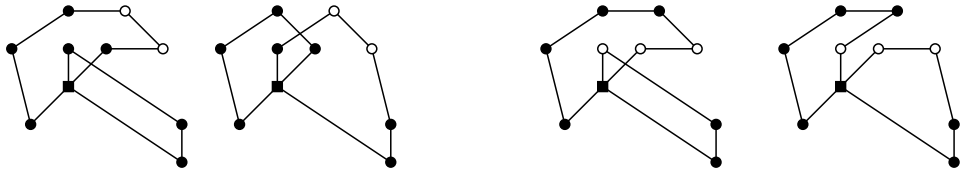


Figura 2.17: Movidas String Relocation y String Exchange.

SR y en 2.17(c)-(d) se ilustra una movida SE.

2.5.2. Metaheurísticas

Para obtener mejores soluciones que las heurísticas clásicas, es necesario recurrir a técnicas que realicen una exploración más adecuada del espacio de soluciones. Las metaheurísticas son procedimientos genéricos de exploración del espacio de soluciones para problemas de optimización y búsqueda que buscan cumplir con ese objetivo. Proporcionan ideas generales que, adaptadas a cada contexto, permiten construir algoritmos de solución. En general, las metaheurísticas obtienen mejores resultados que las heurísticas clásicas, pero demandan más recursos computacionales y tiempo de ejecución (que, de todos modos, no son comparables con las exigencias de los algoritmos exactos). Por una descripción de las metaheurísticas a nivel general puede consultarse el Apéndice A. En lo que queda de este capítulo se comentan algunas aplicaciones de metaheurísticas para resolver problemas de ruteo de vehículos. Se sugiere consultar los trabajos de Golden et al. [76], Gendreau et al. [63] y Cordeau et al. [32], en los que se realizan descripciones más detalladas.

Tabu search

La técnica *Tabú Search* (TS) fue propuesta por Glover [66] y consiste en realizar una búsqueda local aceptando soluciones que deterioran el valor de la función objetivo. Esa estrategia permite escapar de los óptimos locales. Como contrapartida, luego de realizar una movida debe procurarse no retornar a la solución anterior (en general, debe evitarse las últimas soluciones visitadas) para que el algoritmo no entre en un ciclo. En TS, ese objetivo se logra utilizando una memoria de corto plazo en la cual se almacenan ciertas movidas denominadas *tabú*. Dichas movidas no pueden realizarse hasta que no dejen de ser consideradas tabú.

En el algoritmo propuesto por Osman [103] la vecindad de una solución se define mediante intercambios de clientes entre pares de rutas. Luego de realizar una movida, se aplica el algoritmo 2-opt [92] sobre cada una de las rutas implicadas.

Gendreau, Hertz y Laporte [60] propusieron un algoritmo de búsqueda tabú para el VRP llamado Taburoute. En esta propuesta se acepta que las soluciones violen la restricción de capacidad. Dichas violaciones son penalizadas en la función objetivo utilizada

para seleccionar las movidas. Las movidas se definen mediante inserciones generalizadas (GENI) [59]. Si un cliente es eliminado de una ruta, volver a insertarlo en es una movida tabú durante una cantidad de iteraciones que se sortea uniformemente en un intervalo (que es un parámetro del algoritmo). Como método de diversificación se penaliza la repetición sistemática de movidas sobre los mismos clientes. Al considerar el costo de una movida en la que participa cierto cliente, se suma un término proporcional a la cantidad de veces que dicho cliente fue movido durante la ejecución del algoritmo. Cordeau, Laporte y Mercier [34] adaptaron este algoritmo para resolver el VRPTW, utilizando una versión de las inserciones generalizadas que permite contemplar ventanas de tiempo [61].

En el algoritmo de Taillard [116] se realiza una partición del conjunto de clientes y cada partición se resuelve como un VRP independiente de los demás mediante TS. Se utiliza la misma definición de vecindad que en el algoritmo de Osman [103]. La inversa de una movida es considerada tabu por una cantidad de iteraciones que se sortea uniformemente en un intervalo que depende de la cantidad de clientes del problema. Periódicamente, cada ruta es optimizada resolviendo un TSP sobre sus clientes de forma exacta mediante el algoritmo de Volgenant y Jonker [129].

En el algoritmo propuesto por Xu y Kelly [135] se utiliza una red de flujo para buscar intercambios de clientes entre las rutas. Periódicamente se aplica el algoritmo 2-opt o el 3-opt sobre las rutas. Se mantiene un conjunto con las mejores soluciones encontradas durante la búsqueda y al cabo de cierta cantidad de iteraciones, se comienza una nueva búsqueda a partir de alguna de esas soluciones.

Rego y Roucariol [109] propusieron un algoritmo basado en TS en el cual la vecindad se define mediante cadenas de expulsiones (o *ejection chains*). Éstas fueron propuestas por Glover [67] como una técnica general para definir vecindades en un espacio de soluciones. En el caso de los problemas de ruteo de vehículos se trata de cambiar las posiciones de algunos nodos en las rutas. Una cadena de expulsiones de ℓ niveles consiste en ternas de nodos consecutivos en una ruta $\left(v_{i-1}^k, v_i^k, v_{i+1}^k\right)$ para $k = 0, \ldots, \ell$, en las que cada v_i^{k-1} es *expulsado* y reemplazará a v_i^k luego de la movida. El último nodo en ser expulsado es v_i^ℓ, debe ubicarse de modo tal que se obtenga rutas cerradas y se proponen dos alternativas que se ilustran en las Figuras 2.18 y Figuras 2.19.

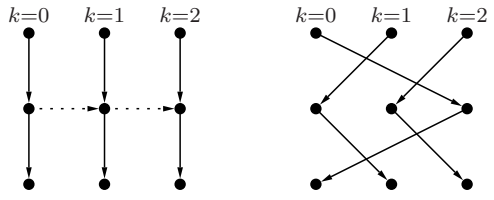


Figura 2.18: Una Cadena de Expulsiones de Tipo I para 3 niveles.

La idea principal del algoritmo *granular tabu search* propuesto por Toth y Vigo [127]

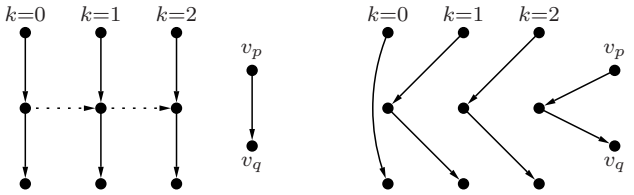


Figura 2.19: Una Cadena de Expulsiones de Tipo II para 3 niveles.

consiste en disminuir la cantidad de arcos considerados al evaluar las movidas, eliminando aquellos que conectan nodos muy lejanos, pues es de esperar que dichos arcos no formen parte de las buenas soluciones. Se define un umbral ν y solo se permite efectuar las movidas que involucran arcos en el conjunto $E' = \{(i,j) \in E \,|\, c_{ij} < \nu\} \cup I$, donde I es un conjunto de arcos "importantes", como por ejemplo, los incidentes al depósito. Esto permitió reducir la cantidad de arcos considerados a un valor entre el 10 % y el 20 % del total. Si bien esta idea puede ser utilizada en cualquier contexto, en la propuesta original se aplica una variante de Taburoute como algoritmo de solución.

Algoritmos genéticos

Existen diversas aplicaciones de algoritmos genéticos para resolver el TSP, que pueden consultarse en el trabajo de Potvin [106]. A continuación se describen algunos algoritmos genéticos para el VRP y sus variantes.

El algoritmo propuesto por Baker y Ayechew [8] para el VRP se basa en la técnica de asignar primero y rutear después (ver Sección 2.5.1). Cada individuo codifica la asignación de clientes a vehículos. La función de fitness se calcula generando una ruta para cada vehículo que pase por todos los clientes asignados a él utilizando 2-opt y luego 3-opt para obtener la secuencia de nodos. Se permite violar las restricciones del problema y dichas violaciones son penalizadas en la función de fitness. El Algoritmo GIDEON de Thangiah [124] también se basa en esta técnica. Los clusters se generan mediante la ubicación de "puntos semilla" en el plano. Desde el depósito se trazan semirrectas hacia cada punto semilla, definiendo sectores que particionan al conjunto de clientes en clusters. El algoritmo genético se utiliza para determinar la mejor ubicación de los puntos semilla.

Blanton y Wainwright [12] propusieron un AG en el cual cada individuo es una permutación del conjunto de los clientes. Para obtener el conjunto de rutas que la permutación representa, se utilizan los primeros clientes para inicializar las rutas y el resto se inserta secuencialmente en aquella ruta que genere menos incremento en el costo (siempre que la inserción sea factible). Si luego de ese proceso quedan clientes sin visitar, esto se penaliza en la función de fitness del individuo.

Potvin y Bengio propusieron un algoritmo genético llamado GENEtic ROUting System

(GENEROUS) [107] en el cual los operadores evolutivos se aplican directamente sobre las soluciones factibles y no sobre una codificación de estas. Se proponen dos operadores de cruzamiento. En el *sequence-based crossover*, se selecciona una ruta de cada padre, se elimina un arco de cada una y se construye una nueva combinando las sub-rutas generadas por el corte. El operador *route-based crossover* consiste simplemente en copiar una ruta de un padre en el otro. En ambos casos, la solución obtenida puede contener clientes duplicados y clientes no visitados. Los autores dan procedimientos para corregir estas deficiencias.

En esta la propuesta de Berger, Barkaoui y Bräysy [11] se utilizan dos poblaciones que evolucionan en paralelo y operan directamente sobre las soluciones. Los individuos de una misma población tienen la misma cantidad de rutas. Siendo $r^{\min}$ la mínima cantidad de rutas para la que se ha conseguido una solución factible, una de las poblaciones opera con individuos de $r^{\min}$ rutas (buscando minimizar el costo) y la otra con $r^{\min} - 1$ rutas (buscando una solución factible con esa cantidad de rutas). Ambas poblaciones evolucionan de la misma manera, utilizando los mismos operadores y la misma función de fitness.

En el algoritmo propuesto por Zhu [138], cada solución se representa como una permutación de los clientes. Para obtener las rutas, se agrega clientes a una ruta siguiendo el orden dado por la permutación y cuando incluir el siguiente cliente en la ruta viole las restricciones del problema, se crea una nueva.

En la propuesta de Bräysy y Dullaert [20] el algoritmo genético no utiliza una población de soluciones sino una solución. Partiendo de esa solución, se consideran todos los pares de rutas y se les aplica un operador de cruzamiento para generar nuevas rutas. Finalmente, a cada ruta generada se le aplica un operador de mutación. Las rutas obtenidas se almacenan en un conjunto y luego de analizar todos los pares se construye una solución seleccionando rutas de dicho conjunto.

Algoritmos de hormigas

El primer problema al que se aplicó esta estrategia fue el TSP [30]. Al comienzo del algoritmo se coloca una hormiga en cada nodo. Cada hormiga construye una solución seleccionando el próximo nodo a visitar de acuerdo a una regla probabilística que combina los datos del problema con información histórica de la ejecución del algoritmo. Si está ubicada en el nodo i, la hormiga se mueve al nodo j con probabilidad

$$p_{ij}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{h\in\Omega}[\tau_{ih}(t)]^{\alpha}[\eta_{ih}]^{\beta}} & \text{si } j \in \Omega \\ 0 & \text{si } j \notin \Omega \end{cases} \tag{2.28}$$

siendo Ω el conjunto de nodos aún no visitados por dicha hormiga. $\tau_{ij}(t)$ representa la cantidad de feromona depositada en el arco (i, j) y $\eta_{ij} = \frac{1}{c_{ij}}$ es una medida de qué tan

bueno es, *a priori*, moverse por dicho arco. Los parámetros α y β regulan la importancia relativa de la feromona y la visibilidad, es decir, de la información recolectada por la colonia y el criterio de selección ávido.

Cuando todas las hormigas han construido su solución, se actualiza la feromona en cada arco (i, j) según la siguiente ecuación

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \sum_{k=1}^{M} v(i,j,k)\frac{Q}{L_k} \tag{2.29}$$

donde $v(i, j, k)$ vale 1 si la hormiga k utilizó el arco (i, j) y 0 en otro caso, L_k es el costo de la solución encontrada por dicha hormiga y Q es un parámetro del algoritmo. El parámetro $\rho \in [0, 1]$, llamado *coeficiente de evaporación*, establece el porcentaje de feromona que permanece de una iteración a otra y se utiliza con el objetivo principal de evitar la convergencia prematura del algoritmo.

El algoritmo repite el ciclo de construcción de soluciones y actualización de las feromonas hasta que se cumpla un criterio de terminación.

Entre las diversas variantes existentes para este esquema, Dorigo et al. [44] proponen la introducción de una estrategia elitista que busca dar más énfasis a la mejor solución encontrada en cada iteración para la actualización de la feromona. Otra propuesta, llamada *selección por ranking* [23] consiste en permitir que sólamente las hormigas que obtuvieron mejores resultados en una iteración, actualicen las feromonas. El algoritmo ANT-Q [54], el *Ant Colony System* (ACS) [55] y los *MAX-MIN Ant Systems* (MMAS) [115] son algunas de las propuestas para resolver el TSP.

Este tipo de algoritmos también ha sido utilizado, aunque en menor medida, para resolver otro tipo de problemas de ruteo de vehículos. Entre los trabajos más destacados están el de Bullnheimer et al. [22, 21] para el VRP y el Multiple Ant Colony System (MACS) para el VRPTW, propuesto por Gambardella et al. [56].

Capítulo 3

Memorias adaptativas

3.1. Introducción

Muchos de los problemas relativos a la toma de decisiones, particularmente en las áreas de transporte y logística, requieren la resolución de problemas de optimización combinatoria que pertenecen a la clase NP-Hard. Debido a la complejidad intrínseca de estos problemas, los métodos de solución exactos suelen consumir tiempos de ejecución muy elevados para obtener soluciones de calidad aceptable y no proporcionan una alternativa viable para resolver instancias de tamaño realista. Es de interés, entonces, el desarrollo de métodos aproximados (o heurísticas) que permitan obtener soluciones de buena calidad en tiempos de ejecución moderados. En los últimos años se han desarrollado una serie de técnicas que permiten acercarse a ese objetivo.

Las metaheurísticas pueden definirse como descripciones de alto nivel de algoritmos aproximados. Son estrategias de propósito general, que deben ser adaptadas a cada problema en particular. Estas técnicas suelen ser métodos iterativos, que exploran el espacio de soluciones del problema visitando una gran cantidad de soluciones. Entre la amplia gama de metaheurísticas existentes, se encuentran las *metaheurísticas con memoria*. En este tipo de métodos se lleva un registro de las soluciones visitadas durante la ejecución, llamado *memoria*. La información almacenada en la memoria se utiliza como insumo para tomar las decisiones que guiarán la búsqueda en iteraciones posteriores.

En este capítulo se presentan dos metaheurísticas con memoria: *Tabú Search* y *Adaptive Memory Procedure*. Estos métodos, utilizados de manera combinada, permiten resolver una gran cantidad de problemas de manera eficiente y robusta. Se da una descripción de las ideas que originan cada uno de los métodos, sus principales ventajas y las dificultades que trae consigo. Asimismo, se muestra una aplicación de cada uno para resolver el VRP. Ambas aplicaciones guardan una estrecha vinculación con los métodos propuestos más adelante en este trabajo (ver Capítulo 5).

3.2. Metaheurísticas

En esta sección se considera un problema de optimización de la forma

$$\begin{aligned} \min \quad & f(x) \\ \text{s.a.} \quad & x \in X \end{aligned}$$

donde X es un conjunto discreto llamado *espacio de soluciones* o *espacio de búsqueda*. Una solución $s \in X$ es un *óptimo global* si $f(s) \leq f(s') \, \forall \, s' \in X$. El objetivo del problema es encontrar un óptimo global. Particularmente, estamos interesados en los casos en los cuales cumplir con ese objetivo es un problema NP-Hard.

En los últimos 20 años, ha surgido un conjunto de algoritmos aproximados que, combinando heurísticas simples con algunas ideas intuitivas, buscan explorar de manera inteligente el espacio de soluciones del problema. Este tipo de algoritmos suelen caracterizarse mediante descripciones de alto nivel, llamadas *metaheurísticas*. No existe una única definición para este término, pero resulta adecuado, para el propósito de este trabajo, entender a una metaheurística como una estrategia para explorar espacios de soluciones de manera efectiva [13]. Proporcionan grandes lineamientos que deben ser adaptadas a las particularidades de cada problema.

Entre las metaheurísticas más divulgadas se encuentran Simulated Annealing [84], Tabu Search [66, 69], GRASP [47, 105], Variable Neighborhood Search [98, 78], Guided Local Search [131], Iterated Local Search [94], Algoritmos Genéticos [79, 71], Scatter Search [68, 70] y Ant Colonies [30, 44]. En el Apéndice A se realiza un relevamiento de las principales metaheurísticas que han propuestas en la literatura. Los trabajos de Voß [130] y Blum y Roli [13], proveen un panorama general actualizado de las diferentes técnicas. Sobre heurísticas de búsqueda local, puede consultarse el libro de Aarts y Lenstra [2].

3.2.1. Algunos criterios de clasificación

Diversos criterios han sido propuestos para clasificar a las metaheurísticas. Los más importantes a los efectos de este trabajo, son [13]:

De trayectoria o basadas en poblaciones. Los métodos de trayectoria, como Tabu Search y Simulated Annealing, entre otros, mantienen una única solución que es modificada en las sucesivas iteraciones. Los métodos basados en poblaciones, en cambio, operan sobre un conjunto de soluciones de manera simultánea. En cada iteración, la población (o parte de ella) es reemplazada por una nueva. Los Algoritmos Genéticos y Scatter Search son métodos basados en poblaciones.

Con o sin memoria. Las metaheurísticas con memoria utilizan información de la ejecución pasada del algoritmo para tomar decisiones en las iteraciones futuras. Ejemplos de este tipo de técnicas son Tabu Search y Ant Colonies. En los métodos sin memoria, como GRASP, los resultados obtenidos en iteraciones previas no condicionan explícitamente las decisiones tomadas por el algoritmo.

De función objetivo estática o dinámica. En algunos métodos, la función objetivo utilizada para guiar el algoritmo es la misma función objetivo del problema y permanece invariante durante todo el algoritmo. En otros casos, como Guided Local Search, se utiliza una modificación de dicha función y se la hace varíar conforme pasan las iteraciones.

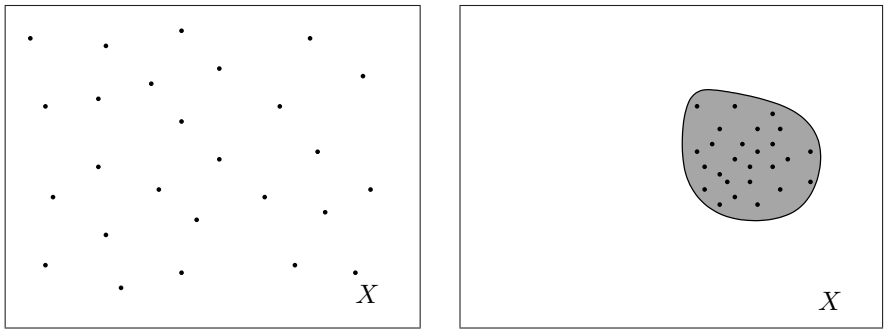
Estos criterios pueden aplicarse a las metaheurísticas en su estado más abstracto. Sin embargo, las implementaciones reales de estos algoritmos suelen tomar elementos de diversas fuentes, dando lugar a métodos híbridos difíciles de clasificar según criterios rígidos. De todos modos, no se pretende con esto llegar a una taxonomía de metaheurísticas, sino dar una idea clara de las diferentes estrategias.

3.2.2. Metaheurísticas con memoria

En las metaheurísticas con memoria se almacena información sobre las soluciones que han sido visitadas durante la ejecución (en una estructura llamada *memoria*), y se utiliza esa información para tomar decisiones en las sucesivas iteraciones. En ese sentido, este tipo de metaheurísticas busca realizar un mejor aprovechamiento de la información contenida en las soluciones ya visitadas.

En general, cuando una metaheurística sin memoria visita una solución que no cumple con cierto criterio de aceptación (por ejemplo, no mejora a la mejor solución encontrada), la descarta. El punto de partida para las metaheurísticas con memoria es que dicha solución podría contener información relevante para la búsqueda, más allá de su valor objetivo. Más aún, el análisis de un conjunto de soluciones puede permitir obtener nuevas soluciones de mejor calidad mediante la detección de patrones repetidos, como se verá en la Sección 3.5.

La información almacenada en la memoria puede utilizarse, por ejemplo, para evitar la exploración de ciertas regiones del espacio de soluciones. Este tipo de estrategias, llamadas de *inhibición*, son utilizadas en el caso de Tabu Search. En este caso, se almacena información de las últimas soluciones visitadas y se evita seleccionarlas al explorar la vecindad. Otro posible uso de la información en memoria, contrapuesto con el anterior, consiste en sesgar la búsqueda hacia ciertas regiones del espacio de búsqueda. Esta *retroalimentación* se utiliza, por ejemplo, en los Ant Systems. Cada hormiga es influenciada positivamente por las cantidades de feromona depositados en el pasado, para no apartarse demasiado de las soluciones de cierta calidad encontradas en iteraciones anteriores.



(a) Diversificación (b) Intensificación

Figura 3.1: Soluciones visitadas durante una búsqueda diversa y una búsqueda intensa.

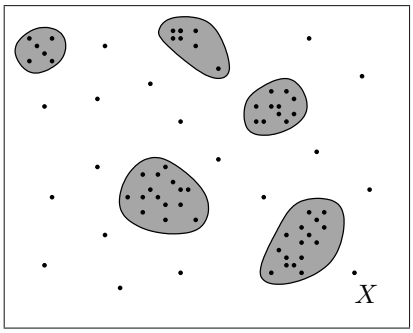


Figura 3.2: Compromiso entre diversificación e intensificación.

3.2.3. Intensificación y diversificación

Una característica común a todas las metaheurísticas es que durante su ejecución examinan un conjunto de soluciones. Las diferentes propuestas buscan que dentro del conjunto de soluciones examinadas (o visitadas), se encuentre alguna solución óptima o cercana a la optimalidad.

La búsqueda realizada por una metaheurística debería, por un lado, concentrarse en examinar minuciosamente las regiones del espacio en las que aparezcan soluciones de buena calidad, y, por otro lado, visitar áreas no exploradas en busca de mejores soluciones. Estos conceptos suelen denominarse, respectivamente, *intensificación* y *diversificación*. La noción de "región" del espacio de búsqueda depende la definición de alguna métrica en el conjunto X. Es empleada aquí de manera informal para ilustrar estos conceptos. En la Figura 3.1 se muestra de manera esquemática estas ideas.

Intensificación y diversificación son efectos causados por los diferentes componentes de un algoritmo y su interacción. Si bien aparecen como conceptos contrapuestos, debe pensárselos como complementarios. El éxito de una metaheurística depende, en buena medida, de un adecuado balance entre intensificación (para obtener soluciones de buena calidad) y diversificación (para decidir en qué regiones hacer una búsqueda más intensa). La Figura 3.2 muestra esquemáticamente este compromiso.

3.2.4. El uso de metaheurísticas

Los problemas candidatos a ser atacados mediante metaheurísticas son aquellos para los cuales el conocimiento que se dispone acerca de su espacio de soluciones y su función objetivo, no permite diseñar otro tipo de algoritmos que resulten eficientes en la práctica. Para algunos problemas es posible derivar un principio de optimalidad, formulaciones matemáticas compactas o cotas inferiores ajustadas. En algunos casos, haciendo uso de esas propiedades particulares del problema, puede construirse algoritmos *a medida* que garanticen encontrar una solución óptima en un número finito de pasos, y que, además, consuman recursos computacionales moderados en la práctica. Sin embargo, en muchos casos no se cuenta con tales elementos o, aún contando con ellos, no son suficientes para lidiar con la complejidad intrínseca del problema o con el tipo de instancias que se pretende resolver. Las metaheurísticas surgen como una alternativa válida para la resolución de problemas en ese contexto.

Desafortunadamente, existen escasos resultados acerca de la convergencia de las metaheurísticas y los que existen resultan poco útiles. Por ejemplo, bajo ciertas hipótesis puede probarse que la probabilidad de que un algoritmo basado en Simulated Annealing encuentre un óptimo global tiende a 1 cuando la cantidad de iteraciones tiende a infinito [1]. Este resultado no plantea ninguna ventaja de este método frente a los métodos exactos, que en general encuentran un óptimo global luego de una cantidad exponencial, pero finita, de iteraciones. Además, las implementaciones reales suelen incorporar ideas heterogéneas, dando lugar a algoritmos híbridos con diversos componentes cuya interacción resulta muy difícil de modelar analíticamente.

Sin embargo, desde un punto de vista experimental, puede decirse que este tipo de algoritmos suelen obtener buenos resultados además de un compromiso adecuado entre la calidad de las soluciones obtenidas y el tiempo de ejecución necesario para hallarlas. La dificultad para dar evidencia analítica de los buenos resultados experimentales que se obtienen en la práctica constituye una de las principales desventaja que trae consigo el uso de estos métodos. Entre los esfuerzos que se han realizado por avanzar en esta dirección, cabe mencionar el uso de Cadenas de Markov para analizar algoritmos basados en Simulated Annealing [1] y el teorema de esquemas para los Algoritmos Genéticos [71]. En el trabajo de Hoos y Stützle se discuten algunas herramientas que permiten analizar la estructura del espacio de soluciones en los algoritmos de búsqueda local [82].

3.3. Algoritmos de búsqueda local

Un algoritmo de búsqueda local se basa en alguna noción de proximidad en el espacio de soluciones. Para cada solución s, debe definirse un *conjunto de soluciones vecinas* $N(s)$, también llamado *vecindad*. Usualmente la vecindad se define mediante modificaciones sim-

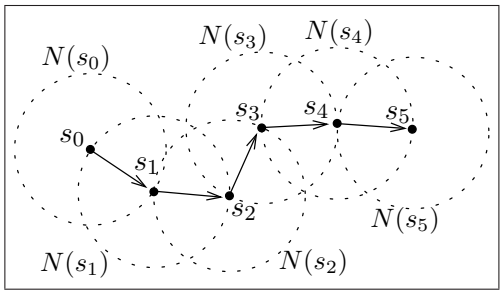


Figura 3.3: Trayectoria generada por un algoritmo de búsqueda local.

Paso 1 (inicialización).
Construir una solución inicial s_0. Hacer $k \leftarrow 0$.

Paso 2 (Búsqueda local).
Hacer $N \leftarrow \{s \in N(s_k) : f(s) < f(s_k)\}$.
Si $N = \emptyset$, devolver s_k y terminar.
Si no, elegir $s_{k+1} \in N$, hacer $k \leftarrow k + 1$ y repetir el paso 2.

Figura 3.4: Algoritmo de descenso simple.

ples sobre la solución, llamadas *movidas*, que no alteran sustancialmente su estructura.

Un *óptimo local* es una solución s para la cual $f(s) \leq f(s') \, \forall \, s' \in N(s)$. Es decir, una solución para la cual ninguna de sus vecinas tiene un mejor valor objetivo. Cada definición de vecindad genera diferentes óptimos locales. Toda definición de vecindad trae consigo la presencia de óptimlos locales.

En un algoritmo de búsqueda local se parte de una solución inicial s_0 y en la iteración k se selecciona una nueva solución $s_{k+1} \in N(s_k)$. La secuencia de soluciones visitadas $(s_0, s_1, \ldots)$ determina una trayectoria en el espacio de soluciones, como se ilustra en la Figura 3.3. La definición de una estructura de vecindad junto con un criterio para seleccionar la próxima solución dentro de ella, caracterizan a un algoritmo de este tipo.

En un algoritmo de descenso simple (ver Figura 3.4) el criterio es moverse hacia alguna solución de menor costo. Cuando el algoritmo alcanza un óptimo local, ninguna de las soluciones vecinas tiene menor costo y, por lo tanto, se finaliza la ejecución. Esquemáticamente, este comportamiento se muestra en la Figura 3.5, en la cual se finaliza la ejecución luego de haber visitado la secuencia $(s_0, s_1, \ldots, s_N)$, y se devuelve s_N.

Existe una relación entre la solución inicial y las soluciones que pueden ser alcanzadas siguiendo el algoritmo de descenso simple. Dicha relación se ilustra en la Figura 3.6. Para acceder al óptimo global s^{opt}, es necesario que $s_0 \in A(s^{\text{opt}})$, de lo contrario el algoritmo convergerá a un óptimo local (s_1^* o s_2^*).

Detenerse en el primer óptimo local encontrado puede no ser suficiente si se pretende alcanzar soluciones con cierto nivel de calidad. En esos casos, debe encontrarse alguna manera de proseguir la búsqueda. La manera más simple para continuar con la exploración

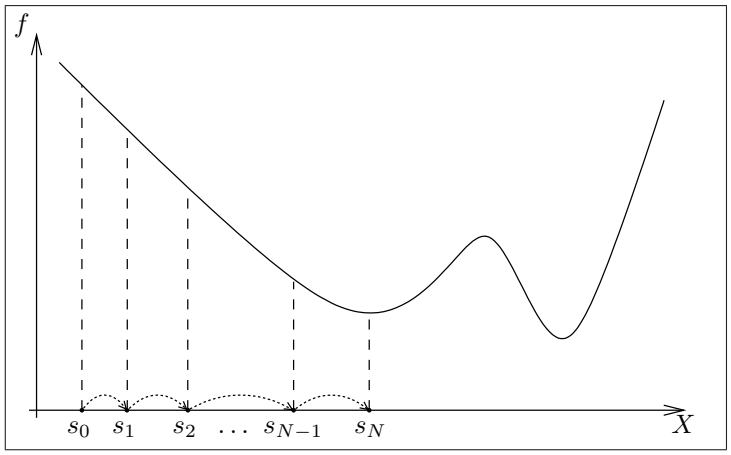


Figura 3.5: Convergencia de un algoritmo de búsqueda de descenso.

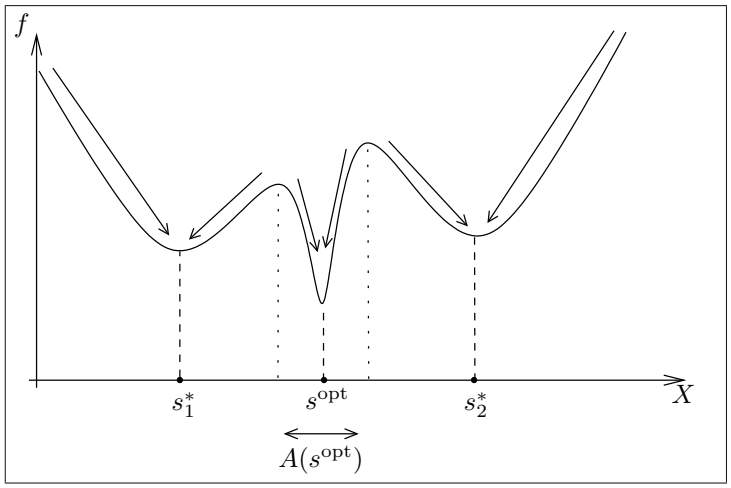


Figura 3.6: Relación entre la solución inicial y la final para un algoritmo de descenso.

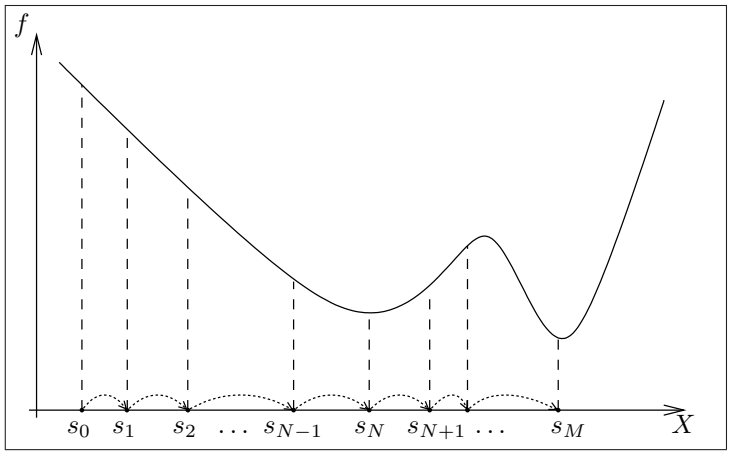


Figura 3.7: Aceptar un deterioro de la solución puede permitir escapar de óptimos locales.

del espacio de soluciones consiste en ejecutar el algoritmo nuevamente, comenzando desde una solución que no haya sido visitada anteriormente. Ejemplos de estrategias de este tipo, denominadas multi-comienzo o multi-arranque, son GRASP e Iterated Local Search.

3.4. Tabu Search – memoria de corto plazo

La metaheurística *Tabu Search* (TS) [66] o búsqueda tabú opera como un algoritmo de búsqueda local y provee un mecanismo para escapar de los óptimos locales y poder continuar la exploración. A diferencia de los métodos multi-comienzo, se busca no apartarse demasiado de la región en la que se encontró el óptimo local.

El efecto que se pretende obtener se ilustra en la Figura 3.7. Para que ello ocurra es necesario permitir que en la búsqueda se visiten soluciones que empeoran la función objetivo. Así se logra pasar de s_N a s_{N+1} en el ejemplo. Entonces, en lugar de moverse siempre hacia una solución que decremente el valor objetivo, se busca la mejor solución de la vecindad (que posiblemente sea peor que la solución actual, en términos de la función objetivo).

Este criterio tan simple no es suficiente para escapar de óptimos locales. Si en el ejemplo anterior se cumpliera que $s_N \in N(s_{N+1})$, la mejor decisión podría ser moverse de s_{N+1} nuevamente a s_N. Esto generaría que la búsqueda entre en un ciclo infinito que visita alternadamente s_N y s_{N+1}. De manera similar, podría entrarse en ciclos de más de dos soluciones. Para evitar que se den estos casos, debe lograrse que el algoritmo no considere las soluciones que han sido visitadas, al menos en un pasado cercano. La estrategia consiste en considerar, en la iteración k, una vecindad reducida $N_k^*(s_k) \subseteq N(s_k)$ que excluya esas soluciones y realizar las movidas en base a esa nueva vecindad.

Una manera directa de lograr que el algoritmo eluda las soluciones más recientes es almacenar un conjunto, llamado *lista tabú*, con las últimas δ soluciones visitadas (donde

Paso 1 (inicialización).
Construir una solución inicial s_0 y hacer $s^{\text{best}} \leftarrow s_0$.
Hacer $L_0 = \emptyset$ y $k \leftarrow 0$.

Paso 2 (búsqueda local).
Definir $N_k^*(s_k) = N(s_k) \setminus L_k$.
Hacer $s_{k+1} \leftarrow \arg\min_{s \in N_k^*(s_k)} f(s)$.

Paso 3 (actualización).
Si $f(s_{k+1}) < f(s^{\text{best}})$, hacer $s^{\text{best}} \leftarrow s_{k+1}$.
Hacer $L_{k+1} \leftarrow L_k \cup \{s_k\}$. Si $k \geq \delta$, hacer $L_{k+1} \leftarrow L_{k+1} \setminus \{s_{k-\delta}\}$.

Paso 4 (terminación).
Hacer $k \leftarrow k + 1$.
Si $k = TS^{\text{iter}}$ devolver s^{best} y terminar; si no, ir al paso 2.

Figura 3.8: Tabú Search con lista de soluciones tabú.

δ es un parámetro del algoritmo). En la iteración k, la lista tabú es

$$L_k = \{s_i : \max(0, k - \delta) \leq i \leq k - 1\} \tag{3.1}$$

y la vecindad reducida se define como $N_k^*(s_k) = N(s_k) \setminus L_k$. Las soluciones de la lista tabú se denominan *soluciones tabú*. Con esta vecindad se prohíbe volver a pasar por cualquiera de las últimas δ soluciones, lo que evita que el algoritmo entre en ciclos de hasta $\delta + 1$ soluciones repetidas. El valor del parámetro δ puede ser fijo o variar dinámicamente durante la ejecución. Usualmente se toman valores bajos, que sean suficientes para lograr escapar del óptimo local, lo cual depende de cada problema en particular.

El algoritmo con el enfoque de lista de soluciones tabú, se da en la Figura 3.8. Esta estrategia requiere almacenar una lista de soluciones y, además, comparar soluciones por igualdad cada vez que se considera una solución vecina. Estas dos operaciones pueden requerir un tiempo de ejecución excesivo.

Un enfoque alternativo consiste en almacenar en la lista tabú, las inversas de las últimas movidas realizadas. Al explorar la vecindad se evita realizar las movidas que están almacenadas en la lista tabú. En general, esta alternativa es más sencilla de implementar, puesto que las movidas, a diferencia de las soluciones, suelen ser caracterizadas utilizando poca información. Por ejemplo, si una movida para algún problema sobre grafos consiste en intercambiar dos nodos de posición, ésta puede codificarse mediante un par ordenado, simplificando las operaciones de almacenamiento y comparación de movidas.

Puede permitirse, en algunos casos, que el algoritmo se mueva a una solución aún cuando esta sea una solución tabú. Por ejemplo, una solución puede ser aceptada siempre que mejore la mejor solución encontrada hasta el momento. Este criterio es denominado *criterio de aspiración.*

La lista tabú utilizada para inhibir la exploración de ciertas regiones del espacio de búsqueda, ya sea que almacene soluciones o movidas, actúa como una *memoria de corto plazo*, pues los elementos abandonan la lista al cabo de δ iteraciones del algoritmo. Aún utilizando este tipo de memorias de corto plazo es posible, dado, precisamente, el carácter volátil de la memoria, que el algoritmo quede atrapado en cierta región del espacio. El uso de una *memoria de largo plazo* (que se analiza en la Sección 3.5) surge para remediar ese defecto y lograr una mayor diversificación de la búsqueda.

3.4.1. El algoritmo Taburoute

El algoritmo *Taburoute* es una aplicación de la metaheurística Tabu Search y fue propuesto por Gendreau, Hertz y Laporte [60] para resolver una versión del VRP en la cual el costo de cada ruta no puede exceder cierto valor prefijado L.

Dada una solución s, la estructura de vecindad utilizada se define de la siguiente manera. Se seleccionan q clientes diferentes de manera aleatoria, con probabilidad uniforme. Para cada cliente, se consideran todas las soluciones que pueden obtenerse eliminándolo de la ruta a la que pertenece e insertándolo en otra ruta que contenga a alguno de sus p clientes más cercanos. Los valores de p y q se calculan al comienzo de la ejecución en función de las características de la instancia que se esté resolviendo. Para insertar un cliente en una ruta se utilizan las inserciones generalizadas GENI [59], presentadas en la Sección 2.5.1. Todas las soluciones generadas de esta manera forman la vecindad $N(s)$. Dado que $N(s)$ depende de la elección aleatoria de q clientes, diferentes ejecuciones del algoritmo podrían dar diferentes resultados.

Cuando el cliente i es eliminado de la ruta r, cualquier movida que vuelva a insertar i en r es declarada tabú por δ iteraciones, donde δ se sortea uniformemente en el intervalo $[5, 10]$. Este es otro factor que contribuye al no determinismo del algoritmo. Se utiliza el criterio de aspiración usual, aceptando soluciones que mejoran a la mejor solución encontrada, aunque sean soluciones tabú.

Un elemento adicional, que aporta flexibilidad al algoritmo, es que se considera la posibilidad de visitar soluciones que no son factibles. En particular, se permite violar las restricciones de capacidad de los vehículos y costo máximo de las rutas. Dada una solución s, se utiliza la siguiente función objetivo para evaluar la calidad de las soluciones y moverse en la vecindad:

$$f(s) = c(s) + \alpha Q(s) + \beta L(s)$$

donde $c(s)$ es el costo de la solución, $Q(s)$ y $L(s)$ miden la infactibilidad de la solución respecto a dichas restricciones y α y β son valores que se calculan dinámicamente durante la ejecución del algoritmo.

Al comienzo de la ejecución del algoritmo $\alpha = \beta = 1$. En cada iteración se chequea

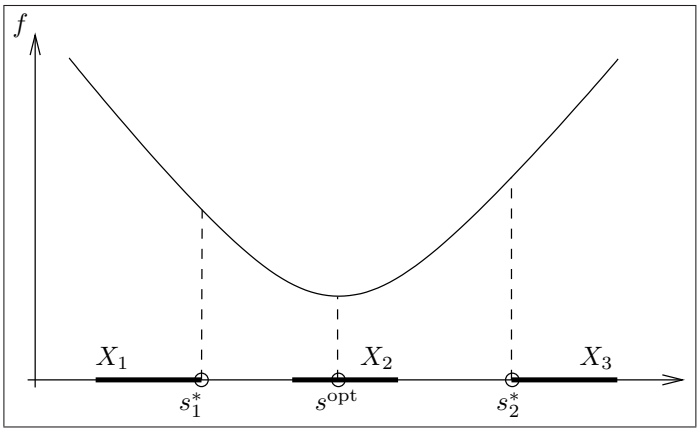


Figura 3.9: Óptimos locales causados por las restricciones del problema.

si las últimas 10 soluciones fueron factibles respecto a la restricción de capacidad. En caso afirmativo, se hace $\alpha \leftarrow \alpha/2$. Si todas fueron no factibles, se hace $\alpha \leftarrow 2\alpha$. El valor de β se actualiza del mismo modo, pero respecto a la restricción de costo máximo de las rutas. Una misma movida puede resultar muy atractiva en cierto momento y, sin embargo, al pasar las ejecuciones volverse de poco interés. Por ejemplo, ante una movida que decrementa el costo pero introduce infactibilidades y otra que incrementa el costo pero elimina algún tipo de infactibilidad, la decisión a tomar es compleja. Con esta propuesta, cuando se está explorando una región de soluciones factibles, se pone énfasis en el costo de las soluciones. Del mismo modo, cuando no se logra hallar soluciones factibles, se prioriza la factibilidad sobre el costo.

En muchos problemas de optimización, es la presencia de restricciones lo que provoca óptimos locales. Por ejemplo, en la Figura 3.9 se muestra un problema cuya región factible está formada por los conjuntos disjuntos X_1, X_2 y X_3. La función objetivo tiene un sólo mínimo global s^{opt}, pero al considerar las restricciones se agregan los mínimos locales s_1^* y s_2^*. Ningún algoritmo de búsqueda local que solamente considere soluciones factibles, podrá alcanzar la solución óptima global, si $s_0 \notin X_2$. Al aceptar soluciones no factibles se contribuye a eliminar este tipo de dificultades. Sin embargo, debe procurarse un adecuado balance entre soluciones factibles y no factibles, ya que no tiene sentido que el algoritmo sólamente visite soluciones que no respetan las restricciones del problema.

3.5. Adaptive Memory Procedure – memoria de largo plazo

El *Adaptive Memory Procedure* (AMP) es una metaheurística que puede utilizarse en combinación con cualquier algoritmo de búsqueda local. Fue propuesta por Rochat y Taillard [112] en base a las ideas manejadas por Glover [65] sobre *surrogate constraints*. Inicialmente este procedimiento fue utilizado como un mecanismo de diversificación para TS, lo cual provoca algunos autores utilicen los términos AMP y TS indiscriminadamente.

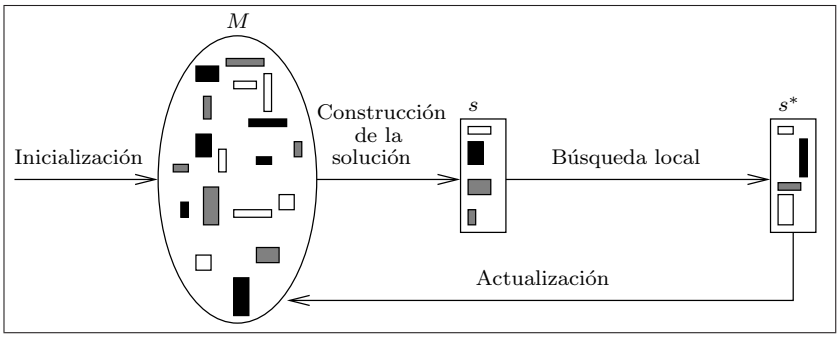


Figura 3.10: Esquema del funcionamiento del AMP.

En este trabajo, se optó por separar los conceptos, en el entendido de que el AMP puede ser utilizado en combinación con cualquier algoritmo de búsqueda local.

El punto de partida para este método es una visión estructurada de las soluciones del problema que se pretende resolver: una solución debe ser considerada como una agregación de ciertos *componentes*. De modo general, la propuesta consiste en identificar componentes que sistemáticamente aparecen en buenas soluciones. La motivación está dada por dos expectativas optimistas:

- es de esperar que componentes similares (o eventualmente idénticos) aparezcan en soluciones de mejor calidad,
- si se construyen soluciones combinando algunos de esos componentes, es posible que aparezcan nuevos componentes que anteriormente no se percibían como buenos.

La idea consiste en almacenar componentes de algunas de las mejores soluciones encontradas por el algoritmo, en una estructura llamada *memoria de largo plazo*. El esquema de la Figura 3.10 ilustra el funcionamiento del método. Periódicamente, se construye una solución en base a la información almacenada en la memoria y se realiza una búsqueda local comenzando en dicha solución. La solución obtenida por el algoritmo de búsqueda local se utiliza para actualizar la información almacenada en la memoria. Un seudocódigo del algoritmo se muestra en la Figura 3.11.

Al inicializar el algoritmo, debe dotarse a la memoria de componentes para iniciar el proceso. Para ello, puede construirse algunas soluciones mediante un método simple e insertar los componentes en la memoria. En la fase de construcción se busca combinar algunos componentes almacenados en la memoria para generar una solución. Típicamente, el proceso consistirá en seleccionar algunos componentes. En la búsqueda local se procura mejorar dicha solución. Finalmente, en la fase de actualización se agrega, eventualmente, los componentes de la solución obtenida a la memoria. Este proceso se repite durante una cantidad prefijada de iteraciones AMP^{iter}.

El comportamiento esperado del algoritmo es el siguiente. Al comienzo la memoria debería contener componentes bien diferenciados y, por lo tanto, las soluciones construídas

Paso 1 (inicialización).
Inicializar la memoria M. Hacer $s^{\text{best}} \leftarrow \emptyset$, $z^{\text{best}} \leftarrow \infty$ y $k \leftarrow 0$.

Paso 2 (construcción de la solución).
Construir una solución s en base a la información en M.

Paso 3 (búsqueda local).
Aplicar la búsqueda local partiendo de s. Sea s^* la solución obtenida.

Paso 4 (actualización).
Actualizar M utilizando la solución s^*.
Si $f(s^*) < z^{\text{best}}$, hacer $s^{\text{best}} \leftarrow s^*$ y $z^{\text{best}} \leftarrow f(s^*)$.

Paso 5 (terminación).
Hacer $k \leftarrow k + 1$.
Si $k = AMP^{\text{iter}}$ devolver s^{best} y terminar; si no, ir al paso 2.

Figura 3.11: El Adaptive Memory Procedure (AMP).

en sucesivas ejecuciones del paso 2 deberían ser también muy diferentes entre sí. En consecuencia, las búsquedas locales se realizarán sobre regiones bien diferentes del espacio de soluciones. A medida que transcurran las iteraciones, la búsqueda local fallará en su intento por mejorar la solución inicial (o al menos, provocará cambios muy pequeños). Por lo tanto, la memoria se modificará cada vez con menos frecuencia, con lo cual las soluciones construídas en el paso 2 estarán ahora restringidas a unas pocas regiones del espacio de soluciones. Es decir, en las primeras iteraciones se espera un proceso de diversificación y, a medida que pasen las iteraciones, la búsqueda se intensificará en ciertas regiones promisorias del espacio de soluciones.

Cuánto más diversa[1] sea la memoria, más diferentes serán sus elementos entre sí. Este concepto juega un rol central en el desempeño del algoritmo. Son los cambios a la diversidad de la memoria los que provocan el comportamiento descripto en el párrafo anterior. Cuánto menor sea la diversidad de la memoria, más se intensificará la búsqueda, como se muestra en la Figura 3.12.

Para obtener un algoritmo basado en este esquema debe proveerse, además de una heurística de búsqueda local, un esquema de manejo de la memoria. Esto implica definir la estructura de la memoria junto con operaciones para inicializarla, agregar soluciones y construir soluciones en base a la información almacenada. Algunos aspectos que consideramos importantes en el diseño de un algoritmo basado en el AMP, son los siguientes:

Descomposición de las soluciones en componentes. En general, para un mismo problema existen varias maneras de definir la estructura de sus soluciones. Una buena

[1] Entendiendo por diversidad a alguna medida que cuantifique la diferencia entre los elementos de la memoria.

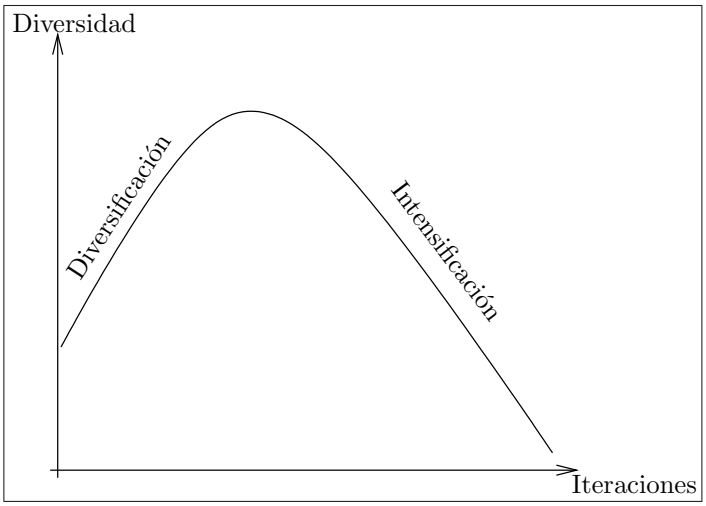


Figura 3.12: Intensificación y diversificación en el AMP.

descomposición debe lograr un balance adecuado de la cantidad de información que contiene cada componente aislado. Si en cada componente se almacena demasiada información, habrá pocas posibilidades de diversificar la búsqueda en la fase de construcción (pues la selección de un componente de la memoria condicionará demasiado la solución que se puede construir). Por otro lado, si los componentes contienen poca información, no habrá grandes diferencias con ejecutar la búsqueda local comenzando de soluciones aleatorias.

Elementos repetidos en la memoria. Dada una estrategia de descomposición, es usual que un mismo componente aparezca en diferentes soluciones. Por lo tanto, debe decidirse como se manejarán los componentes repetidos dentro de la memoria. Si se decide aceptarlos, es posible que en algunos casos ocurra una rápida pérdida de la diversidad. Por otro lado, si no se aceptan elementos repetidos es posible que no se logre capturar la importancia de dichos elementos (que es la motivación del algoritmo, como se mencionó al comienzo de esta sección).

Inicialización de la memoria. Al comienzo, la memoria debe ser lo suficientemente diversa como para que las soluciones construidas permitan cubrir diferentes regiones del espacio de búsqueda.

Construcción de la solución. Las soluciones deben construirse procurando aprovechar la información contenida en la memoria. Una buena opción es seleccionar ciertos componentes de manera no determinística y construir con ellos la solución. La selección debería estar sesgada hacia los componentes de las mejores soluciones y hacia aquellos que se repiten con frecuencia (en caso de que se acepten componentes repetidos).

No determinismo en la búsqueda local. Luego de algunas iteraciones del AMP, se espera que las soluciones construidas en el paso 2 sean cada vez más similares. Esto hace necesario que la búsqueda local incorpore cierto grado de no determinismo. De lo contrario, en el caso extremo de que se genere la misma solución varias veces, en lugar de intensificar la búsqueda cerca de dicha solución, simplemente se estaría repitiendo un mismo proceso una y otra vez.

Actualización de la memoria. El proceso de actualización de la memoria no solamente implica agregar los componentes de las nuevas soluciones, sino que también debe incluir alguna política de reemplazo. Por motivos de eficiencia, es aconsejable que la cantidad de elementos en la memoria esté acotada de antemano. Además, a medida que se van encontrando mejores soluciones, muchos de los componentes que se utilizaron anteriormente ya no resultarán tan atractivos como los nuevos y pueden ser eliminados de la memoria.

3.5.1. El algoritmo de Rochat y Taillard

La propuesta original de este método fue realizada por Rochat y Taillard para resolver el VRP y el VRPTW [112]. En este caso, los componentes de una solución son rutas. Cuando una solución es agregada a la memoria, se subdivide en las rutas individuales que la componen. Cada una de las rutas es agregada a la memoria etiquetada con el valor de la solución. Al construir una solución con los datos de la memorias se seleccionan rutas de manera probabilística, privilegiando aquellas con menor valor de la etiqueta.

Esta estrategia está fuertemente basada en la idea de que no es muy difícil construir una buena ruta, sino que lo complejo es hallar un conjunto de rutas que sea simultáneamente bueno para todos los clientes. En ese sentido, la combinación de rutas que estén en diferentes soluciones de buena calidad, puede originar mejores soluciones. Por ejemplo, si bien las soluciones de las Figuras 3.13(a) y 3.13(b) tienen características que las alejan de la optimalidad, algunas de sus rutas pueden utilizarse para construir una solución mejor. La solución ilustrada en la Figura 3.13(c) surge de combinar las rutas marcadas de cada una de las soluciones anteriores.

Las dos suposiciones planteadas al comienzo de la Sección 3.5 para justificar intuitivamente la metodología, se traducen en este contexto de la siguiente manera. Si una ruta aparece repetidas veces en las mejores soluciones que visita el algoritmo, es de esperar que participe en otras soluciones de buena calidad. Más aún, si se combinan algunas de estas rutas para construir una solución, podría esperarse que otras buenas rutas para el resto de los clientes surjan de la búsqueda en una manera similar. Es en ese sentido que se explota la información contenida en las soluciones visitadas por el algoritmo.

En este caso, la memoria M es un multi-conjunto de rutas, cada una de las cuales

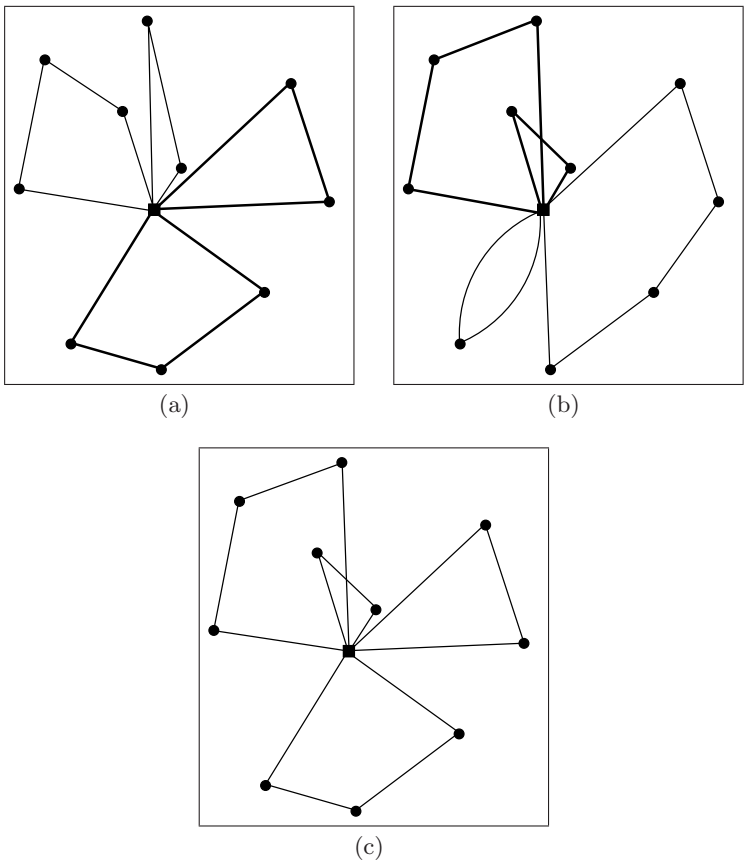


Figura 3.13: Algunas rutas de soluciones de cierta calidad pueden ser utilizadas para construir una mejor solución.

se etiqueta con el costo de la solución por la que fue agregada a la memoria. Un aspecto importante es que la memoria sea un multi-conjunto, es decir, que una misma ruta pueda aparecer repetidas veces. Si una ruta aparece en diferentes buenas soluciones, esta propuesta sugiere tenerla en cuenta en la fase de construcción. Por cuestiones de eficiencia, resulta útil mantener la memoria ordenada en forma creciente según las etiquetas. El tamaño máximo de la memoria está especificado de antemano por el parámetro M^{size}.

Inicialización

Al comienzo, se genera I soluciones diferentes mediante una heurística constructiva no determinística, simple y rápida. A cada una de ellas se le aplica la búsqueda local y la solución obtenida se agrega a la memoria.

Construcción de una solución

El procedimiento de construcción de una solución basado en datos de la memoria juega un papel central del AMP, pues es donde efectivamente se utiliza la información histórica de la búsqueda. En este caso, se selecciona rutas de M de manera probabilística, privilegiando a las que aparecen al comienzo (es decir, las que forman parte de las mejores soluciones halladas durante la ejecución del AMP).

Dado que las rutas seleccionadas deben visitar exactamente una vez a cada cliente, aquellas rutas que tienen algún cliente en común con las ya seleccionadas se descartan de la selección. La distribución de probabilidades utilizada al elegir una ruta no depende de los valores de las etiquetas sino de su orden relativo, y está sesgada hacia las que ocupan las primeras posiciones. Tres aspectos favorecen la selección de una ruta en la fase de construcción:

- que forme parte de una buena solución que ya fue visitada, pues ocupará alguna de las primeras posiciones de M, lo que provoca que su probabilidad de selección sea alta respecto a las demás,

- que pertenezca a varias soluciones, pues como se aceptan elementos repetidos en la memoria, basta con que una de las copias de r sea elegida,

- que las rutas que aparecen antes en la memoria compartan clientes con alguna de las ya seleccionadas (y, por lo tanto, no puedan ser elegidas).

Búsqueda local

La búsqueda local utilizada es un algoritmo basado en TS propuesto por Taillard [116].

Actualización

Cuando una solución es agregada a la memoria, cada ruta se etiqueta con el valor de dicha solución y se inserta en el conjunto M preservando el orden creciente de las etiquetas. Con este esquema, las rutas que pertenecen a las mejores soluciones encontradas aparecen al comienzo de M. Si luego de la inserción, $|M| > M^{\text{size}}$, las últimas $|M| - M^{\text{size}}$ rutas se eliminan de la memoria.

3.6. Relevamiento bibliográfico

La idea de construir soluciones mediante información almacenada en una memoria de largo plazo y mejorarlas con un algoritmo basado en TS, ha sido aplicada con éxito a la resolución de diversos problemas de optimización.

Luego de que Rochat y Taillard [112] propusiera en AMP, muchos autores modificaron esa propuesta para considerar otras variantes de problemas de ruteo de vehículos. Golden, Laporte y Taillard [75] diseñaron un algoritmo similar para resolver problemas con objetivo "min-max" (por ejemplo, minimizar el costo de la ruta más costosa). Un algoritmo para resolver el VRPMT, basado en las mismas ideas, fue propuesto por Taillard, Laporte y Gendreau [120]. El VRP con ventanas de tiempo flexibles [118], el VRPHF [117] y una versión del VRP con múltiples depósitos [36], también fueron resueltos utilizando variantes del AMP.

Recientemente, Tarantilis y Kiranoudis [123] y Tarantilis [122], propusieron una manera alternativa de organizar la memoria. Los componentes de las soluciones, en lugar de ser rutas, son secuencias de clientes (que no necesariamente tienen al depósito en los extremos). En lugar de extraer rutas completas de la memoria se extraen dichas secuencias, llamadas *bones* (o huesos). En este caso, cada componente de una solución contiene menos información que en la propuesta original del AMP.

Las aplicaciones del AMP trascienden a los problemas de ruteo de vehículos, destacándose su uso para la resolución de problemas de asignación [52], de clustering [5], de recolección de recompensas [121] e incluso para la planificación de distritos electorales [16].

El AMP puede considerarse una visión unificada, de alto nivel, de muchas metaheurísticas con memoria. Taillard et al. [119] ilustran como los Algoritmos Genéticos, Scatter Search, Tabu Search y Ant Systems, pueden ser considerados instancias del AMP con diferentes estrategias de manejo de memoria. Si bien esto puede parecer una propiedad de poca importancia, desde el punto de vista práctico resulta útil hacer un esfuerzo por unificar las decenas de metaheurísticas que han sido propuestas en los últimos años.

Capítulo 4

El problema de ruteo de vehículos con múltiples viajes

4.1. Introducción

Algunas características importantes que surgen en las aplicaciones reales no son tenidas en cuenta en los modelos clásicos de ruteo de vehículos. En particular, en el VRP y sus derivados, suele suponerse que cada vehículo recorre una sola ruta en un mismo período de planificación y que el tamaño de la flota es ilimitado.

En muchos contextos no es realista suponer una correspondencia uno a uno entre vehículos y rutas. Es común en las zonas urbanas, donde las distancias son relativamente cortas, que cada vehículo recorra más de una ruta en un mismo período de planificación (que usualmente es de un día). En muchos casos, porque esto permite una reducción en la cantidad de vehículos utilizados, y por lo tanto, en los costos de operación. En otros, porque quizás sea la única alternativa posible para satisfacer toda la demanda con la flota disponible.

En la mayor parte de los modelos suele suponerse que la cantidad de vehículos disponibles no está acotada. Este tipo de modelos resulta de utilidad, principalmente, en un contexto de planificación estratégica en el cual se desea dimensionar la flota de vehículos o simplemente se busca una idea aproximada del costo de la distribución. En gran parte de los casos, especialmente a nivel operativo, debe resolverse el problema considerando la flota disponible. Es decir, debe considerarse una cantidad finita de vehículos.

Resulta extraño que siendo características esenciales de diversos problemas de la realidad, estos aspectos sean tan pocas veces tenidos en cuenta en la literatura. Sorprende aún más el hecho de que la mayor parte del software comercial de ruteo de vehículos sí ofrezca soluciones a este tipo de problemas [77].

En este capítulo se extiende la formulación del VRP para considerar múltiples rutas por vehículo y una flota de tamaño fijo. Se formula el problema como un problema de

programación lineal entera y se da una descomposición del VRPMT en dos subproblemas. Luego se estudia la complejidad computacional y se da cotas para el valor óptimo. Se presenta además algunas medidas de infactibilidad de las soluciones y, finalmente, se da un relevamiento bibliográfico de los trabajos que proponen algoritmos para resolver esta importante variante del VRP.

4.2. Definición del problema

Se considera un grafo dirigido $G = (V, E)$ donde $V = \{0, 1, \ldots, n\}$ es el conjunto de nodos y $E \subseteq V \times V$ es el conjunto de arcos. El nodo 0 representa un depósito y los nodos $i \in V \setminus \{0\}$ representan clientes. Cada cliente i tiene asociada una demanda d_i. Si $(i, j) \in E$, es posible viajar directamente desde i hasta j a un costo c_{ij} y en un tiempo t_{ij}.

Se dispone de una flota $K = \{1, ..., m\}$ de m vehículos idénticos (la flota es homogénea). Cada vehículo tiene una capacidad Q. Existe un horizonte de tiempo T que indica la duración del período de planificación.

Se supone que Q, d_i y T son enteros positivos y que G es fuertemente conexo. También se considera que $t_{ij} = \lambda c_{ij}\ \forall\, (i, j) \in E$ para cierto $\lambda > 0$, lo cual resulta razonable en la práctica pues los costos de viaje suelen considerar principalmente el gasto de combustible y son proporcionales al tiempo de utilización de los vehículos.

Una ruta es un ciclo simple en G que comienza y termina en el depósito y será denotada como $r = (v_0, \ldots, v_{n(r)+1})$, donde $v_0 = v_{n(r)+1} = 0$ y $(v_i, v_{i+1}) \in E\ \forall\, i \in 0..n(r)$. Para cada ruta r, $n(r)$ representa la cantidad de clientes que visita. La demanda cubierta por una ruta, su costo y su duración están dadas, respectivamente, por

$$d(r) = \sum_{i=1}^{n(r)} d_{v_i},\ c(r) = \sum_{i=0}^{n(r)} c_{v_i, v_{i+1}} \text{ y } t(r) = \sum_{i=0}^{n(r)} t_{v_i, v_{i+1}}.$$

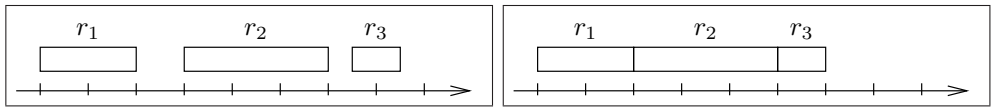
En algunos casos se utilizará c_r y t_r para denotar el costo y la duración de una ruta.

El Problema de Ruteo de Vehículos con Múltiples Viajes o *Vehicle Routing Problem with Multiple Trips* (VRPMT) consiste en determinar un conjunto de rutas junto con una asignación de cada ruta a un vehículo, de modo que el costo total de las rutas sea mínimo y se satisfagan las siguientes restricciones:

(i) **Cubrimiento**: cada cliente debe ser visitado por exactamente una ruta,

(ii) **Capacidad**: la demanda de los clientes en la misma ruta no debe exceder la capacidad del vehículo que la recorre,

(iii) ***Overtime*** **(horas extra)**: cada vehículo debe poder recorrer las rutas que tiene asignadas dentro del período de planificación.



(a) Secuencia con esperas (b) Secuencia sin esperas

Figura 4.1: Dos diagramas de Gantt para una misma secuencia de rutas.

Una solución para el problema es una secuencia de conjuntos de rutas

$$s = (R_1(s), \ldots, R_m(s))$$

donde el conjunto $R_k(s)$ está formado por las rutas asignadas al vehículo k. El conjunto de rutas de la solución es

$$R(s) = \bigcup_{k \in K} R_k(s).$$

En los casos en que no haya ambigüedades, se denotará una solución simplemente como $s = (R_1, \ldots, R_m)$. El costo de una solución es la suma de los costos de cada ruta:

$$f(s) = \sum_{r \in R(s)} c(r). \tag{4.1}$$

Dado un vehículo k denotamos como $t_k(s)$ a la duración total de las rutas que tiene asignadas:

$$t_k(s) = \sum_{r \in R_k(s)} t(r). \tag{4.2}$$

Las restricciones del problema pueden ser escritas utilizando esta notación como:

(i) $\forall\, i \in V \setminus \{0\}$, $\exists!\, r = (v_0, \ldots, i, \ldots, v_{n(r)+1}) \in R(s)$,

(ii) $d(r) \leq Q \;\forall\, r \in R(s)$,

(iii) $t_k(s) \leq T \;\forall\, k \in K$.

4.2.1. Orden de recorrida de las rutas

En este problema, la duración de una ruta depende únicamente de la secuencia de nodos que visita. Además, se supone que un vehículo puede finalizar una ruta y comenzar la siguiente sin tener que esperar en el depósito. Dado que la duración total de una secuencia con esperas no puede ser menor que la duración de la misma secuencia sin esperas (ver Figura 4.1), solamente se considerará secuencias sin espera.

Verificar si para un vehículo k se cumple la restricción de overtime, implica determinar si es posible recorrer las rutas de $R_k(s)$ en un tiempo no mayor que T. Dadas las hipótesis presentadas en el párrafo anterior, todos los $|R_k(s)|!$ ordenes posibles para recorrer dichas rutas tienen una duración de $t_k(s)$. Por lo tanto, se satisface la restricción si $t_k(s) \leq T$.

Frente a problemas con restricciones adicionales, ese chequeo de factibilidad podría ser más complejo. En los problemas con ventanas de tiempo, por ejemplo, la duración de una ruta depende del momento en que ésta comience. Esto ocurre porque si se arriba a un cliente antes de que empiece su horario de servicio, el vehículo debe esperar. Además, como no se puede llegar a un cliente luego de que su horario finalice, existen restricciones sobre la hora de comienzo de las rutas. En esos casos, el problema de determinar si un conjunto de rutas puede ser recorrido por un vehículo dentro del período de planificación puede formularse como un *One Machine Problem* que es NP-Completo [57].

4.3. Formulaciones Matemáticas

En esta sección se dan dos formulaciones del VRPMT como un problema de programación entera binaria. Ambas son adaptaciones de modelos concebidos originalmente para el VRP. Se pretende dar una definición precisa del problema en cuestión. En este trabajo no se buscan métodos de solución basados en estos modelos ni se profundiza en su estudio.

4.3.1. Formulación como un Set Partitioning Problem

Es posible adaptar la formulación del VRP de Balinsky y Quandt [9] para formular el VRPMT como un Set Partitioning Problem (SPP) con restricciones adicionales. Sea S el conjunto que contiene a todas las rutas cuya demanda no supera Q, es decir, que respetan la restricción de capacidad. Para cada $r \in S$ se define a_{ir} indicando si la ruta r visita al cliente i ($a_{ir} = 1$) o no ($a_{ir} = 0$). El VRPMT puede ser formulado de la siguiente manera:

$$\text{(VRPMT1)} \min \sum_{k \in K} \sum_{r \in S} c_r x_r^k \tag{4.3}$$

$$\text{s.a.} \sum_{k \in K} \sum_{r \in S} a_{ir} x_r^k = 1 \qquad \forall\, i \in V \setminus \{0\} \tag{4.4}$$

$$\sum_{r \in S} t_r x_r^k \leq T \qquad \forall\, k \in K \tag{4.5}$$

$$x_r^k \in \{0,1\} \qquad \forall\, k \in K, \forall\, r \in S$$

Si $x_r^k = 1$, entonces la ruta r es parte de la solución y es asignada al vehículo k (es decir, $r \in R_k(s)$). Si $x_r^k = 0 \,\forall\, k \in K$, entonces r no forma parte de la solución. La función objetivo (4.3) establece que se debe minimizar el costo total. El hecho de que cada cliente deba pertenecer a exactamente una ruta se impone en (4.4) y las restricciones de overtime se establecen en (4.5).

A pesar de tener una cantidad exponencial de variables, esta formulación enfatiza la estructura de las restricciones del VRPMT. La capacidad de los vehículos y el hecho de que las rutas deban comenzar y finalizar en el depósito son restricciones *locales* a cada

ruta y, por lo tanto, pueden encapsularse en la definición del conjunto S. Sin embargo, visitar a cada cliente exactamente una vez y no sobrepasar el horizonte de tiempo son *globales*, en el sentido que involucran a todas las rutas de la solución. Las violaciones de las restricciones locales pueden encontrarse examinando las rutas por separado, pero para asegurar el cumplimiento de las restricciones globales debe analizarse toda la solución.

4.3.2. Formulación del tipo Flujo de Vehículos

En el trabajo de Zhao et al. [137] se propone un modelo del VRPMT basado en la formulación de flujo de vehículos de tres índices. Se agrega un cuarto índice para discriminar entre las diferentes rutas asignadas a un mismo vehículo. Se utiliza un conjunto de variables binarias x_{ij}^{kp} que indica si el vehículo $k \in K$ viaja del nodo i al nodo j en su p-ésimo viaje. Para determinar el rango del índice p se utiliza una cota superior $\bar{p}$ para la cantidad de rutas que puede asignarse a un mismo vehículo como parámetro del modelo (si no hubiera tal cota, puede considerarse $\bar{p} = n$). La formulación es la siguiente:

$$\text{(VRPMT2)} \min \sum_{k \in K} \sum_{p=1}^{\bar{p}} \sum_{(i,j) \in E} c_{ij} x_{ij}^{kp} \tag{4.6}$$

$$\text{s.a.} \sum_{k \in K} \sum_{p=1}^{\bar{p}} \sum_{j \in \Delta^+(i)} x_{ij}^{kp} = 1 \qquad \forall\, i \in V \setminus \{0\} \tag{4.7}$$

$$\sum_{j \in \Delta^+(i)} x_{ij}^{kp} - \sum_{j \in \Delta^-(i)} x_{ji}^{kp} = 0 \qquad \begin{array}{r} \forall\, i \in V \setminus \{0\}, \\ \forall\, k \in K,\ \forall\, p \in 1, \ldots, \bar{p} \end{array} \tag{4.8}$$

$$\sum_{i \in V \setminus \{0\}} d_i \sum_{j \in \Delta^+(i)} x_{ij}^{kp} \leq Q \qquad \forall\, k \in K,\ \forall\, p \in 1, \ldots, \bar{p} \tag{4.9}$$

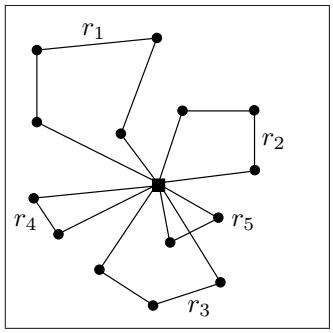
$$\sum_{p=1}^{\bar{p}} \sum_{(i,j) \in E} t_{ij} x_{ij}^{kp} \leq T \qquad \forall\, k \in K \tag{4.10}$$

$$\sum_{i \in S} \sum_{j \in S \cap \Delta^+(i)} x_{ij}^{kp} \leq |S| - 1 \qquad \begin{array}{r} \forall\, S \subseteq V \setminus \{0\},\ |S| \geq 2, \\ \forall\, k \in K,\ \forall\, p \in 1, \ldots, \bar{p} \end{array} \tag{4.11}$$

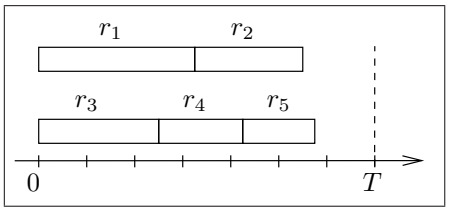
$$x_{ij}^{kp} \in \{0, 1\} \qquad \forall\, (i,j) \in E,\ \forall\, k \in K,\ \forall\, p \in 1, \ldots, \bar{p}$$

La función objetivo (4.6) indica que se debe minimizar el costo de todas las rutas. Las restricción (4.7) exige que todo cliente sea un nodo intermedio de exactamente una ruta. En (4.8) se impone que si el vehículo k llega al cliente i en su viaje p, entonces el mismo vehículo debe abandonar el cliente en ese mismo viaje. La capacidad de cada vehículo y el horizonte de tiempo se imponen en (4.9) y (4.10), respectivamente. Finalmente, las desigualdades (4.11) actúan como restricciones de eliminación de subtours.

Dado que se hace explícito el orden de las rutas, esta formulación tiene problemas de degeneración.



(a) Componente de ruteo



(b) Componente de asignación

Figura 4.2: Dos componentes de una solución del VRPMT.

4.4. Descomposición del problema

El VRPMT exige, además de diseñar un conjunto de rutas, hacer explícita una asignación de dichas rutas a los m vehículos disponibles. Por lo tanto, las soluciones del VRPMT tienen dos componentes: el ruteo y la asignación.

Componente de ruteo: un conjunto de rutas R que satisface las restricciones del VRP,

Componente de asignación: una partición de R en $(R_1, \ldots, R_m)$ de modo que cada conjunto R_k satisface la restricción de horizonte de tiempo.

Dado un conjunto R de rutas factibles para el VRP, llamaremos *subproblema de asignación* al de determinar una solución s factible para el VRPMT cuyas rutas sean las de R (es decir, $R(s) = R$). Esto es, obtener una solución del VRPMT partiendo de una solución factible para la componente de ruteo. Se busca una partición de R en m subconjuntos $R_1, \ldots, R_m$ de modo que $t_k(s) = \sum_{r \in R_k} t(r) \leq T \; \forall \, k \in K$. Este problema puede verse como un *Bin Packing Problem* (BPP) con m recipientes de capacidad T y un item de peso $t(r)$ por cada ruta $r \in R$. En el Apéndice B se da una descripción del BPP.

En la Figura 4.2 se ilustran las dos componentes de una solución para una instancia del problema con $m = 2$. Dicha solución es $s = (R_1, R_2)$ con $R_1 = \{r_1, r_2\}$ y $R_2 = \{r_3, r_4, r_5\}$. Cabe mencionar que si bien en el diagrama de Gantt se muestra un orden en el cual las rutas son recorridas (ver Figura 4.2(b)), dicho orden no forma parte de la solución. En efecto, los R_k son conjuntos y no secuencias. El ordenamiento se da solamente a los efectos de representar gráficamente una solución.

Es importante notar que el conjunto de rutas de una solución factible para el VRPMT siempre es factible para el VRP. Sin embargo, el recíproco de esta afirmación es falso. Puede haber soluciones factibles para el VRP cuyas rutas no puedan asignarse a los vehículos disponibles respetando las restricciones de overtime. Es decir, el subproblema de asignación puede no tener solución para ciertos conjuntos de rutas R. Por lo tanto, resolver un VRP y luego asignar las rutas a los vehículos resolviendo un BPP puede dar lugar a soluciones

no factibles respecto al horizonte de tiempo.

4.5. Complejidad computacional

El hecho de que la cantidad de vehículos sea fija hace que la restricción de horizonte de tiempo sea difícil de satisfacer en el caso general. Por un lado, la duración de las rutas asignadas a un mismo vehículo aumenta cuando la cantidad de vehículos disminuye (pues algunos de los vehículos restantes deben realizar más trabajo). Por otro lado, el horizonte de tiempo impone una cota superior a la duración de dichas rutas. Ciertas combinaciones de m y T pueden dar lugar a instancias no factibles.

Denominaremos *problema de factibilidad del VRPMT* al de, dada una instancia, encontrar una solución factible o determinar que no existe. Dicho problema es NP-Hard. Esto puede probarse reduciendo polinomialmente el BPP a este problema. Consideremos una instancia de la versión de decisión del BPP. Sean I el conjunto de items, w_i el peso del item i, U la cantidad de recipientes y B la capacidad de cada uno de ellos. A partir de esa instancia, se construye (en tiempo polinomial) una instancia del VRPMT donde:

1. $V = \{0\} \cup I$ y $E = V \times V$,
2. $d_i = 1 \; \forall \, i \in I$ y $Q = 1$,
3. $c_{0i} = c_{i0} = t_{0i} = t_{i0} = w_i/2 \; \forall \, i \in I$,
4. $m = U$,
5. $T = B$.

En esta instancia del VRPMT existe un cliente por cada item (1). Como la capacidad del vehículo alcanza para satisfacer la demanda de exactamente un cliente (2), cualquier ruta en una solución factible será de la forma $r_i = (0, i, 0)$. La definición de los tiempos de viaje (3) implica que $t(r_i) = c_{0i} + c_{i0} = w_i$. Por lo tanto, en una solución factible cada ruta representa un item y la duración de la ruta es igual al peso del item. Por otro lado, la cantidad de vehículos es igual a la cantidad de recipientes (4) y el horizonte de tiempo es igual a la capacidad de cada recipiente (5).

Consideremos una solución factible $s = (R_1(s), \ldots, R_m(s))$ para esta instancia del VRPMT. Dicha solución puede interpretarse como una solución del BPP original en la que si $r_i \in R_k(s)$, entonces el item i está asignado al recipiente k. Como s es factible para el VRPMT, se cumple que $\sum_{r_i \in R_k} t(r_i) \leq T \; \forall k \in K$, es decir, $\sum_{r_i \in R_k} w_i \leq B$. En consecuencia, el peso de los items asignados al recipiente k no supera la capacidad de dicho recipiente. Finalmente, como hay tantos vehículos como recipientes, la factibilidad de la solución del VRPMT garantiza la factibilidad de la solución del BPP.

Los párrafos anteriores ilustran como un algoritmo para resolver el problema de factibilidad del VRPMT en tiempo polinomial, puede utilizarse para resolver la versión de decisión del BPP en tiempo polinomial. Como dicha versión del BPP es un problema NP-Completo [57], el problema de factibilidad del VRPMT es NP-Hard.

Para muchos problemas de optimización, hallar una solución óptima es NP-Hard, pero una solución factible puede encontrarse en tiempo polinomial. En ese sentido, el VRPMT puede considerarse aún más difícil que esa clase de problemas, pues no puede computarse soluciones factibles en tiempo polinomial (a no ser que $P = NP$). La complejidad computacional del VRPMT lo hace poco tratable mediante técnicas de solución exacta y justifica el uso de métodos aproximados o heurísticas para poder resolver instancias de tamaño realista en tiempos de cómputo razonables.

4.6. Cotas para el valor óptimo

Al trabajar con problemas de optimización de cierta complejidad, se busca obtener soluciones factibles cuyo valor sea lo más cercano posible al valor óptimo del problema. De aquí en más, se supondrá que el valor de cualquier solución factible es estríctamente positivo. Siendo z^* el valor óptimo de una instancia de un problema de minimización y dada una solución factible de costo z, interesa estimar

$$GAP^*(z) = \frac{z - z^*}{z^*}$$

que otorga una medida normalizada de la calidad de la solución. Como z^* es desconocido *a priori*, suele utilizarse la siguiente medida

$$GAP(z) = \frac{z - LB}{LB} \tag{4.12}$$

como estimativo de $GAP^*(z)$, siendo LB una cota inferior para z^*. Dado que $LB \leq z^*$, se cumple que $GAP^*(z) \leq GAP(z)$. Cuanto mas ajustada sea la cota inferior, es decir, cuanto menor sea $z^* - LB$, mejor será dicha estimación.

Por otro lado, dada una instancia de un problema, puede ser de utilidad contar con una cota superior para el valor de las soluciones factibles. Es decir, un valor UB tal que si una solución factible tiene costo z, pueda garantizarse que $z \leq UB$. En este caso, se cumple que

$$GAP(z) \leq \frac{UB - LB}{LB}.$$

Es decir, puede determinarse una cierta garantía en la calidad de las soluciones factibles. Por ejemplo, si $\frac{UB-LB}{LB} = 0.05$, entonces toda solución factible del problema tiene un GAP que no supera el 5 % y quizás sea suficiente para los aspectos prácticos diseñar un algoritmo

que solamente busque la factibilidad. Además, si se diera un caso en el que $LB > UB$, entonces la instancia no tiene soluciones factibles.

4.6.1. Una cota inferior para el VRPMT

Consideremos una instancia factible del VRPMT de valor óptimo z^*_{VRPMT}. Puesto que las rutas de cualquier solución factible del VRPMT son factibles para el VRP correspondiente, se cumple que

$$z^*_{\text{VRPMT}} \geq z^*_{\text{VRP}}, \tag{4.13}$$

siendo z^*_{VRP} el valor óptimo del VRP asociado a la instancia del VRPMT, lo cual da una cota inferior.

Puede probarse que esta cota es alcanzada por algunas instancias del VRPMT. En efecto, consideremos un instancia cualquiera del VRP cuya solución óptima está formada por el conjunto de rutas $R = \{r_1, \ldots, r_p\}$. Para esta instancia tenemos que $z^*_{\text{VRP}} = \sum_{r \in R} c(r)$. Eligiendo, por ejemplo, $\lambda = 1$, $m = p$ y $T = \max_{r \in R} c(r)$, se obtiene una instancia del VRPMT en la cual la solución $s^* = (\{r_1\}, \ldots, \{r_p\})$ es factible y su valor es z^*_{VRP}. Por lo tanto, s^* es la solución óptima para la instancia construida y en esta instancia se cumple que $z^*_{\text{VRPMT}} = z^*_{\text{VRP}}$.

4.6.2. Una cota superior para el VRPMT

Consideremos una instancia del VRPMT y una solución factible $s^* = (R_1(s^*), \ldots, R_m(s^*))$. Como s^* es factible, verifica la restricción de overtime:

$$\sum_{r \in R_k(s^*)} t(r) \leq T \; \forall k \in K,$$

lo cual implica (agregando las m desigualdades) que

$$\sum_{r \in R(s^*)} t(r) \leq mT.$$

Dado que s^* es factible para la instancia y que para cierto $\lambda > 0$ se cumple que $t_{ij} = \lambda c_{ij}$ $\forall\, (i,j) \in E$:

$$z^*_{\text{VRPMT}} = \sum_{r \in R(s^*)} c(r) = \frac{1}{\lambda} \sum_{r \in R(s^*)} t(r).$$

Se deriva entonces la siguiente cota superior

$$z^*_{\text{VRPMT}} \leq \frac{mT}{\lambda}. \tag{4.14}$$

También esta cota es alcanzada en algunas instancias. Dados m, T, $\lambda > 0$ y $n \geq m$, es

posible construir instancias del VRPMT que alcancen la cota superior, fijando:

1. $d_i = 1 \; \forall \, i \in V \setminus \{0\}$ y $Q = 1$,

2. $t_{0i} = t_{i0} = T/2 \; \forall i \in 1..(m-1)$,

3. $t_{0i} = t_{i0} = T/(2(n-m+1)) \; \forall i \in m..n$.

Considerando la solución $s^* = (R_1(s^*), \ldots, R_m(s^*))$, donde $R_k(s^*) = \{r_k\} \; \forall k = 1..(m-1)$ y $R_m(s^*) = \{r_m, \ldots, r_n\}$. Esta solución es factible para el problema, pues cada ruta satisface las restricciones de capacidad y además tenemos

$$\sum_{r \in R_k(s^*)} t_r = t_{0k} + t_{k0} = T \quad \forall \, k = 1..(m-1) \text{y}$$

$$\sum_{r \in R_m(s^*)} t_r = \sum_{i=m}^{n} t_{0i} + t_{i0} = \sum_{i=m}^{n} T/(n-m+1) = T.$$

Además, dado que la demanda de cada cliente es igual a la capacidad del vehículo, para cualquier otra solución factible s se tiene que $R(s) = R(s^*)$ y por lo tanto, s^* es una solución óptima. El costo de s^*, y por lo tanto el valor óptimo de la instancia del VPRMT construida, está dado por

$$\sum_{r \in R(s^*)} c(r) = \sum_{k=1}^{m-1} \sum_{r \in R_k(s^*)} \frac{t(r)}{\lambda} + \sum_{r \in R_m(s^*)} \frac{t(r)}{\lambda} = \frac{mT}{\lambda},$$

por lo tanto, en la instancia construida se alcanza la cota superior.

4.7. Medidas de infactibilidad

Dada la complejidad computacional asociada al problema de encontrar soluciones que respeten todas las restricciones del problema, resulta útil contar con medidas para evaluar la calidad de las soluciones no factibles. Se considera una solución $s = (R_1(s), \ldots, R_m(s))$, que no necesariamente satisface las restricciones de capacidad o de horizonte de tiempo.

4.7.1. Violaciones a la restricción de capacidad

Una medida de la violación de la restricción de capacidad para una ruta $r \in R(s)$ está dada por:

$$D(r) = (d(r) - Q)^+ \tag{4.15}$$

siendo $a^+ = \max(a, 0)$. Si la ruta es factible respecto a esta restricción, entonces $D(r) = 0$. Si no lo es, el valor de $D(r)$ indica la mínima cantidad de capacidad adicional necesaria

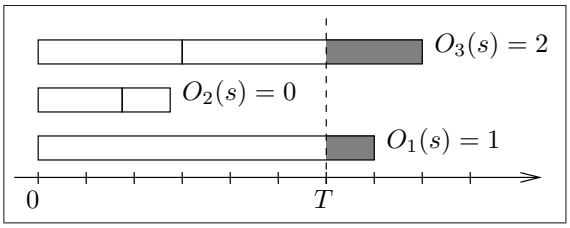


Figura 4.3: Un ejemplo de la medida overtime para una solución no factible.

para que el vehículo pueda cargar la demanda de la ruta. Esta medida puede agregarse para considerar todas las rutas de la solución:

$$D(s) = \sum_{r \in R(s)} D(r) \tag{4.16}$$

y se cumple que $D(s) = 0$ si s es factible respecto a la restricción de capacidad.

4.7.2. Violaciones a la restricción de overtime

Taillard et al. Taillard et al. [120] proponen tres medidas para cuantificar la infactibilidad de una solución respecto a la restricción de overtime.

Overtime (horas extra)

Dado un vehículo $k \in K$, el tiempo que éste trabaja luego de transcurrido el horizonte de tiempo T se utiliza para definir la medida *overtime*:

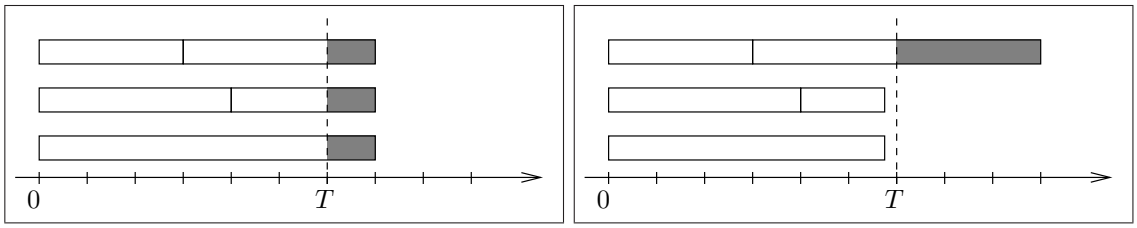
$$O_k(s) = \left(\sum_{r \in R_k(s)} t(r) - T \right)^+ . \tag{4.17}$$

Si el vehículo satisface la restricción de horizonte de tiempo, entonces $O_k(s) = 0$. En caso contrario, la ecuación puede re-escribirse como

$$\sum_{r \in R_k(s)} t(r) = T + O_k(s). \tag{4.18}$$

Es decir, el tiempo total que un vehículo que no respeta la restricción de overtime está trabajando, es el horizonte de tiempo más su overtime.

En la Figura 4.3 se muestra el diagrama de Gantt para una solución en la que los vehículos 1 y 3 violan la restricción de horizonte de tiempo ($O_1(s) = 1$ y $O_3(s) = 2$) y el vehículo 2 la satisface ($O_2(s) = 0$).



(a) Solución con $O(s) = 3$ y $LTR(s) = 7/6$ (b) Solución con $O(s) = 3$ y $LTR(s) = 9/6$

Figura 4.4: Dos soluciones con el mismo valor de *overtime*.

El *overtime total* de una solución se obtiene sumando en todos los vehículos:

$$O(s) = \sum_{k \in K} O_k(s). \tag{4.19}$$

Al ser una medida agregada, el overtime total puede no proporcionar cierta información relevante. Por ejemplo, el overtime total de las dos soluciones que se muestran en la Figura 4.4 es 3. Sin embargo, en la solución de la Figura 4.4(a) cada vehículo sobrepasa en 1 el horizonte de tiempo, mientras que en la de la Figura 4.4(b) hay un único vehículo que no satisface la restricción y sobrepasa el horizonte en 3 unidades de tiempo. Este fenómeno no es capturado por la medida propuesta. En algunas situaciones reales puede ser preferible obtener soluciones en las que las violaciones de cada vehículo sean pequeñas (aunque ocurran en muchos de ellos) y en otros casos puede preferirse que el overtime se concentre en unos pocos vehículos.

En algunos casos, como cuando se desea comparar valores obtenidos para diferentes instancias del problema, puede ser útil normalizar esta medida de modo que no dependa de las magnitudes específicas involucradas en cada instancia. Puede entonces medirse el overtime total en términos del horizonte de tiempo, es decir

$$OT(s) = \frac{O(s)}{T}. \tag{4.20}$$

Longest Tour Ratio

Teniendo en cuenta lo anterior, la medida *longest tour ratio* (LTR) da otra manera de cuantificar la infactibilidad de una solución, considerando únicamente el vehículo que llega más tarde al depósito luego de su último viaje, es decir, el peor caso:

$$LTR(s) = \frac{1}{T} \max_{k \in K} \sum_{r \in R_k(s)} t(r) \tag{4.21}$$

Si la solución es factible, entonces $LTR(s) \leq 1$. Si no, utilizando (4.18), tenemos que

$$LTR(s) = \frac{1}{T} \max_{k \in K} (T + O_k(s)) = 1 + \frac{\max_{k \in K} O_k(s)}{T}. \tag{4.22}$$

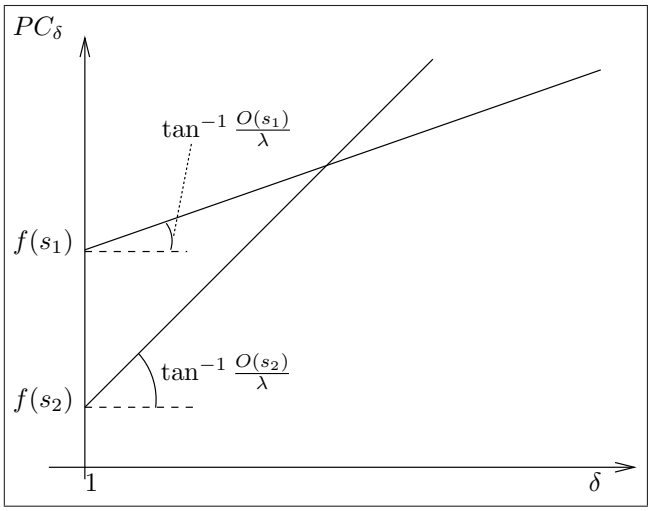


Figura 4.5: PC_δ vs. δ para dos soluciones s_1 y s_2.

Es decir, se mide el overtime del vehículo que llega más tarde en términos del horizonte de tiempo.

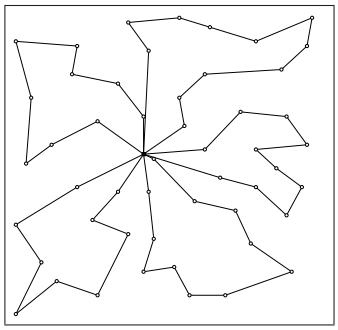
Penalized Cost

Esta medida combina las violaciones de la restricción de horizonte de tiempo con los costos del problema. El *costo penalizado* surge de suponer que los costos de viaje se multiplican por un parámetro $\delta > 1$ luego de que se cumple el horizonte de tiempo. Es decir, al cabo de T unidades de tiempo, viajar del nodo i al j aumenta su costo de c_{ij} a δc_{ij}. Dado que el tiempo en que se realizan viajes una vez finalizado el horizonte T es $O(s)$, una expresión simple para esta medida es

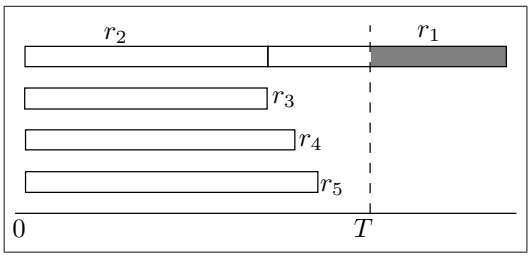
$$PC_\delta(s) = f(s) + \frac{(\delta - 1)O(s)}{\lambda} \tag{4.23}$$

Esta medida depende de un parámetro y, por lo tanto, puede ocurrir que dadas dos soluciones s_1 y s_2 se cumpla $PC_{\delta_1}(s_1) > PC_{\delta_1}(s_2)$ pero $PC_{\delta_2}(s_1) < PC_{\delta_2}(s_2)$, para dos valores diferentes del parámetro δ_1 y δ_2. Por ejemplo, dada una solución s_1 podría ser posible obtener otra solución s_2 que sea mejor en términos de la función objetivo ($f(s_2) < f(s_1)$) a costa de un incremento en la infactibilidad ($O(s_2) > O(s_1)$). En este caso podría darse un comportamiento similar al que se ilustra en la Figura 4.5 para la medida PC_δ al variar δ. Debe observarse que es el valor de $O(s)$ el que determina la sensibilidad de esta medida con respecto al valor de δ, dado que

$$\frac{\partial}{\partial \delta} PC_\delta(s) = \frac{O(s)}{\lambda}.$$

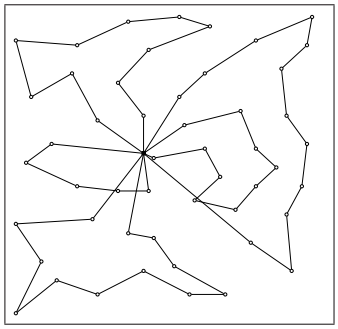


(a) Rutas de la solución.

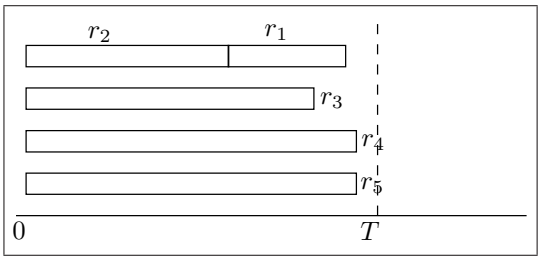


(b) Asignación (no factible).

Figura 4.6: Solución óptima del VRP y no factible para el VRPMT.



(a) Rutas de la solución.



(b) Asignación (factible).

Figura 4.7: Solución factible para el VRPMT.

4.8. Ejemplo

Se considera la instancia CMT-1 propuesta por Christofides et al. [28] para el VRP. Para esta instancia se conoce el valor óptimo $z^*_{\text{VRP}} = 524.61$. Considerando los mismos nodos, demandas y capacidades que en dicha instancia junto con $m = 4$, $T = 144$ y $\lambda = 1$ se construye una instancia para el VRPMT. Para esta instancia $LB = z^*_{\text{VRP}} = 524.61$ y $UB = mT/\lambda = 576$, por tanto el máximo GAP posible es $(UB - LB)/LB = 9.8\,\%$.

La solución óptima para el subproblema de ruteo (es decir el VRP sin considerar los valores de m y T), se muestra en la Figura 4.6. Dicha solución (de costo 524.61) está compuesta por 5 rutas ($R = \{r_1, \ldots, r_5\}$) que se muestran en la Figura 4.6(a) y cuyas duraciones son $t_1 = 98.45$, $t_2 = 99.25$, $t_3 = 99.33$, $t_4 = 109.06$ y $t_5 = 118.52$.

No existe una solución factible s para la instancia del VRPMT que cumpla $R(s) = R$, pues no es posible colocar items de peso $t_1, \ldots, t_5$ en 4 recipientes de capacidad 144. Una solución no factible que puede construirse con las rutas de la solución del VRP es $s_1 = (\{r_1, r_2\}, \{r_3\}, \{r_4\}, \{r_5\})$ (ver Figura 4.6(b)).

Sin embargo, la instancia del VRPMT construida tiene al menos una solución factible. En la Figura 4.7 se muestran las rutas y la asignación correspondiente. La solución está formada por 5 rutas diferentes ($R(s') = \{r'_1, \ldots, r'_5\}$) de duraciones $t'_1 = 57.82$, $t'_2 = 82.12$, $t'_3 = 122.80$, $t'_4 = 141.76$ y $t'_5 = 141.79$ y $s' = (\{r'_1, r'_2\}, \{r'_3\}, \{r'_4\}, \{r'_5\})$. El costo de esta

	Sol. de Fig 4.6	Sol. de Fig 4.7
Factible	No	Si
f	524.61	546.29
GAP	–	4.1 %
O	53.7	0
LTR	1.373	0.985
PC_2	578.31	546.29
PC_3	623.01	546.29

Tabla 4.1: Algunas medidas para las soluciones del ejemplo.

solución es 546.29.

En la Tabla 4.1 se muestran algunas de las medidas definidas en las secciones precedentes, para las soluciones del ejemplo. Este ejemplo ilustra el hecho de que un enfoque de solución en dos fases, que primero resuelve un VRP y luego asigna las rutas a los vehículos puede dar lugar a soluciones no factibles, mencionado en la Sección 4.4.

4.9. Revisión bibliográfica

El primer trabajo acerca de este problema se atribuye a Fleischmann [51] y fue escrito en 1990. En su artículo, propuso la versión que se ataca en este trabajo y dos extensiones para considerar flota heterogénea y ventanas de tiempo. Se propone un algoritmo simple de dos fases. El subproblema de ruteo se resuelve mediante una adaptación del algoritmo de ahorros [29] y el de asignación mediante una heurística para el BPP. Sólamente se reportan resultados computacionales sobre 3 instancias de prueba cuyos datos no están disponibles.

Taillard et al. [120] propusieron en 1996 un algoritmo de tres fases para resolver el VRPMT. En la primera fase se genera, mediante varias ejecuciones de tabu search, un conjunto M con varias rutas que satisfacen las restricciones del VRP. Luego, se forman diversas soluciones para el VRP combinando adecuadamente las rutas almacenadas en M. En la última etapa, para cada solución del VRP las rutas son asignadas a los vehículos mediante una heurística para el BPP, procurando respetar la restricción de horizonte de tiempo. Se propone un conjunto de 104 instancias de prueba sobre los que se reportan resultados computacionales.

En 1997, Brandão y Mercer [18] diseñaron un algoritmo basado en tabu search para resolver un caso real que incluía, entre otras restricciones, las del VRPMT. Más tarde, simplificaron dicho algoritmo para resolver el VRPMT [19]. Ambas propuestas comparten sus características esenciales. Se permite visitar soluciones no factibles (que son penalizadas en la función objetivo), las movidas se definen mediante intercambios de clientes entre rutas y se utiliza el algoritmo GENI [59] para realizar las inserciones. La restricción de horizonte de tiempo se considera en una etapa intermedia del algoritmo. Se dan resultados

computacionales sobre las instancias de prueba propuestos por Taillard et al. [120].

Recientemente, Petch y Salhi [104] propusieron un algoritmo de múltiples fases para resolver el VRPMT. Se genera un conjunto de soluciones para el VRP sobre las que luego se ejecuta una heurística buscando asignar las rutas a los vehículos. Las rutas se construyen mediante dos vias: utilizando la versión de Yellow del algoritmo de ahorros [136] e independientemente, utilizando una población de rutas como en Taillard et al. [120]. También se reportan resultados sobre las instancias de prueba propuestos por Taillard et al. [120].

Zhao et al. [137] reportan una adaptación de tabu search en la cual antes de realizar las movidas se aplica un algoritmo de asignación de rutas a vehículos. Se reportan resultados computacionales sobre 3 instancias de prueba.

Capítulo 5

Memorias adaptativas para la resolución del VRPMT

5.1. Introducción

Como se mencionó en el Capítulo 3, el *Adaptive Memory Procedure* (AMP) [112] ha permitido obtener soluciones de buena calidad para algunos problemas de ruteo de vehículos como el VRP y el VRPTW. Además, provee un esquema general y flexible que permite su adaptación a nuevos problemas. En este capítulo se presenta un algoritmo basado en el AMP para resolver el VRPMT.

Las ideas utilizadas para el manejo de la memoria son muy similares a las propuestas originalmente por Rochat y Taillard [112]. En ambos casos la memoria es una secuencia ordenada de rutas, donde las rutas de las mejores soluciones ocupan las primeras posiciones. La construcción de una solución se realiza seleccionando rutas de la memoria probabilísticamente.

El algoritmo de búsqueda local toma elementos de *Taburoute* [60], como la aceptación de soluciones no factibles y la estrategia para su penalización. Además, se utilizan las ideas presentadas en el *Granular Tabu Search* propuesto por Toth y Vigo [127], para disminuir la cantidad de soluciones examinadas al explorar la vecindad. Se proponen dos estrategias para implementar dicha reducción, que dan lugar a diferentes métodos de búsqueda local.

Durante la ejecución del AMP se permite visitar soluciones que violen la restricción de overtime. En la búsqueda local se permite, además, violar la restricción de capacidad de los vehículos. Dado que se aceptan soluciones no factibles, se define un criterio de comparación de soluciones para tener en cuenta tanto la factibilidad como el costo, a la hora de tomar las decisiones dentro del algoritmo.

Se diseñó una heurística simple para resolver el subproblema de asignación, que se utiliza para obtener una solución del VRPMT a partir de un conjunto de rutas.

Paso 1 (inicialización).
Hacer $R_k \leftarrow \emptyset$, para $k = 1, \ldots, m$.

Paso 2 (selección).
Seleccionar la ruta de mayor duración: $r^* \leftarrow \arg\max_{r \in R} t(r)$.
Seleccionar el vehículo menos ocupado: $k^* \leftarrow \arg\min_{k \in K} \sum_{r \in R_k} t(r)$.

Paso 3 (asignación).
Asignar r^* a k^*: $R_{k^*} \leftarrow R_{k^*} \cup \{r^*\}$.
Hacer $R \leftarrow R \setminus \{r^*\}$.

Paso 4 (terminación).
Si $R = \emptyset$, terminar; si no, ir al paso 2.

Figura 5.1: Heurística de asignación de rutas a vehículos.

5.2. Resolución del subproblema de asignación

En algunas fases del algoritmo, se contará con un conjunto de rutas R que debe ser particionado en m conjuntos para conseguir una solución $s = (R_1, \ldots, R_m)$ del VRPMT. Como se mencionó en la Sección 4.4, este subproblema puede modelarse como un BPP con m recipientes de capacidad T y un item de peso $t(r)$ por cada ruta $r \in R$. Dado que para algunos conjuntos R puede no existir una solución factible, se buscará una asignación que minimice el overtime total.

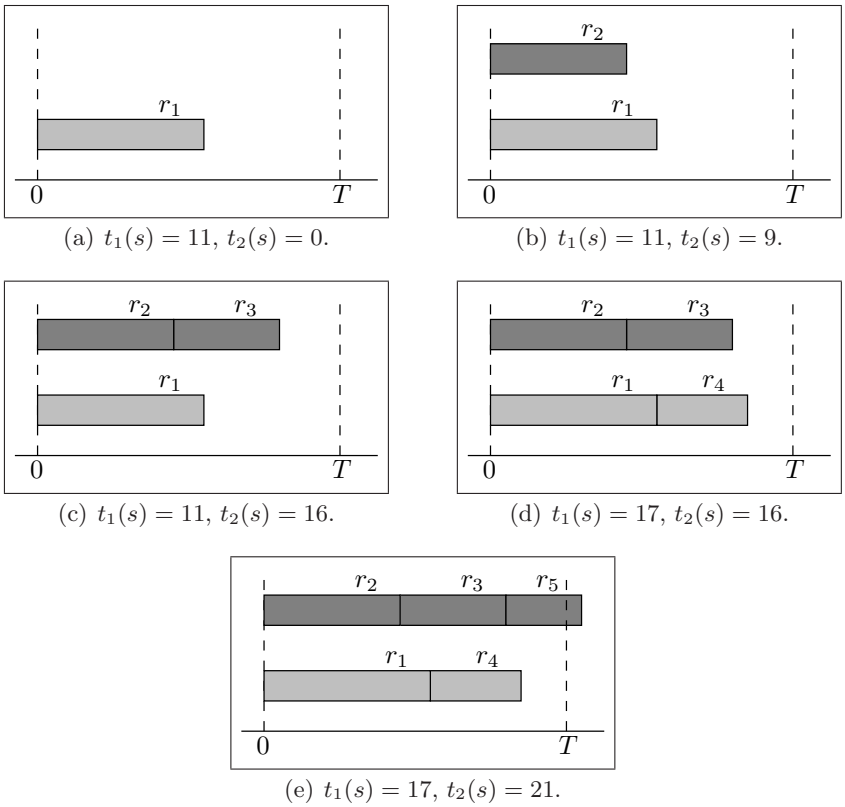
5.2.1. Heurística de asignación

Para obtener una asignación se utiliza un método iterativo que, en cada paso, asigna la ruta de mayor duración (considerando las que aún no han sido asignadas) al vehículo menos ocupado. El proceso termina cuando todas las rutas han sido asignadas. Un seudocódigo de esta heurística se muestra en la Figura 5.1. Este algoritmo es similar a la heurística *Best Fit Decreasing* (BFD) propuesta para el BPP [96].

En la Figura 5.2 se ilustra la ejecución de este algoritmo sobre un conjunto de 5 rutas $R = \{r_1, \ldots, r_5\}$ de duraciones 11, 9, 7, 6 y 5 respectivamente, para $m = 2$ y $T = 20$. La solución obtenida es $s = (R_1, R_2)$, formada por $R_1 = \{r_1, r_4\}$ y $R_2 = \{r_2, r_3, r_5\}$. Esta solución no es factible respecto a la restricción de overtime, siendo $O_1(s) = 0$ y $O_2(s) = 1$.

5.2.2. Heurística de post-procesamiento

Si bien la heurística de asignación puede funcionar adecuadamente en algunos casos, es posible que la partición obtenida pueda ser mejorada. Para ello se aplica un algoritmo de post-procesamiento sobre la asignación, buscando corregir algunas decisiones tomadas por la heurística. El objetivo es encontrar intercambios de rutas entre vehículos que disminuyan el overtime total.



(a) $t_1(s) = 11$, $t_2(s) = 0$.

(b) $t_1(s) = 11$, $t_2(s) = 9$.

(c) $t_1(s) = 11$, $t_2(s) = 16$.

(d) $t_1(s) = 17$, $t_2(s) = 16$.

(e) $t_1(s) = 17$, $t_2(s) = 21$.

Figura 5.2: Ejemplo de ejecución de la Heurística de Asignación.

Consideremos una solución s con $O(s) > 0$, dos vehículos diferentes k y k' y una ruta de cada vehículo $r \in R_k(s)$ y $r' \in R_{k'}(s)$, que cumplan

(a) $O_k(s) > 0$ y $O_{k'}(s) = 0$,

(b) $t(r) > t(r')$,

(c) $t_{k'}(s) + t(r) - t(r') \leq T$.

Si se construye una solución s^{new} de modo que la única diferencia con s es que r se asigna a k' y r' se asigna a k (es decir, se intercambian las asignaciones), las duraciones para los vehículos k y k' en esta solución son:

$$t_k(s^{\text{new}}) = t_k(s) - t(r) + t(r')$$
$$t_{k'}(s^{\text{new}}) = t_{k'}(s) + t(r) - t(r')$$

La condición (a) implica que $O_k(s) = t_k(s) - T > 0$ y la condición (b) implica que $t_k(s) > t_k(s^{\text{new}})$. Por lo tanto

$$O_k(s^{\text{new}}) = (t_k(s^{\text{new}}) - T)^+ < t_k(s) - T = O_k(s)$$

Es decir, en la nueva solución se disminuye el overtime del vehículo k. Esta disminución,

Paso 1 (inicialización).
Inicializar el conjunto de rutas marcadas $L = \emptyset$.

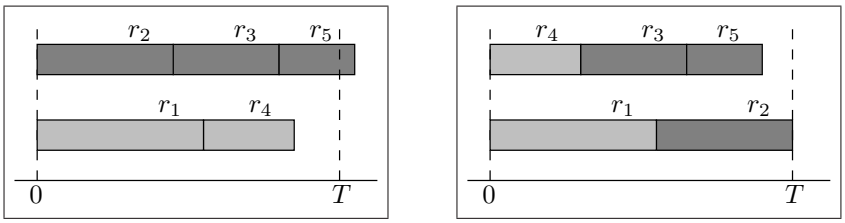
Paso 2 (selección del vehículo).
Seleccionar el vehículo que termina más tarde: $k \leftarrow \arg\max_{k \in K} t_k(s)$.

Paso 3 (selección de una ruta).
Si $R_k \subseteq L$, terminar (todas las rutas de k están marcadas).
Seleccionar una ruta del vehículo que no esté marcada: $r \in R_k \setminus L$.
Marcar la ruta $L = L \cup \{r\}$.

Paso 4 (buscar cambio).
Buscar $k' \in K$ y $r' \in R_{k'}$ que verifiquen las condiciones (a)-(c).
Si existen, ir al paso 5. Si no, ir al paso 3.

Paso 5 (realizar cambio).
Hacer $R_k \leftarrow R_k \cup \{r'\} \setminus \{r\}$ y $R_{k'} \leftarrow R_{k'} \cup \{r\} \setminus \{r'\}$.
Si $O(s) = 0$, terminar. Si no, ir al paso 2.

Figura 5.3: Algoritmo de post-procesamiento de la asignación.



(a) Solución obtenida con la heurística de asignación

(b) Intercambio de las rutas r_2 y r_4

Figura 5.4: Intercambio que mejora el resultado de la heurística de asignación.

trae consigo un aumento de la duración del vehículo k', puesto que $t_{k'}(s^{\text{new}}) > t_{k'}(s)$. La condición (c) asegura que $t_{k'}(s^{\text{new}}) \leq T$ y en consecuencia $O_{k'}(s^{\text{new}}) = O_{k'}(s) = 0$. En consecuencia, no se incrementa el overtime del vehículo k'. Por lo tanto, $O(s^{\text{new}}) < O(s)$, como se buscaba.

En el algoritmo de la Figura 5.3, se buscan vehículos y rutas que cumplan las condiciones (a)-(c) para realizar los intercambios mencionados. Este procedimiento solamente se aplica cuando la solución obtenida por la heurística de asignación no respeta la restricción de overtime.

En la Figura 5.4 se muestra un posible intercambio que realiza este algoritmo sobre la solución de la Figura 5.2(e). En este caso, se toma $r = r_2$ y $r' = r_4$ y la asignación obtenida luego de intercambiar las rutas es $R_1 = \{r_1, r_2\}$ y $R_2 = \{r_3, r_4, r_5\}$. La nueva solución es factible, puesto que se tiene $O_1(s^{\text{new}}) = O_2(s^{\text{new}}) = 0$.

Paso 1 (inicialización).
Inicializar la memoria M.
Guardar la mejor solución encontrada al inicializar la memoria en s^{best}.
Hacer $k \leftarrow 0$.

Paso 2 (construcción de la solución).
Construir una solución s en base a la información en M.

Paso 3 (búsqueda local).
Aplicar la búsqueda local partiendo de s.
Sea s^* la solución obtenida.

Paso 4 (actualización de la mejor solución).
Si $s^* \prec s^{\text{best}}$, hacer $s^{\text{best}} \leftarrow s^*$, $s \leftarrow s^*$ e ir al paso 3.

Paso 5 (actualización de la memoria).
Actualizar M utilizando la solución s^*.

Paso 6 (terminación).
Hacer $k \leftarrow k + 1$.
Si $k = AMP^{\text{iter}}$ devolver s^{best} y terminar; si no, ir al paso 2.

Figura 5.5: El Adaptive Memory Procedure (AMP).

5.3. Descripción general del AMP

El algoritmo que proponemos en este trabajo para resolver el VRPMT responde al esquema presentado en la Figura 5.5. Se mantiene una memoria M con información acerca de las mejores soluciones visitadas por el algoritmo. Al comenzar la ejecución (paso 1) se inicializa la memoria M y la mejor solución encontrada s^{best}. Periódicamente se construye una nueva solución combinando rutas de la memoria (paso 2) y en el paso 3 se le aplica un algoritmo de búsqueda local. En el paso 4 se chequea si la solución obtenida por la búsqueda local es mejor que s^{best}. En caso afirmativo, se actualiza s^{best} y la búsqueda local se ejecuta nuevamente. La solución obtenida en la última ejecución de la búsqueda local se utiliza para actualizar la memoria (paso 5). El algoritmo termina al haber realizado AMP^{iter} fases de construcción y búsqueda local (AMP^{iter} es un parámetro del algoritmo).

5.3.1. Comparación de soluciones

Dada la complejidad de respetar la restricción de overtime, durante toda la ejecución del AMP se aceptan las soluciones que no satisfacen dicha restricción. Esto introduce un problema al intentar definir cuándo una solución es mejor que otra. El criterio usual de comparar los valores objetivo de las soluciones pierde sentido por sí solo, pues en general puede obtenerse una solución con un valor objetivo bajo cuando que se permite violar las restricciones del problema.

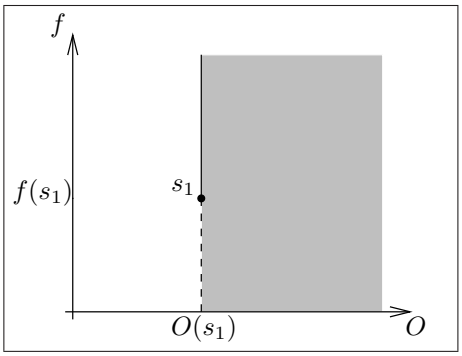


Figura 5.6: Visión gráfica del criterio de comparación de soluciones.

Para definir un orden entre las soluciones se recurre al criterio lexicográfico, definido de la siguiente manera:

$$(a_1, b_1) \prec (a_2, b_2) \iff a_1 < a_2 \vee a_1 = a_2 \wedge b_1 < b_2 \tag{5.1}$$

Dadas dos soluciones s_1 y s_2, consideramos que s_1 es mejor que s_2, y lo denotamos como $s_1 \prec s_2$, cuando se cumple $(O(s_1), f(s_1)) \prec (O(s_2), f(s_2))$, es decir, cuando se verifica alguna de las siguientes condiciones:

1. $O(s_1) < O(s_2)$,
2. $O(s_1) = O(s_2)$ y $f(s_1) < f(s_2)$.

El caso 1 implica que s_1 está mas cerca de la factibilidad que s_2. El caso 2, establece que ambas soluciones son equivalentes desde el punto de vista de la factibilidad, pero s_1 es mejor en cuanto al costo. En el caso particular de dos soluciones factibles, es decir cuando $O(s_1) = O(s_2) = 0$, el criterio obtenido es el habitual: s_1 es mejor que s_2 si $f(s_1) < f(s_2)$.

En la Figura 5.6 se ilustra este criterio de comparación de soluciones desde un punto de vista gráfico. Una solución s se representa en un par de ejes cartesianos por el punto $(O(s), f(s))$. La solución s_1 es mejor que las soluciones que caen dentro de la zona sombreada, excluyendo la línea punteada. La zona punteada corresponde al caso 1, mientras que la línea llena se corresponde con el caso 2. El resto de los puntos representa soluciones que son mejores que s_1 respecto a este criterio lexicográfico.

5.4. Manejo de la memoria

Al igual que en la propuesta de Rochat y Taillard [112], la memoria está formada por rutas que pertenecen a alguna de las "buenas" soluciones que van siendo visitadas por el algoritmo. La memoria se mantiene ordenada, de modo que las rutas de las mejores soluciones estén al comienzo. En este caso M es una secuencia de rutas que denotaremos $M = (r_1, \ldots, r_{|M|})$, siendo $|M|$ la cantidad de rutas almacenadas en la memoria. El hecho

Paso 1 (inicialización).
Seleccionar un cliente v_1 al azar.
Ordenar los clientes $j = 1, \ldots, n$ de forma creciente según el ángulo $\widehat{v_1 0 j}$.
Sea $(v_1, \ldots, v_n)$ la secuencia ordenada de clientes.
Hacer $R = \emptyset$ y $k \leftarrow 1$.

Paso 2 (nueva ruta).
Crear una nueva ruta $r \leftarrow (0, v_k, 0)$.
Inicializar la demanda $d \leftarrow d(v_k)$.

Paso 3 (ampliación de la ruta).
Hacer $k \leftarrow k + 1$.
Si $k = n + 1$, hacer $R \leftarrow R \cup \{r\}$ e ir al paso 4.
Si $d + v_k > Q$, hacer $R \leftarrow R \cup \{r\}$ e ir al paso 2.
Si no, agregar v_k luego de v_{k-1} en r. Hacer $d \leftarrow d + v_k$. Repetir el paso 3.

Paso 4 (asignación).
Aplicar la heurística de asignación a R, para obtener $s = (R_1, \ldots, R_m)$.

Paso 5 (búsqueda local).
Aplicar la búsqueda local sobre s.

Figura 5.7: Construcción de una solución inicial.

de que la ruta r esté almacenada en la memoria se denotará como $r \in M$. El valor de $|M|$ está acotado superiormente por M^{size}, que es un parámetro de entrada del algoritmo.

Como se mencionó anteriormente, dada la complejidad computacional del problema de encontrar una solución factible para el VRPMT, las rutas almacenadas en la memoria pueden eventualmente pertenecer a una solución que no respete la restricción de overtime. Asimismo, las soluciones construídas a partir de rutas de la memoria también pueden ser no factibles respecto a esa restricción. Sin embargo, la restricción de capacidad de los vehículos será respetada por todas las rutas de la memoria.

5.4.1. Inicialización

Para comenzar la ejecución del algoritmo debe poblarse la memoria con rutas. Para ello, se generan I soluciones y cada una de ellas se agrega a M. Para crear cada una de las soluciones iniciales se aplica el algoritmo que se muestra en la Figura 5.7.

En los pasos 1-3, se construyen rutas utilizando la *Heurística de Barrido* [64] (ver Sección 2.5.1). Se parte de un cliente elegido aleatoriamente y se agregan clientes a una ruta hasta colmar la capacidad del vehículo. Cuando la ruta está completa se comienza otra con un nuevo cliente y se sigue el mismo procedimiento. Por la forma en que son construidas, las rutas del conjunto R respetan la restricción de capacidad y visitan a todos los clientes. El orden en que se consideran los clientes es determinado por el ángulo formado por $(v_1, 0, j)$.

Paso 1 (inserción de las rutas).
Para cada $r \in R(s)$, etiquetar r con $(O(s), f(s))$ e insertar r en M.

Paso 2 (chequeo de tamaño).
Si $|M| > M^{\text{size}}$, eliminar las últimas $|M| - M^{\text{size}}$ rutas de M.

Figura 5.8: Actualización de la memoria.

Luego de obtenidas las rutas, debe calcularse una asignación de éstas a los vehículos, para lo cual se utiliza el algoritmo de asignación presentado en la Sección 5.2. Esto da una solución para el VRPMT(posiblemente no factible respecto a las restricciones de overtime), sobre la cual se aplica el algoritmo de búsqueda local presentado más adelante en la Sección 5.5.

Se realizan I ejecuciones de este algoritmo, utilizando clientes iniciales elegidos aleatoriamente en el paso 1. Cada una de las soluciones obtenidas se agrega a la memoria. La cantidad de soluciones iniciales (I) es un parámetro del algoritmo.

5.4.2. Actualización

Al agregar una solución s a la memoria, cada ruta $r \in R(s)$ se etiqueta con el par $(O(s), f(s))$ y se inserta en la secuencia ordenada M, utilizando el criterio de ordenación lexicográfico (ver Sección 5.3.1). Luego de agregar todas las rutas de la solución, si el tamaño de la memoria supera el valor máximo permitido, se eliminan las rutas de las últimas posiciones hasta que $|M| = M^{\text{size}}$. El algoritmo utilizado se da en la Figura 5.8.

5.4.3. Construcción de una solución

Para construir una solución en base a la información almacenada en la memoria, se seleccionan rutas de M en forma no determinística, privilegiando a las que ocupan las primeras posiciones, pues forman parte de las mejores soluciones encontradas hasta el momento. Un seudocódigo de este proceso se muestra en la Figura 5.9.

En los pasos 1-3, se seleccionan rutas de la memoria de modo que compartan clientes. Las rutas que comparten clientes con las seleccionadas, se eliminan provisoriamente de la memoria. Podría ocurrir que en cierta iteración la memoria quede vacía y aún haya clientes no visitados. Eso se chequea en el paso 4 y, en caso de que efectivamente ocurra, se crean rutas para ellos utilizando el algoritmo de barrido (los pasos 1-3 del algoritmo de la Figura 5.7). Una vez que se dispone de un conjunto de rutas R que visita a cada cliente exactamente una vez, se asignan dichas rutas a los vehículos mediante la heurística presentada en la Sección 5.2. Con esto se obtiene una solución para el VRPMT que visita exactamente una vez a cada cliente y verifica la restricción de capacidad, pero no necesariamente verifica la restricción de overtime.

Paso 1 (inicialización).
Hacer $M' \leftarrow M$ y $R = \emptyset$.

Paso 2 (selección de la próxima ruta).
Seleccionar $r \in M'$ de manera no determinística.

Paso 3 (actualización).
Agregar r a la solución: $R \leftarrow R \cup \{r\}$.
Eliminar de M' las rutas que comparten clientes con r.

Paso 4 (fin de la selección).
Si $M' \neq \emptyset$, ir a 2.
Si hay clientes sin visitar, crear rutas para ellos y agregarlas a R.

Paso 5 (asignación).
Aplicar el algoritmo de asignación sobre R y obtener $s = (R_1, \ldots, R_m)$.

Figura 5.9: Construcción de una solución en base a la memoria.

La distribución de probabilidades utilizada al elegir una ruta de M' en el paso 2 no depende de los valores de las etiquetas, sino de su orden relativo. Siendo $M' = (r_1, \ldots, r_{|M'|})$, la probabilidad de elegir la ruta r_i está dada por

$$p_i = 2\frac{|M'| + 1 - i}{|M'|(|M'| + 1)} \tag{5.2}$$

Con esta distribución se cumple que

$$p_i - p_{i+1} = \frac{2}{|M'|(|M'| + 1)} \; \forall \, i = 1, \ldots, |M'| - 1.$$

Es decir, la selección de rutas está sesgada hacia las que ocupan las primeras posiciones, pero el decrecimiento de las probabilidades se da de forma lineal, como se muestra en la Figura 5.10.

5.5. Búsqueda local

El algoritmo diseñado para realizar la búsqueda local está basado en la metaheurística Tabu Search. Las ideas principales fueron tomadas de dos algoritmos propuestos para resolver el VRP: *Taburoute* [60] y *Granular Tabu Search* [127]. Para la búsqueda local se supone que el grafo G es completo (en caso de no serlo, puede adaptarse fácilmente introduciendo algunos chequeos).

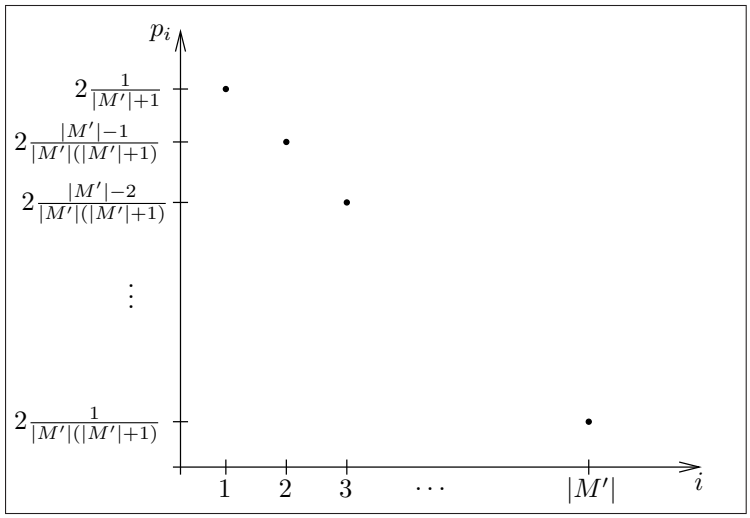


Figura 5.10: Distribución de probabilidades de selección de las rutas.

5.5.1. Función objetivo penalizada

Durante la ejecución del algoritmo se permite visitar soluciones que violen tanto las restricciones de capacidad como las de horizonte de tiempo. Dentro del algoritmo, el costo de una solución s, está dado por

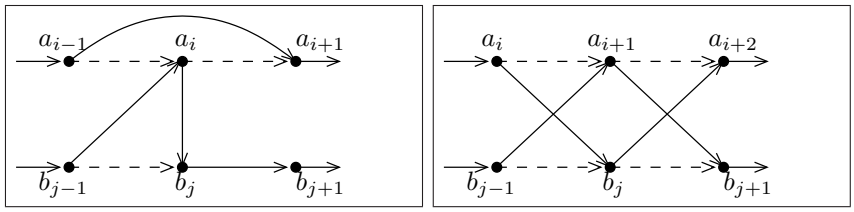
$$f_P(s) = f(s) + \alpha D(s) + \beta O(s) \tag{5.3}$$

donde $f(s)$ es el costo de las rutas definido en (4.1), $D(s)$ mide la violación de las restricciones de capacidad (definida en (4.16)), $O(s)$ es el overtime de la solución (4.19) y α y β son valores que regulan la importancia relativa del costo y cada una de las restricciones del problema.

Los valores de α y β varían durante la ejecución algoritmo. Si en una iteración la solución viola la restricción de capacidad (es decir, $D(s) > 0$), se incrementa el valor de α buscando que en la próxima iteración se le de mayor importancia a esta restricción. Del mismo modo, si $O(s) > 0$ se aumenta el valor del parámetro β. Cuando la solución visitada satisface alguna restricción, el parámetro correspondiente disminuye su valor para dar mayor importancia al resto de los términos. Los incrementos de los parámetros se realizan duplicando su valor, y los decrementos se efectúan dividiendo el valor del parámetro entre dos. Para evitar problemas numéricos, se fijan cotas superiores e inferiores a los parámetros, de modo que en cada iteración se cumpla que $\alpha \in [\alpha^{\min}, \alpha^{\max}]$ y $\beta \in [\beta^{\min}, \beta^{\max}]$.

5.5.2. Estructuras de vecindad

La vecindad de una solución se define mediante movidas. Como se mencionó en la Sección 4.4, en una solución hay dos componentes principales que son las rutas y la asignación de las rutas a los vehículos. Para contemplar esta característica se definieron dos tipos de



(a) Customer Insert, CI_{a_i,b_j} (b) Customer Swap, CS_{a_i,b_j}

Figura 5.11: Los dos tipos de movidas que definen la vecindad de ruteo.

vecindades. La *vecindad de ruteo* introduce modificaciones sobre la topología de las rutas, sin modificar la asignación. La *vecindad de asignación* varía la asignación pero no modifica la secuencia de visitas de las rutas.

Vecindad de ruteo

La vecindad de ruteo se define mediante las movidas *customer insert* (CI) y *customer swap* (CS), que se ilustran en la Figura 5.11. Ambos tipos de movidas involucran a dos rutas diferentes r_1 y r_2 y quedan unívocamente determinadas por dichas rutas y el par de nodos (a_i, b_j).

En las movidas CI, se elimina al nodo a_i de r_1 y se lo agrega a r_2 inmediatamente antes del nodo b_j. Dadas r_1 y r_2, denotaremos como $CI_{a_i,b_j}(s)$ a la solución que se obtiene al aplicar la movida CI determinada por (a_i, b_j) sobre la solución s. La nueva solución se obtiene realizando las siguientes modificaciones sobre s:

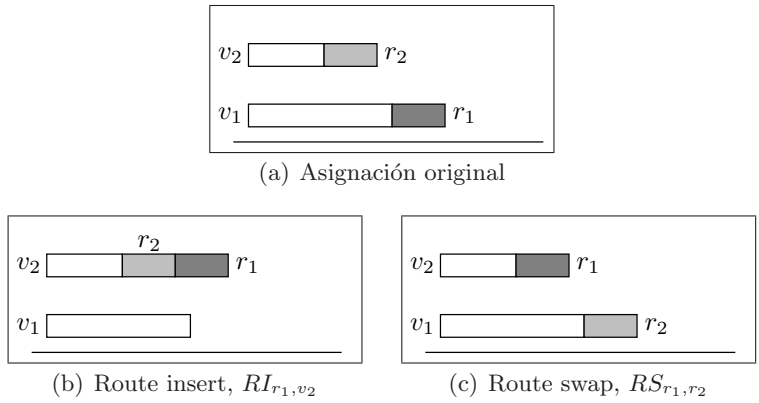
$$r_1 \leftarrow (0, a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{n(r_1)}, 0)$$
$$r_2 \leftarrow (0, b_1, \ldots, b_{j-1}, a_i, b_j, b_{j+1}, \ldots, b_{n(r_2)}, 0).$$

Para que el resultado sea una solución, debe cumplirse que $a_i \neq 0$, pues de lo contrario r_1 no visitaría al depósito en la nueva solución. En el caso de que la ruta r_1 tenga un solo cliente, es decir, $r_1 = (0, a_1, 0)$, luego de aplicar la movida quedará vacía y debe ser eliminada de la solución.

En las movidas CS, los nodos a_{i+1} y b_j intercambian sus posiciones en sus respectivas rutas. Se denotará por $CS_{a_i,b_j}(s)$ a la solución obtenida luego de aplicar la movida CS determinada por (a_i, b_j) sobre la solución s. En la nueva solución se realizan los siguientes reemplazos:

$$r_1 \leftarrow (0, a_1, \ldots, a_i, b_j, a_{i+2}, \ldots, a_{n(r_1)}, 0)$$
$$r_2 \leftarrow (0, b_1, \ldots, b_{j-1}, a_{i+1}, b_{j+1}, \ldots, \ldots, b_{n(r_2)}, 0).$$

En este caso, debe cumplirse que $a_{i+1} \neq 0$ y $b_j \neq 0$. Como las movidas CS_{a_i,b_j} y $CS_{b_{j-1},a_{i+1}}$ dan el mismo resultado, se exigirá que $a_i < b_{j-1}$ para no examinar movidas duplicadas.



(a) Asignación original

(b) Route insert, RI_{r_1,v_2}

(c) Route swap, RS_{r_1,r_2}

Figura 5.12: Los dos tipos de movidas que definen la vecindad de asignación.

Dada una solución s, denotaremos por $N_R(s)$ a todas las soluciones que pueden obtenerse aplicando alguna de las movidas de ruteo sobre s.

Vecindad de asignación

La vecindad de asignación se define utilizando ideas similares, mediante las movidas *route insert* (RI) y *route swap* (RS). Estas movidas toman dos vehículos diferentes v_1 y v_2 y modifican sus asignaciones como se muestran en la Figura 5.12.

En las movidas RI se elimina una ruta r_1 del conjunto R_1 de rutas asignadas al vehículo v_1 y se la asigna al vehículo v_2. Se utilizará $RI_{r_1,v_2}(s)$ para denotar a la solución obtenida al aplicar esta movida sobre s. Dicha solución se obtiene realizando los siguientes reemplazos:

$$R_1 \leftarrow R_1 \setminus \{r_1\}$$
$$R_2 \leftarrow R_2 \cup \{r_1\}.$$

En las movidas RS, además de lo anterior, se elimina una ruta r_2 de las asignadas vehículo v_2 y se la asigna a v_1. La solución obtenida se denotará como $RS_{r_1,r_2}(s)$ y consiste en realizar:

$$R_1 \leftarrow R_1 \setminus \{r_1\} \cup \{r_2\}$$
$$R_2 \leftarrow R_2 \setminus \{r_2\} \cup \{r_1\}.$$

Las movidas RS_{r_1,r_2} y RS_{r_2,r_1} son la misma. Por lo tanto, de manera similar a lo hecho para las movidas CS, se exigirá en este caso que $v_1 < v_2$.

Dada una solución s, denotaremos por $N_A(s)$ a todas las soluciones que pueden obtenerse aplicando alguna de las movidas de asignación sobre s.

5.5.3. Soluciones tabú

Cuando un cliente es eliminado de una ruta mediante una movida de ruteo, todas las movidas que vuelven a insertarlo en esa ruta se declaran tabú por θ iteraciones. De manera similar, cuando una ruta es eliminada de un vehículo, todas las movidas que vuelven a asignarla a dicho vehículo son movidas tabú por θ iteraciones. En ambos casos, el valor de θ se sortea con probabilidad uniforme en el intervalo $[\theta^{\min}, \theta^{\max}]$. Se utiliza el criterio de admisibilidad usual, es decir, se aceptan las movidas tabú cuando mejoran a la mejor solución encontrada hasta el momento.

Dada una solución s, denotaremos como $T(s)$ a las soluciones que se obtienen aplicando movidas tabú sobre s (salvo que se mejore a la mejor solución encontrada hasta el momento). Al explorar la vecindad de s, se evita seleccionar soluciones de $T(s)$.

5.5.4. Estrategia de exploración de las vecindades

En cada iteración del algoritmo de búsqueda tabú debe determinarse la solución de la vecindad (excluyendo a las soluciones tabú) que minimiza el valor de la función objetivo. Dado que la función objetivo del algoritmo varía a lo largo de la ejecución, los cálculos hechos en una iteración pierden utilidad en las iteraciones siguientes. Por lo tanto, en cada iteración se debe recorrer toda la vecindad evaluando f_P en cada una de las soluciones vecinas. Esto hace que la mayor parte del tiempo de ejecución del AMP se invierta en evaluar la vecindad de la solución actual.

Dado que en general habrá más clientes que vehículos ($n > m$), es de esperar que la vecindad de ruteo tenga más elementos que la de asignación ($|N_R(s)| > |N_A(s)|$). Resulta de interés, entonces, contar con un mecanismo para reducir el tamaño de $N_R(s)$. Dicha reducción debería eliminar de la vecindad a las soluciones que, sin ser evaluadas, puede suponerse que no serán de buena calidad.

Dada una solución, tanto las movidas CI como las CS quedan determinadas por un par ordenado de nodos $(a_i, b_j) \in E$ y las rutas correspondientes. Dicho par se corresponde con uno de los arcos agregados a la solución. En lugar de examinar las movidas determinadas por todos los posibles arcos de E, la estrategia implementada para reducir la cantidad de soluciones en $N_R(s)$ consiste en considerar únicamente las movidas generadas por un subconjunto de arcos $E' \subset E$. Es decir, solamente se evalúan las movidas de ruteo que quedan determinadas por arcos de dicho conjunto. La estrategia no implica que los arcos que no pertenecen a E' nunca se introduzcan en la solución, sino que las movidas analizadas deben agregar al menos un arco de E'. Llamaremos $N_R^*(s)$ a la vecindad de ruteo reducida.

Esta estrategia fue propuesta por Toth y Vigo [127] en el contexto del VRP. Como lo indican sus autores, constituye una manera poco invasiva de reducir el tiempo de ejecución de una heurística de búsqueda local.

Cuánto menos arcos se consideren en el conjunto E', más se intensificará la búsqueda y, a su vez, menos tiempo consumirá la exploración de la vecindad. Por un lado, el conjunto debe ser lo suficientemente reducido como para lograr una disminución en el tamaño de las vecindades. Pero por otro lado, debe procurarse que haya una cantidad razonable de alternativas para que el algoritmo encuentre soluciones de buena calidad. Una elección adecuada implica un balance entre esos dos aspectos.

El conjunto de arcos utilizados para generar la vecindad de ruteo reducida se define como $E' = S \cup I$, donde S es un conjunto de arcos "cortos" (es decir, de bajo costo) y el conjunto I contiene arcos "importantes". Con los arcos cortos se pretende obtener movidas que den soluciones de bajo costo, y con los arcos importantes se busca encontrar movidas que si bien pueden no resultar atractivas desde el punto de vista de los costos, permiten aproximarse a la región factible.

El conjunto de un arcos importantes está formado por los arcos que tienen al depósito en uno de sus extremos y los que participan en alguna solución de la memoria:

$$I = \{(i,j) \in E : i = 0 \vee j = 0 \vee \exists\, r \in M : r = (0,\ldots,i,j,\ldots,0)\} \tag{5.4}$$

Consideramos dos estrategias para definir el conjunto de arcos cortos S: *reducción estática de la vecindad* y *reducción dinámica de la vecindad*, que dan lugar a dos algoritmos de búsqueda local diferentes.

Reducción estática de la vecindad

Para cada nodo $i \in V$ se define como $\delta_p(i)$ al conjunto de los p clientes más cercanos a i considerando los costos de los arcos de la forma (i,j). En esta estrategia, el conjunto E' queda definido de la siguiente manera:

$$S = \{(i,j) \in E : j \in \delta_p(i)\}\,. \tag{5.5}$$

Es decir, para cada cliente solo se consideran los p clientes más cercanos. El valor de p es un parámetro del algoritmo. Cuanto menor sea su valor, menor será también la cantidad de movidas analizadas. En este caso el conjunto S no varía durante la ejecución del algoritmo, lo que da el carácter de estático a la estrategia.

Una ventaja de este método es que mediante la fijación del parámetro p se puede controlar el tamaño del conjunto S, que es $p(n-1)$. Sin embargo, utilizar un valor fijo para establecer el límite podría resultar demasiado rígido en algunos casos.

Reducción dinámica de la vecindad

En este caso, el conjunto de arcos se define como:

$$S = \{(i,j) \in E : c_{ij} \leq \vartheta\}. \tag{5.6}$$

donde ϑ actúa como un umbral para los costos. Solamente se consideran los arcos cuyo costo no supera a ϑ. El valor de dicho umbral se define como

$$\vartheta = \gamma \frac{f(s^{\text{best}})}{n + |R(s^{\text{best}})|} \tag{5.7}$$

donde s^{best} es la mejor solución encontrada hasta el momento por la búsqueda local, n es la cantidad de clientes del problema y γ es un parámetro.

El valor de ϑ/γ representa el costo promedio de un arco en la solución (en términos de la función objetivo del problema). El parámetro γ (llamado *parámetro de diversificación*) regula el tamaño del conjunto S. Dado que la solución s^{best} varía durante la ejecución del algoritmo, también lo hace el valor de ϑ y el conjunto de arcos S. Esta característica hace que esta estrategia sea dinámica.

Con esta estrategia, a diferencia de la anterior, no se puede predecir el tamaño del conjunto E', aunque fijando valores bajos para γ puede esperarse tener cierto control. La ventaja de este método es que el umbral se fija en función de las características del problema y de las soluciones que van siendo encontradas por el algoritmo.

5.5.5. Evaluación de las movidas

Al explorar la vecindad de una solución s, es necesario evaluar la función objetivo f_P sobre cada solución $s^{\text{new}} \in N_R^*(s) \cup N_A(s)$. Al aplicar una movida sobre una solución, se introducen cambios en los tres términos que afectan a f_P. Dichos cambios pueden expresarse como

$$f(s^{\text{new}}) = f(s) + \Delta f \tag{5.8}$$

$$D(s^{\text{new}}) = D(s) + \Delta D \tag{5.9}$$

$$O(s^{\text{new}}) = O(s) + \Delta O \tag{5.10}$$

donde Δf, ΔD y ΔO encapsulan las modificaciones realizadas por la movida a cada uno de los términos. El cálculo de $f_P(s^{\text{new}})$ se reduce a determinar dichas variaciones. Dado que las modificaciones realizadas sobre la estructura de la solución son relativamente pequeñas, estos valores pueden determinarse de manera eficiente.

Para el análisis siguiente se consideran las rutas $r_1 = (0, \ldots, a_i, \ldots, 0)$ asignada al vehículo v_1 y $r_2 = (0, \ldots, b_j, \ldots, 0)$ asignada al vehículo v_2.

Movidas de ruteo

Consideremos el caso de las movidas de ruteo determinadas por (a_i, b_j). Tenemos

$$\Delta f = \Delta c(r_1) + \Delta c(r_2) \tag{5.11}$$
$$\Delta D = -D(r_1) - D(r_2) + (d(r_1) + \Delta d(r_1))^+ + (d(r_2) + \Delta d(r_2))^+ \tag{5.12}$$

En el caso que las dos rutas estén asignadas al mismo vehículo, es decir, cuando $v_1 = v_2$, tenemos

$$\Delta O = -O_{v_1}(s) + (t(v_1) + \Delta t(r_1) + \Delta t(r_2) - T)^+ \tag{5.13}$$

y cuando $v_1 \neq v_2$,

$$\Delta O = -O_{v_1}(s) - O_{v_2}(s) + (t(v_1) + \Delta t(r_1) - T)^+ + (t(v_2) + \Delta t(r_2) - T)^+ \tag{5.14}$$

Los valores de $\Delta c(r_i)$, $\Delta t(r_i)$ y $\Delta d(r_i)$ para $i = 1, 2$, reflejan los cambios en el costo, la duración y la demanda de cada una de las rutas implicadas en la movida. En el caso de las movidas CI_{a_i,b_j}, dichos valores se calculan de la siguiente manera:

$$\begin{aligned}
\Delta c(r_1) &= -c(a_{i-1}, a_i) - c(a_i, a_{i+1}) + c(a_{i-1}, a_{i+1}) \\
\Delta c(r_2) &= -c(b_{j-1}, b_j) + c(b_{j-1}, a_i) + c(a_i, b_j) \\
\Delta d(r_1) &= -d(a_i) \\
\Delta d(r_2) &= d(a_i)
\end{aligned}$$

mientras que para las movidas CS_{a_i,b_j} se tiene que:

$$\begin{aligned}
\Delta c(r_1) &= -c(a_i, a_{i+1}) - c(a_{i+1}, a_{i+2}) + c(a_i, b_j) + c(b_j, a_{i+2}) \\
\Delta c(r_2) &= -c(b_{j-1}, b_j) - c(b_j, b_{j+1}) + c(b_{j-1}, a_{i+1}) + c(a_{i+1}, b_{j+1}) \\
\Delta d(r_1) &= d(b_j) - d(a_{i+1}) \\
\Delta d(r_2) &= d(a_{i+1}) - d(b_j)
\end{aligned}$$

En ambos casos $\Delta t(r)$ se determina utilizando $\Delta t(r) = \lambda \Delta c(r)$.

Movidas de asignación

Las movidas de asignación son más sencillas de evaluar, puesto que no modifican ni el costo, ni la duración, ni la demanda de las ruta. Es decir, $\Delta f = 0$ y $\Delta D = 0$. Simplemente debe determinarse el valor de ΔO.

En el caso de las movidas RI_{r_1,v_2}, tenemos

$$\Delta O = -O_{v_1}(s) - O_{v_2}(s) + (t(v_1) - t(r_1) - T)^+ + (t(v_1) + t(r_1) - T)^+ \tag{5.15}$$

y para las movidas RS_{r_1,r_2} se cumple que

$$\Delta O = -O_{v_1}(s) - O_{v_2}(s) + (t(v_1) - t(r_1) + t(r_2) - T)^+ + (t(v_1) - t(r_2) + t(r_1) - T)^+ \tag{5.16}$$

5.5.6. Resumen de la búsqueda local

En la Figura 5.13 se da un seudocódigo del algoritmo de búsqueda tabú propuesto. El algoritmo recibe una solución inicial s_0, factible respecto a la restricción de capacidad y no necesariamente factible respecto a la restricción de overtime. En el paso 1 se inicializan las variables y estructuras utilizadas en el algoritmo. En el paso 2 se exploran las vecindades. Considerando la solución actual s_k y la función objetivo penalizada, se selecciona la mejor solución de la vecindad de ruteo s_R y la mejor de la vecindad de asignación s_A. La exploración de la vecindad se realiza según las estrategias dadas en la Sección 5.5.5. Luego de exploradas las vecindades, se selecciona la mejor de ambas soluciones. En caso de que la solución mejore a la mejor solución obtenida hasta el momento, en el paso 3 se busca mejorar aún más la solución mediante el algoritmo US (ver Sección 2.5.1) y la heurística de asignación, y luego se actualiza la solución s^{best}. En el paso 4 se actualiza el conjunto de soluciones tabú (lo cual depende del tipo de movida realizada en el paso 2), el conjunto de arcos utilizados para generar las movidas (solamente en la estrategia de reducción dinámica) y los factores de penalización utilizados en la función objetivo. El algoritmo termina luego de ejecutar TS^{iter} iteraciones, lo cual se chequea en el paso 5.

5.6. Diferencias con la propuesta de Rochat y Taillard

Si bien la estructura del AMP propuesto en este trabajo comparte las ideas centrales utilizadas en el algoritmo de Rochat y Taillard [112], existen dos diferencias no menores entre ambas propuestas:

1. en los algoritmos propuestos en este trabajo se acepta la incorporación de soluciones no factibles en la memoria, lo cual dota de mayor flexibilidad al método,

2. cada vez que la búsqueda local devuelve una nueva mejor solución se realiza otra ejecución de la misma, partiendo de la solución encontrada (este mecanismo de intensificación no se utiliza en la propuesta original).

Paso 1 (inicialización).
Hacer $s^{\text{best}} \leftarrow s_0$.
Inicializar el conjunto de movidas tabú y el conjunto de arcos E'.
Hacer $\alpha \leftarrow \alpha^{\min}$ y $\beta \leftarrow \beta^{\min}$. Hacer $k \leftarrow 1$.

Paso 2 (exploración de las vecindades).
Hacer $s_R \leftarrow \arg\min_{s \in N_R^*(s_{k-1}) \setminus T(s_{k-1})} f_P(s)$.
Hacer $s_A \leftarrow \arg\min_{s \in N_A(s_{k-1}) \setminus T(s_{k-1})} f_P(s)$.
Si $f_P(s_R) \leq f_P(s_A)$, hacer $s_k \leftarrow s_R$; si no, hacer $s_k \leftarrow s_A$.

Paso 3 (actualización de la solución).
Si $D(s_k) = 0$ y $s_k \prec s^{\text{best}}$, aplicar el algoritmo US y el algoritmo de asignación sobre s_k y hacer $s^{\text{best}} \leftarrow s_k$.

Paso 4 (actualización de los parámetros).
Actualizar el conjunto de movidas tabú.
Actualizar el conjunto de arcos E' (si corresponde).
Si $D(s_k) = 0$, $\alpha \leftarrow \max\{\alpha/2, \alpha^{\min}\}$; si no $\alpha \leftarrow \min\{2\alpha, \alpha^{\max}\}$.
Si $O(s_k) = 0$, $\beta \leftarrow \max\{\beta/2, \beta^{\min}\}$; si no, $\beta \leftarrow \min\{2\beta, \beta^{\max}\}$.

Paso 5 (terminación).
Si $k = TS^{\text{iter}}$, devolver s^{best} y terminar.
Si no, hacer $k \leftarrow k + 1$ e ir al paso 2.

Figura 5.13: Algoritmo de búsqueda tabú.

5.7. Parámetros de los algoritmos

El comportamiento de los algoritmos propuestos es influenciado por los valores que se asigne a los siguientes parámetros:

- la cantidad de iteraciones globales AMP^{iter},
- la cantidad de iteraciones de búsqueda local TS^{iter},
- la cantidad de soluciones utilizadas para inicializar la memoria: I,
- la cantidad máxima de rutas en la memoria: M^{size},
- el intervalo en que se elige la cantidad de iteraciones tabú: $[\theta^{\min}, \theta^{\max}]$,
- los intervalos para las penalizaciones: $[\alpha^{\min}, \alpha^{\max}]$ y $[\beta^{\min}, \beta^{\max}]$,
- la cantidad de vecinos para la estrategia de reducción estática: p,
- el parámetro de diversificación para la estrategia de reducción dinámica: γ.

Capítulo 6

Resultados experimentales

6.1. Introducción

Existen diferentes alternativas para determinar qué tan bueno es determinado algoritmo para resolver un problema de optimización. En algunos casos es posible demostrar que el algoritmo encuentra una solución óptima o, al menos, dar una cota para la diferencia entre la solución que se obtendrá y la solución óptima. La posibilidad de obtener resultados analíticos suele exigir un gran conocimiento de la estructura del problema y de las propiedades del algoritmo en cuestión. Desafortunadamente, con las técnicas y el conocimiento actuales resulta extremadamente difícil realizar un estudio analítico del comportamiento de la gran mayoría de las metaheurísticas. Esta limitación estimula el estudio de los algoritmos de manera experimental. A grandes rasgos, la experimentación consiste en la ejecución del algoritmo sobre un conjunto de instancias de prueba y el análisis de los resultados obtenidos. Este tipo de análisis es uno de los aspectos cuestionables en el uso de este tipo de técnicas [80, 81].

En este capítulo se reportan los experimentos realizados para medir qué tan buenos son los algoritmos propuestos en el Capítulo 5 para resolver el VRPMT. El análisis se dividió en dos fases: *ajuste de parámetros* y *pruebas de desempeño*. En el ajuste de parámetros se busca determinar valores para los parámetros que influyen en el comportamiento de cada algoritmo. Dichos valores son utilizados en las pruebas de desempeño para medir la calidad de los algoritmos y comparar los resultados con los obtenidos por otros autores.

Llamaremos AMPD a la versión del AMP que utiliza la estrategia de reducción dinámica de la vecindad y AMPE a la que utiliza la estrategia de reducción estática de la vecindad (ver Sección 5.5.4). Si bien son algoritmos muy similares y tienen una base común, se analizarán de forma separada.

Los algoritmos fueron implementados en C++ y ejecutados por un procesador AMD Athlon XP 2200+ de 1.8 GHz con 480 MBytes de RAM bajo Windows XP. Todos los tiempos de ejecución se reportan en segundos.

6.2. Instancias de prueba

Se utilizó el conjunto de 104 instancias de prueba para el VRPMT propuesto por Taillard et al. [120]. Estos casos de prueba fueron construídos utilizando los mismos grafos, demandas y capacidades que las instancias CMT-1, CMT-2, CMT-3, CMT-4, CMT-5, CMT-11 y CMT-12 propuestas por Christofides et al. [28] y las instancias F-11 y F-12 propuestas por Fisher [49] para el VRP. Nos referiremos a éstas como "instancias base". Los datos de las instancias base pueden consultarse en el Apéndice C.

Para cada instancia base se generan varias instancias del VRPMT utilizando diferentes valores de m y T. Dada una instancia del VRP y un valor para m, se utilizan dos valores para el horizonte de tiempo

$$T_1 = \left[1.05\frac{z_{\text{VRP}}^{\text{best}}}{m}\right] \text{ y } T_2 = \left[1.1\frac{z_{\text{VRP}}^{\text{best}}}{m}\right] \tag{6.1}$$

donde $z_{\text{VRP}}^{\text{best}}$ es el valor de la mejor solución conocida para la instancia base y $[x]$ denota el entero más próximo a x[1]. En todos los casos se utilizan distancias euclídeas y $\lambda = 1$.

Cada instancia de prueba será denotada como "Base $|\, m \,|\, T_i$", donde Base indica la instancia del VRP utilizada (que será alguna de CMT-1, CMT-2, CMT-3, CMT-4, CMT-5, CMT-11, CMT-12, F-11 y F-12), m representa la cantidad de vehículos disponibles y el horizonte de tiempo es el determinado por T_i (T_1 o T_2). Por ejemplo, CMT-1 $|\, 5 \,|\, T_1$ se refiere al caso de prueba que utiliza a CMT-1 como instancia base, con 5 vehículos y horizonte de tiempo T_1.

6.2.1. Dificultad de las instancias de prueba

Consideremos una instancia de prueba que utiliza el horizonte T_1. Si una solución es factible para esta instancia, las cotas presentadas en (4.13) y (4.14) aseguran que el valor de dicha solución se encuentra en el intervalo $[z_{\text{VRP}}^*, mT_1]$. A continuación se muestra la forma que toman estas cotas para las instancias de prueba utilizadas.

Dado que $[x] \leq x + \frac{1}{2}$, y por la definición del horizonte T_1 dada en (6.1), se cumple que

$$mT_1 \leq m\left(1.05\frac{z_{\text{VRP}}^{\text{best}}}{m} + \frac{1}{2}\right) = 1.05 z_{\text{VRP}}^{\text{best}} + \frac{m}{2}.$$

Además, como z_{VRP}^* es el valor óptimo de la instancia base y $z_{\text{VRP}}^{\text{best}}$ es el mejor valor conocido para dicha instancia, existe $\epsilon \geq 0$ tal que $z_{\text{VRP}}^{\text{best}} = z_{\text{VRP}}^* + \epsilon$. Por lo tanto, cualquier solución factible de una instancia que utiliza el horizonte T_1 tiene su valor en el intervalo

$$\left[z_{\text{VRP}}^*, 1.05(z_{\text{VRP}}^* + \epsilon) + \frac{m}{2}\right]. \tag{6.2}$$

[1] Dados $a \in \mathbb{Z}$ y $b \in \mathbb{R}_+$ con $b < 1$, $[a + b] = a$ si $b < \frac{1}{2}$ y $[a + b] = a + 1$ en otro caso.

Los valores de m utilizados para estas instancias varían entre 1 y 10 y los valores de $z^{\text{best}}_{\text{VRP}}$ varían entre 241.97 y 1291.44. Suponiendo que ϵ es un valor relativamente pequeño en comparación con z^*_{VRP}, puede deducirse que el uso de T_1 impone una cota superior para el valor de las soluciones factibles que es muy cercana al 5 % de la cota inferior z^*_{VRP}.

Mediante un razonamiento análogo puede probarse que los valores de las soluciones factibles para las instancias que utilizan T_2 están comprendidos en el intervalo

$$\left[z^*_{\text{VRP}}, 1.1(z^*_{\text{VRP}} + \epsilon) + \frac{m}{2}\right] \tag{6.3}$$

y que este horizonte de tiempo implica que las soluciones factibles deben tener un valor no mayor al 10 % de la cota inferior z^*_{VRP}.

Dicho de otro modo, para las instancias de prueba consideradas en este trabajo, encontrar una solución factible para el VRPMT y encontrar una solución cercana a la solución óptima del VRP son problemas de dificultad similar. Por otro lado, dado que no necesariamente puede construirse una solución factible para el VRPMT a partir de una solución factible para el VRP (ver Sección 4.8), es posible que las soluciones factibles del VRPMT tengan valores alejados de z^*_{VRP}.

En consecuencia, los horizontes de tiempo utilizados en estas instancias pueden considerarse como muy ajustados y hacen que encontrar soluciones factibles no resulte sencillo. Es posible que algunas instancias sean no factibles. Finalmente, debe notarse que las instancias que utilizan el horizonte T_1 son aún más ajustadas (y por lo tanto más difíciles) que las que utilizan el horizonte T_2.

Estas conclusiones aportan una perspectiva complementaria a la dada en la Sección 4.5 sobre la complejidad del problema, pues que un problema pertenezca a la clase NP-Hard no implica que no existan instancias que resulten sencillas de resolver en la práctica.

6.2.2. Antecedentes sobre las instancias de prueba

En la Tabla 6.1 se muestran las características de cada una de las instancias de prueba utilizadas. Para cada instancia se indica cuáles trabajos reportan soluciones factibles (✓) y cuales no (×), donde TL se refiere a Taillard et al. [120], BM a Brandão y Mercer [19] y PS a Petch y Salhi [104]. Para 12 de las instancias de prueba, ninguno de los autores reporta soluciones factibles. Respecto a estas instancias existen dos posibilidades: que no existan soluciones factibles o que existan pero simplemente no hayan sido encontradas. Con argumentos similares a los dados en la Sección 4.5, puede probarse que determinar si existe una solución factible para una instancia del VRPMT es un problema NP-Hard. Es de esperar que no haya un método simple para verificar *a priori* esa condición.

Instancia Base	m	T_1	TL	BM	PS	T_2	TL	BM	PS
CMT-1	1	551	✓	✓	✓	577	✓	✓	✓
$n = 50$	2	275	✓	✓	×	289	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 524.61$	3	184	×	×	×	192	×	✓	✓
	4	138	×	×	×	144	✓	✓	×
CMT-2	1	877	✓	✓	✓	919	✓	✓	✓
$n = 75$	2	439	✓	✓	✓	459	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 835.26$	3	292	✓	✓	✓	306	✓	✓	✓
	4	219	✓	✓	✓	230	✓	✓	✓
	5	175	✓	✓	✓	184	✓	✓	✓
	6	146	×	×	×	153	✓	✓	✓
	7	125	×	×	×	131	×	✓	×
CMT-3	1	867	✓	✓	✓	909	✓	✓	✓
$n = 100$	2	434	✓	✓	✓	454	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 826.14$	3	289	✓	✓	✓	303	✓	✓	✓
	4	217	✓	✓	✓	227	✓	✓	✓
	5	173	×	✓	×	182	✓	✓	✓
	6	145	×	×	×	151	✓	✓	✓
CMT-4	1	1080	✓	✓	✓	1131	✓	✓	✓
$n = 150$	2	540	✓	✓	✓	566	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 1028.42$	3	360	✓	✓	✓	377	✓	✓	✓
	4	270	✓	✓	×	283	✓	✓	✓
	5	216	✓	✓	✓	226	✓	✓	✓
	6	180	✓	✓	×	189	✓	✓	✓
	7	154	×	×	×	162	✓	✓	×
	8	135	×	×	×	141	×	✓	✓

Instancia Base	m	T_1	TL	BM	PS	T_2	TL	BM	PS
CMT-5	1	1356	✓	✓	✓	1421	✓	✓	✓
$n = 199$	2	678	✓	✓	✓	710	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 1291.44$	3	452	✓	✓	✓	474	✓	✓	✓
	4	339	✓	✓	✓	355	✓	✓	✓
	5	271	✓	✓	×	284	✓	✓	✓
	6	226	✓	✓	✓	237	✓	✓	✓
	7	194	✓	✓	×	203	✓	✓	✓
	8	170	✓	✓	×	178	✓	✓	✓
	9	151	✓	×	×	158	✓	✓	✓
	10	136	×	×	×	142	✓	✓	×
CMT-11	1	1094	✓	✓	✓	1146	✓	✓	✓
$n = 120$	2	547	✓	✓	×	573	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 1042.11$	3	365	✓	✓	×	382	✓	✓	✓
	4	274	×	×	×	287	✓	✓	×
	5	219	✓	✓	×	229	✓	✓	✓
CMT-12	1	861	✓	✓	✓	902	✓	✓	✓
$n = 100$	2	430	✓	✓	✓	451	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 819.56$	3	287	✓	✓	✓	301	✓	✓	✓
	4	215	✓	×	✓	225	✓	✓	✓
	5	172	×	×	✓	180	✓	✓	✓
	6	143	×	×	×	150	✓	✓	✓
F-11	1	254	✓	✓	✓	266	✓	✓	✓
$n = 71$	2	127	×	×	×	133	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 241.97$	3	85	×	×	×	89	×	✓	✓
F-12	1	1221	✓	✓	✓	1279	✓	✓	✓
$n = 134$	2	611	✓	✓	✓	640	✓	✓	✓
$z_{\text{VRP}}^{\text{best}} = 1162.96$	3	407	✓	✓	✓	426	✓	✓	✓

Tabla 6.1: Detalle de las instancias de prueba.

6.3. Ajuste de parámetros

El comportamiento de los algoritmos propuestos en este trabajo depende en gran medida de los valores que se asigne a los parámetros presentados en la Sección 5.7. Un paso previo a medir la calidad de estos algoritmos es determinar un único conjunto de valores para dichos parámetros. En las pruebas de desempeño de la Sección 6.4, los algoritmos serán ejecutados exclusivamente con los valores seleccionados en esta etapa.

El problema de ajustar los parámetros de un algoritmo de este tipo presenta objetivos que usualmente son contrapuestos. Por un lado, es deseable que los valores seleccionados permitan obtener soluciones de buena calidad. Por otro lado, debería procurarse que el tiempo de ejecución del algoritmo sea relativamente moderado. El objetivo de esta etapa es encontrar valores para los parámetros que ofrezcan un compromiso adecuado entre estos dos objetivos. Se busca un conjunto de valores de los parámetros para el algoritmo AMPD y uno (eventualmente diferente) para el algoritmo AMPE.

Como usualmente ocurre al trabajar con metaheurísticas, el conocimiento que se tiene acerca de la influencia de los valores de los parámetros en el desempeño del algoritmo es pobre. Dada esta limitación, la selección de los valores para los parámetros se realizó de manera experimental.

Se realizaron ejecuciones de los algoritmos sobre un subconjunto reducido de instancias de prueba, con diferentes combinaciones de valores para los parámetros. Analizando los resultados de esas ejecuciones (la calidad de las soluciones obtenidas y el tiempo de ejecución) se seleccionó una configuración de parámetros para cada algoritmo.

6.3.1. Configuraciones consideradas

Para disminuir la cantidad de parámetros a ajustar, los valores de algunos parámetros (respecto a los que se presume que el algoritmo es menos sensible) se fijaron de antemano:

- la cantidad de soluciones iniciales: $I = 20$,
- los límites de la cantidad de iteraciones tabú: $\theta^{\min} = 5$ y $\theta^{\max} = 10$,
- los límites de los factores de penalización: $\alpha^{\min} = \beta^{\min} = 1$ y $\alpha^{\max} = \beta^{\max} = 100000$.

Los valores de I y de $\theta^{\min}$ y $\theta^{\max}$ se obtuvieron de trabajos anteriores [112, 60]. Los límites para α y β permiten a esos factores oscilar en un intervalo lo suficientemente grande.

Para cada uno de los restantes parámetros, se definió un conjunto de valores posibles:

- $M^{\text{size}} \in \{150, 250\}$,
- $TS^{\text{iter}} \in \{250, 500, 750\}$,
- $\gamma \in \{1.25, 1.5, 1.75\}$ (en el AMPD) y $p \in \{10, 20, 30\}$ (en el AMPE).

Config.	M^{size}	TS^{iter}	γ	p
1	150	250	1.25	10
2	150	250	1.50	20
3	150	250	1.75	30
4	150	500	1.25	10
5	150	500	1.50	20
6	150	500	1.75	30
7	150	750	1.25	10
8	150	750	1.50	20
9	150	750	1.75	30

Config.	M^{size}	TS^{iter}	γ	p
10	250	250	1.25	10
11	250	250	1.50	20
12	250	250	1.75	30
13	250	500	1.25	10
14	250	500	1.50	20
15	250	500	1.75	30
16	250	750	1.25	10
17	250	750	1.50	20
18	250	750	1.75	30

Tabla 6.2: Detalle de los parámetros en cada configuración.

Núm.	Instancia	n	TL	BM	PS
1	CMT-2 | 6 | T_1	75	×	×	×
2	CMT-3 | 6 | T_1	100	×	×	×
3	CMT-4 | 4 | T_1	150	✓	✓	×
4	CMT-11 | 4 | T_1	120	×	×	×
5	CMT-12 | 4 | T_1	100	✓	×	✓
6	F-11 | 2 | T_1	71	×	×	×
7	F-12 | 2 | T_1	134	✓	✓	✓

Tabla 6.3: Detalle de las instancias de prueba utilizadas en el ajuste de parámetros.

En todos los casos se utilizó $AMP^{\text{iter}} = 300$. Con estos valores se obtienen las 18 configuraciones que se muestran en la Tabla 6.2 (los valores del parámetro γ deben considerarse solamente para el algoritmo AMPD, mientras que el parámetro p debe considerarse al analizar el algoritmo AMPE). De estas 18 configuraciones se pretende seleccionar una para cada algoritmo.

6.3.2. Instancias de prueba utilizadas

Para la experimentación en esta fase se utilizó un conjunto de 7 instancias que se muestran en la Tabla 6.3. Se procuró seleccionar instancias de diferentes grados de dificultad (según los resultados reportados por otros autores), así como de diferentes tamaños.

Para cada configuración se realizaron 6 ejecuciones independientes (utilizando diferentes semillas para el generador de números aleatorios) sobre cada instancia. Por lo tanto, cada configuración se ejecutó un total de 42 veces.

6.3.3. Metodología empleada

La selección de una configuración se realizó en dos fases. En la fase I se busca descartar un sub-conjunto de las configuraciones y en la fase II se realiza la selección final.

El objetivo de la fase I es eliminar de futuras consideraciones a las configuraciones que no ofrezcan ventajas respecto a ninguno de los dos objetivos (calidad de las soluciones y

tiempo de ejecución). Para cada configuración se tiene en cuenta la cantidad de ejecuciones factibles[2] y el tiempo de ejecución promedio (considerando las 42 ejecuciones).

Dadas dos configuraciones c_1 y c_2, siendo f_1 y f_2 la cantidad de soluciones factibles encontradas por cada una y $\bar{t_1}$ y $\bar{t_2}$ los tiempos medios de ejecución, diremos que la configuración c_1 domina a la configuración c_2 cuando

$$f_1 > f_2 \text{ y } \bar{t_1} < \bar{t_2}. \tag{6.4}$$

Cuando se da este caso, consideramos que la configuración c_2 no ofrece ninguna ventaja respecto a c_1 y es descartada. Un conjunto de configuraciones no dominadas es aquel en el cual ninguna configuración del conjunto domina a ninguna otra y, por lo tanto, ninguna podría ser descartada de antemano utilizando este criterio. El concepto de dominancia es fundamental en el área de la *optimización multi-objetivo* [40]. El resultado de esta fase es un conjunto de configuraciones no dominadas respecto a los objetivos de maximizar la cantidad de soluciones factibles encontradas y minimizar el tiempo medio de ejecución.

En la fase II se realiza un estudio más completo de las configuraciones no dominadas con el objetivo de seleccionar una de ellas. Para cada una, se mide la cantidad de casos en que no se pudo obtener una solución factible, el valor medio de la medida LTR sobre estos casos y su desviación estándar, buscando la configuración que ofrezca la mejor combinación de dichos valores. En base a esas medidas y las tomadas en la fase I, se decide cuál es la configuración que otorga el mejor compromiso entre todos los objetivos. A diferencia de la fase anterior, no establecemos un criterio sistemático para esta fase.

6.3.4. Ajuste de parámetros para AMPD

Todas las ejecuciones del algoritmo AMPD obtuvieron soluciones factibles para las instancias 3, 5 y 7. Estas instancias solamente serán consideradas al analizar los tiempos de ejecución, pues no revelan ninguna otra diferencia entre las configuraciones (en cuanto a los demás valores definidos como relevantes en el ajuste de parámetros).

Fase I - Configuraciones no dominadas

En la Tabla 6.4 se resume la información obtenida para esta etapa. Para cada configuración se muestra la cantidad de ejecuciones factibles para las instancias de prueba 1, 2, 4 y 6, junto con el total de ejecuciones factibles de la configuración. Además se reportan los tiempos de ejecución mínimo, máximo y promedio considerando las 42 ejecuciones de cada configuración. En la última fila se muestran los valores medios de cada columna. Los valores resaltados corresponden a casos mejores que el promedio de la columna.

[2]Se entiende por ejecución factible a una ejecución en la que se obtuvo una solución factible.

Config.	Soluciones factibles					Tiempo de ejecución		
	1	2	4	6	Total	Min.	Prom.	Max
1	1	4	**3**	**5**	13	**17**	**63**	**111**
2	**2**	4	2	**5**	13	**16**	**66**	**107**
3	0	**6**	0	4	10	**22**	**73**	**133**
4	**2**	5	**4**	4	**15**	32	**104**	**167**
5	**2**	**6**	1	**5**	**14**	**28**	**118**	227
6	1	**6**	1	4	12	35	128	222
7	**2**	5	1	**6**	**14**	**29**	157	288
8	**3**	5	**3**	4	**15**	47	168	282
9	**2**	5	2	3	12	59	176	308
10	0	5	**4**	**5**	**14**	**17**	**69**	**113**
11	0	5	2	**5**	12	**20**	**74**	**131**
12	**4**	3	2	4	13	**23**	**82**	**152**
13	**3**	**6**	**4**	**5**	**18**	**29**	**112**	**191**
14	1	**6**	0	**6**	13	**24**	126	251
15	**2**	**6**	**3**	**5**	**16**	31	135	253
16	1	**6**	**4**	**6**	**17**	32	163	254
17	**2**	**6**	1	**5**	**14**	41	182	353
18	0	**6**	**5**	**5**	**16**	43	181	323
Promedio	1.56	5.28	2.33	4.78	13.94	30	121	215

Tabla 6.4: Soluciones factibles y tiempos de ejecución para cada configuración en el ajuste de parámetros del AMPD.

En cuanto a las soluciones factibles, puede decirse que las configuraciones que implican mayor esfuerzo computacional son las que encuentran la mayor cantidad. Por otro lado, las configuraciones menos exigentes desde el punto de vista computacional son las que resultan en los tiempos de ejecución más bajos. En particular, la cantidad de iteraciones de la búsqueda local (TS^{iter}) aparece como el parámetro más influyente en el tiempo de ejecución promedio de AMPD.

En la Figura 6.1 se muestra, para cada configuración, la cantidad de soluciones factibles encontradas y el tiempo medio de ejecución. Las configuraciones no dominadas seleccionadas en la fase I son 1, 4, 10 y 13. En todos los casos se utiliza el parámetro de diversificacón más bajo de los propuestos ($\gamma = 1.25$). La diferencia entre estas configuraciones es el tamaño de la memoria y la cantidad de iteraciones de la búsqueda local.

Fase II - Ejecuciones no factibles

En la fase II se analizan las soluciones no factibles obtenidas por las configuraciones seleccionadas en la fase I. En la Tabla 6.5 se muestra la cantidad de ejecuciones no factibles, junto con el máximo, el promedio y la desviación estándar de la medida LTR de dichas soluciones. En la Figura 6.2 se presentan los valores de LTR para cada configuración en forma gráfica.

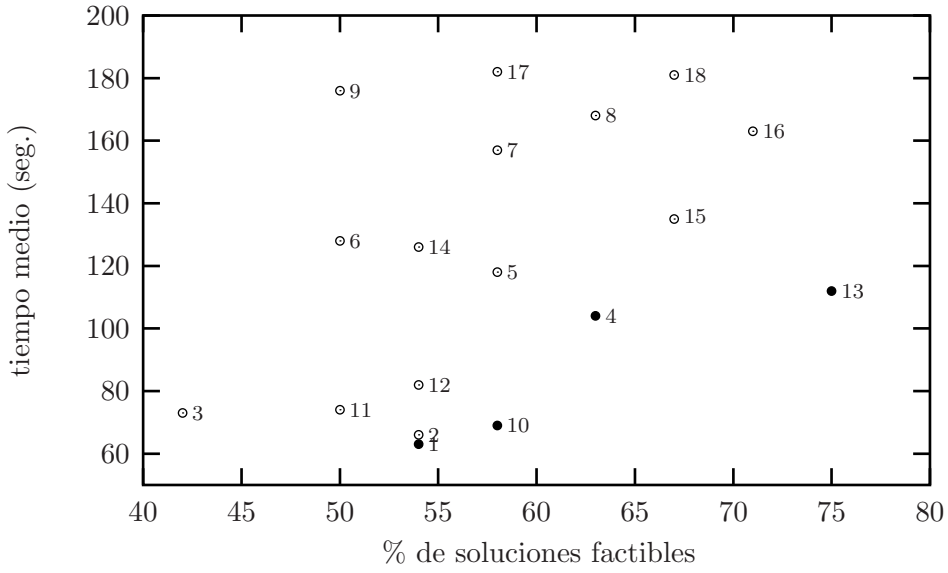


Figura 6.1: Compromiso entre la cantidad de soluciones factibles encontradas y el tiempo medio de ejecución para el AMPD.

Conf.	Casos	Max	Prom.	D. Est.
1	11	1.229	1.056	0.084
4	9	1.189	1.034	0.067
10	10	1.069	1.013	0.020
13	6	1.069	1.033	0.023

Tabla 6.5: Valores de LTR para las configuraciones no dominadas.

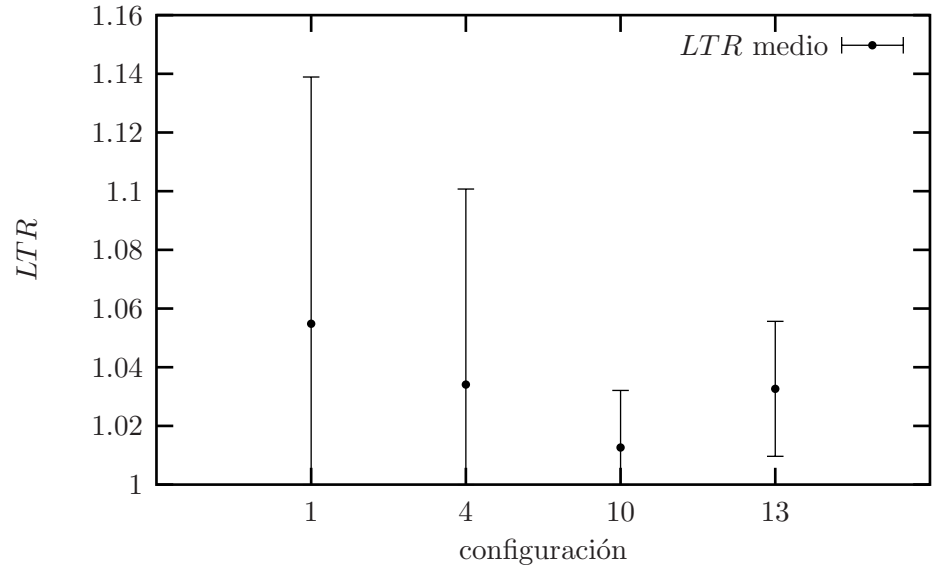


Figura 6.2: LTR medio y desviación estándar para las ejecuciones no factibles de las configuraciones no dominadas en el AMPD.

Criterio	Conf. 1	Conf. 4	Conf. 10	Conf. 13
Tiempo de Ejecución	Bajo	Moderado	Bajo	Alto
Factibilidad	Moderada	Moderada	Moderada	Alta
LTR medio	Moderado	Moderado	Bajo	Moderado
Desv. Estándar LTR	Alta	Alta	Baja	Baja

Tabla 6.6: Resumen del análisis en el ajuste de parámetros del algoritmo AMPD.

La configuración 13 es la que en menos casos obtuvo soluciones no factibles. El resto de las configuraciones tuvieron un comportamiento similar considerando ese aspecto. Las configuraciones 10 y 13 obtuvieron los valores más bajos para el peor caso de la medida LTR, mientras que la configuración 10 resultó la mejor considerando tanto el valor medio como la desviación estándar.

En la Tabla 6.6 se presenta un resumen (subjetivo) del análisis realizado. Se optó por seleccionar la configuración 10, pues obtuvo soluciones de calidad aceptable en tiempos de ejecución razonables. Las diferencias entre los tiempos de ejecución de las configuraciones 10 y 13 resultan, a juicio del autor, más relevantes que las diferencias en cuanto a la cantidad de soluciones factibles obtenidas.

6.3.5. Ajuste de parámetros para AMPE

Al igual que para el AMPD, todas las ejecuciones del AMPE obtuvieron soluciones factibles para las instancias 3, 5 y 7. Solamente se consideraron estas instancias al analizar los tiempos de ejecución.

Fase I - Configuraciones no dominadas

Los resultados para esta etapa se resumen en la Tabla 6.7. Al igual que para el AMPE, las configuraciones que implican mayor esfuerzo computacional dan los mejores resultados, con diferencias poco significativas. Además, las configuraciones con valores más bajos de TS^{iter} son las que presentan los tiempos mas bajos. En la Figura 6.3 se ilustra la cantidad de soluciones factibles encontradas y el tiempo medio de ejecución para cada configuración. Las configuraciones no dominadas seleccionadas en esta fase son las 1, 2 13 y 17.

Fase II - Ejecuciones no factibles

En la Tabla 6.8 se presentan los valores máximo y promedio de la medida LTR, considerando las soluciones no factibles encontradas por cada configuración, así como la desviación estándar. En la Figura 6.4 se muestran esos los valores de LTR para cada configuración en forma gráfica.

Las configuración 17 obtuvo la mayor cantidad de soluciones factibles, mientras que las configuraciones 2 y 13 obtuvieron valores levemente más bajos (pero muy cercanos

Config.	Soluciones factibles					Tiempo de ejecución		
	1	2	4	6	Total	Min.	Prom.	Max
1	0	5	0	4	9	**17**	**55**	**109**
2	**3**	**6**	**2**	**6**	**17**	**15**	**60**	**118**
3	0	5	**3**	5	13	**26**	**79**	**122**
4	2	**6**	1	**6**	15	**20**	**93**	**195**
5	2	**6**	**3**	5	**16**	34	**109**	**179**
6	2	5	**2**	**6**	15	43	154	303
7	**3**	**6**	0	5	14	36	132	275
8	2	**6**	0	**6**	14	44	172	348
9	**4**	5	**2**	**6**	**17**	53	202	387
10	2	5	**2**	**6**	15	**16**	**60**	**107**
11	**3**	**6**	**2**	5	**16**	**20**	**69**	**126**
12	**5**	4	1	**6**	**16**	**26**	**86**	**174**
13	**3**	**6**	**3**	**6**	**18**	**21**	**95**	**200**
14	**4**	5	**3**	**6**	**18**	**28**	120	**217**
15	2	**6**	1	**6**	15	40	153	288
16	**3**	**6**	0	**6**	15	34	131	255
17	**3**	**6**	**4**	**6**	**19**	40	153	256
18	**4**	**6**	**2**	**6**	**18**	55	215	444
Promedio	2.61	5.56	1.72	5.67	15.56	32	119	228

Tabla 6.7: Soluciones factibles y tiempos de ejecución para cada configuración en el ajuste de parámetros del AMPE.

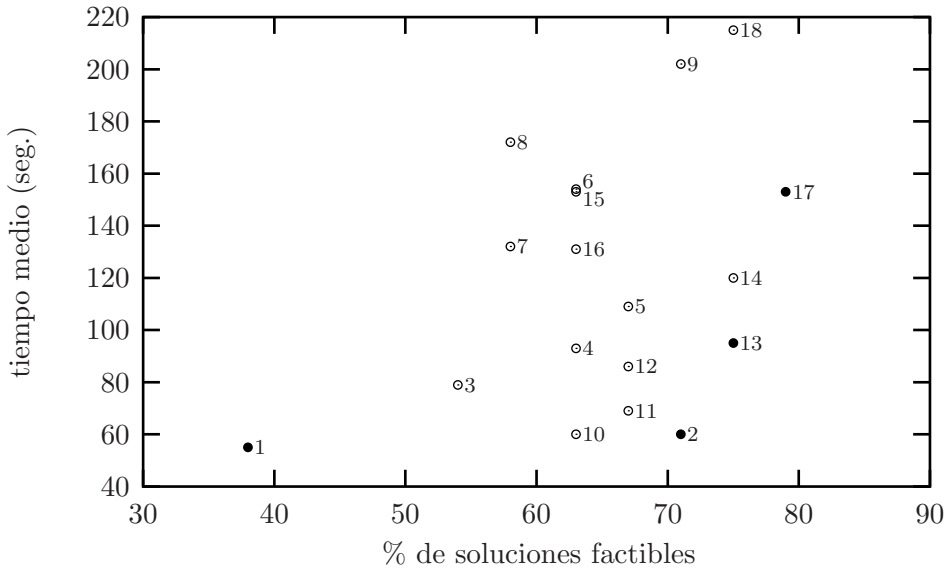


Figura 6.3: Compromiso entre la cantidad de soluciones factibles encontradas y el tiempo medio de ejecución para el AMPE.

Conf.	Casos	Max	Prom.	D. Est.
1	15	1.188	1.053	0.064
2	7	1.188	1.081	0.074
13	6	1.069	1.026	0.034
17	5	1.070	1.033	0.034

Tabla 6.8: Valores medios de LTR para las configuraciones no dominadas.

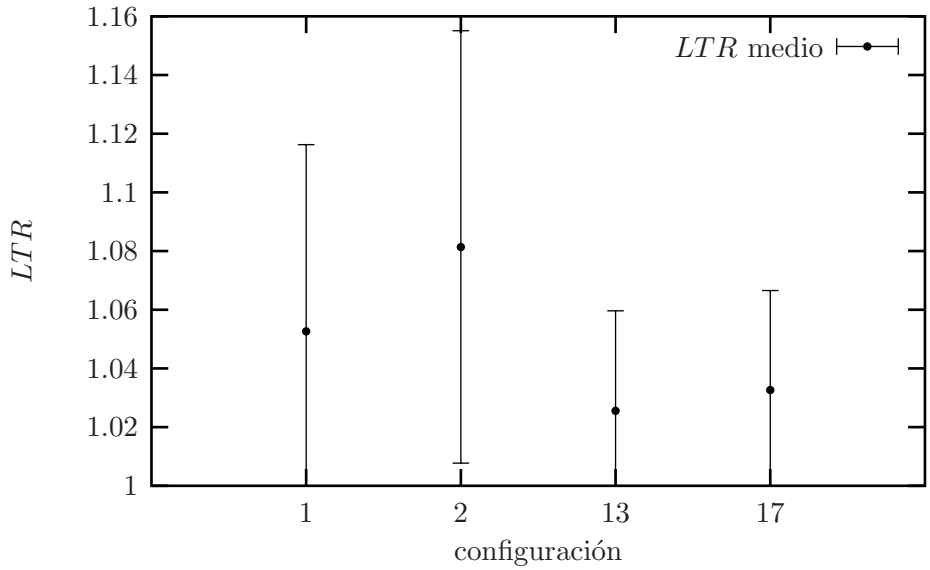


Figura 6.4: LTR medio y desviación estándar para las ejecuciones no factibles de las configuraciones no dominadas en el AMPE.

Criterio	Conf. 1	Conf. 2	Conf. 13	Conf. 17
Tiempo de Ejecución	Bajo	Bajo	Moderado	Alto
Factibilidad	Baja	Alta	Alta	Alta
LTR medio	Alto	Alto	Moderado	Moderado
Desv. Estándar LTR	Alta	Alta	Baja	Baja

Tabla 6.9: Resumen del análisis en el ajuste de parámetros del algoritmo AMPE.

al los de la configuración 17). Los peores resultados en este aspecto fueron obtenidos por la configuración 1. Por otro lado, los tiempos más bajos fueron obtenidos por las configuraciones 1 y 2. La configuración 3 presenta tiempos más altos, aunque moderados, mientras que al configuración 13 tuvo los tiempos más altos (que triplican a los obtenidos por la configuración 1). En cuanto a las soluciones no factibles, las configuraciones 13 y 17 obtienen los resultados más aceptables en términos de la medida LTR, sin diferencias demasiado notorias entre sí.

En la Tabla 6.9 se muestra un resumen del análisis de los resultados. Dado que ofrece un compromiso aceptable entre el tiempo de ejecución y la calidad de las soluciones obtenidas, se optó por seleccionar la configuración 13. Al igual que para el AMPD, se seleccionó una configuración que presente soluciones de calidad aceptable (aunque no sea la que haya obtenido mayor cantidad de soluciones factibles) y tiempos de ejecución moderados.

6.4. Pruebas de desempeño

El objetivo de esta fase es cuantificar el desempeño de los algoritmos AMPD y AMPE en cuanto a la calidad de las soluciones obtenidas y el tiempo de ejecución. Se analizan los resultados obtenidos por cada algoritmo de manera aislada y en comparación con los resultados reportados por otros autores: Taillard et al. [120] (TL), Brandão y Mercer [19] (BM) y Petch y Salhi [104] (PS).

Cada algoritmo se ejecutó sobre cada una de las 104 instancias de prueba, utilizando la configuración de parámetros seleccionada en la fase anterior. Para cada instancia de prueba se realizaron 10 ejecuciones independientes de cada uno de los algoritmos.

6.4.1. Resultados resumidos

En las Tablas 6.10 y 6.11 se muestra un resumen de los resultados obtenidos para los algoritmos AMPD y AMPE respectivamente. Cada fila representa instancias formuladas sobre la misma instancia base. Para cada horizonte de tiempo se reporta la cantidad de instancias en las que alguna de las 10 ejecuciones realizadas obtuvo una solución factible, el total de instancias de la clase, el GAP promedio (considerando las ejecuciones factibles) y el tiempo medio de ejecución. Las instancias se muestran ordenadas en forma no decreciente según la cantidad de nodos.

Ambos algoritmos obtuvieron soluciones factibles para las 52 instancias que utilizan el horizonte de tiempo T_2, al igual que en la propuesta de Brandão y Mercer [19]. Taillard et al. [120] y Petch y Salhi [104] reportan 48 y 46 soluciones factibles sobre estas instancias respectivamente. Con respecto a las 52 instancias con horizonte T_1, tanto el AMPD como el AMPE reportan 46 soluciones factibles (que corresponden a las mismas instancias). Brandão y Mercer [19] obtuvieron 37, Taillard et al. [120] 38 y Petch y Salhi [104] 30.

Instancia Base	n	Horizonte T_1			Horizonte T_2		
		Fact./Tot	GAP	Tiempo	Fact/Tot	GAP	Tiempo
CMT-1	50	2/4	0.83	15	4/4	2.69	8
F-11	71	2/3	1.66	21	3/3	2.32	14
CMT-2	75	6/7	0.79	36	7/7	1.32	24
CMT-3	100	6/6	1.26	43	6/6	0.93	34
CMT-12	100	5/6	0.65	38	6/6	0.19	28
CMT-11	120	5/5	2.91	88	5/5	3.46	67
F-12	134	3/3	0.72	68	3/3	1.04	70
CMT-4	150	8/8	2.29	118	8/8	2.59	83
CMT-5	199	9/10	3.80	249	10/10	5.00	164
		46/52	1.95	–	52/52	2.40	–

Tabla 6.10: Resultados del AMPD agrupados por instancia base.

Instancia Base	n	Horizonte T_1			Horizonte T_2		
		Fact./Tot	GAP	Tiempo	Fact/Tot	GAP	Tiempo
CMT-1	50	2/4	0.82	31	4/4	2.67	17
F-11	71	2/3	1.82	37	3/3	2.09	23
CMT-2	75	6/7	0.41	67	7/7	0.96	45
CMT-3	100	6/6	1.15	67	6/6	0.71	64
CMT-12	100	5/6	0.67	75	6/6	0.18	56
CMT-11	120	5/5	2.80	129	5/5	2.86	99
F-12	134	3/3	0.79	87	3/3	1.01	89
CMT-4	150	8/8	1.79	216	8/8	2.39	154
CMT-5	199	9/10	3.40	399	10/10	4.35	301
		46/52	1.72	–	52/52	2.10	–

Tabla 6.11: Resultados del AMPE agrupados por instancia base.

El GAP medio, considerando ambos horizontes de tiempo, fue 2.19 % para el AMPD y 1.92 % para el AMPE. Teniendo en cuenta que no necesariamente existen soluciones con $GAP = 0$, la calidad de estos resultados puede considerarse aceptable. Dado que obtener soluciones factibles para el VRPMT es un problema complejo desde el punto de vista computacional y además existe una cota superior para el GAP de las soluciones factibles, en los trabajos de otros autores el énfasis está puesto en el análisis de las soluciones no factibles. Por lo tanto, no reportan resultados respecto a los valores de GAP y, en consecuencia, no es posible realizar comparaciones con otros autores respecto a la calidad de las soluciones factibles encontradas.

No se distingue una relación entre los valores de GAP obtenidos y la cantidad de nodos de los problemas. Esto sugiere que la cantidad de nodos no es, por si sola, una medida adecuada de la dificultad de una instancia del problema (al menos con respecto a los algoritmos propuestos en este trabajo). Otros aspectos que deberían considerarse son la relación entre la cantidad de vehículos y el horizonte de tiempo, la distribución de las demandas y la capacidad de los vehículos. Determinar con certeza cuáles son esos factores y cómo se relacionan entre sí, requiere profundizar en la naturaleza combinatoria y las propiedades del VRPMT y de los algoritmos de búsqueda local, lo cual escapa al alcance de este trabajo.

Los tiempos medios de ejecución son razonables, especialmente si se tiene en cuenta la complejidad del problema, los antecedentes sobre la dificultad de las instancias de prueba y la exigencia computacional de los algoritmos de este tipo. El tiempo medio más alto se registró, en ambos casos, en los problemas basados en CMT-5 (que tienen 200 nodos) con horizonte T_1 (que es el más ajustado) y fue cercano a los 4 minutos para el AMPD y a los 7 minutos para el AMPE. Contrariamente a lo observado para el GAP, se aprecia una relación entre los tiempos medios de ejecución y la cantidad de nodos de la instancia. Esto se debe a que la mayor parte del tiempo de ejecución se invierte en la exploración de la vecindad y, dadas las características de las instancias utilizadas, la vecindad de ruteo tiene, en general, más elementos que la vecindad de asignación. Por lo tanto, las instancias con mayor cantidad de nodos necesitan mayor tiempo de ejecución.

Los tiempos medios de ejecución obtenidos por los demás autores, agrupados por instancia base, se muestran en la Tabla 6.12. Si bien estos tiempos son mucho más elevados que los reportados en este trabajo, debe tenerse en cuenta que los algoritmos fueron ejecutados en máquinas con características extremadamente diferentes: Silicion Graphics Indigo - 100 MHz (Taillard et al. [120]), HP Vectra XU Pentium Pro - 200 Mhz (Brandão y Mercer [19]) y Ultra Enterprise 450 - 300 MHz (Petch y Salhi [104]). Realizar una comparación justa de los tiempos de ejecución es extremadamente complejo dadas las grandes diferencias entre los entornos de ejecución de los diferentes algoritmos.

Instancia	AMPD	AMPE	TL	BM	PS
CMT-1	11	24	300	150	108
F-11	17	30	1560	150	258
CMT-2	30	56	420	300	330
CMT-3	38	66	1440	600	828
CMT-12	33	65	1380	600	120
CMT-11	79	114	2700	1500	2430
F-12	70	88	4500	4800	810
CMT-4	101	185	3060	1500	984
CMT-5	207	350	3960	3750	2454

Tabla 6.12: Tiempos medios de ejecución (en segundos).

6.4.2. Análisis comparativo entre el AMPD y el AMPE

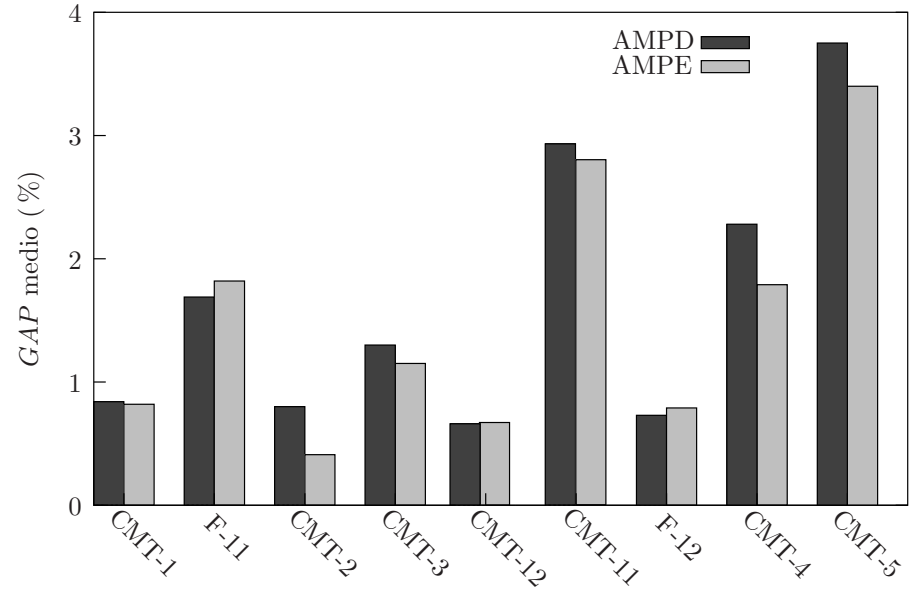
Para facilitar el análisis comparativo entre el AMPD y el AMPE, los valores de las Tablas 6.10 y 6.11 se presentan en forma gráfica. En las Figuras 6.5 y 6.6 se muestran el GAP y el tiempo medio de ejecución obtenidos por ambos algoritmos. Dado que las instancias con T_1 y T_2 revisten diferentes grados de dificultad, se las presenta separadas.

En la Tabla 6.13 se comparan los valores medios de GAP y del tiempo de ejecución obtenidos por ambos algoritmos en términos relativos, agrupando por instancia base y horizonte de tiempo. Para cada grupo de instancias se reporta el valor medio obtenido por el AMPE dividido entre el valor medio obtenido por el AMPD. Estos resultados se presentan en forma gráfica en la Figura 6.7.

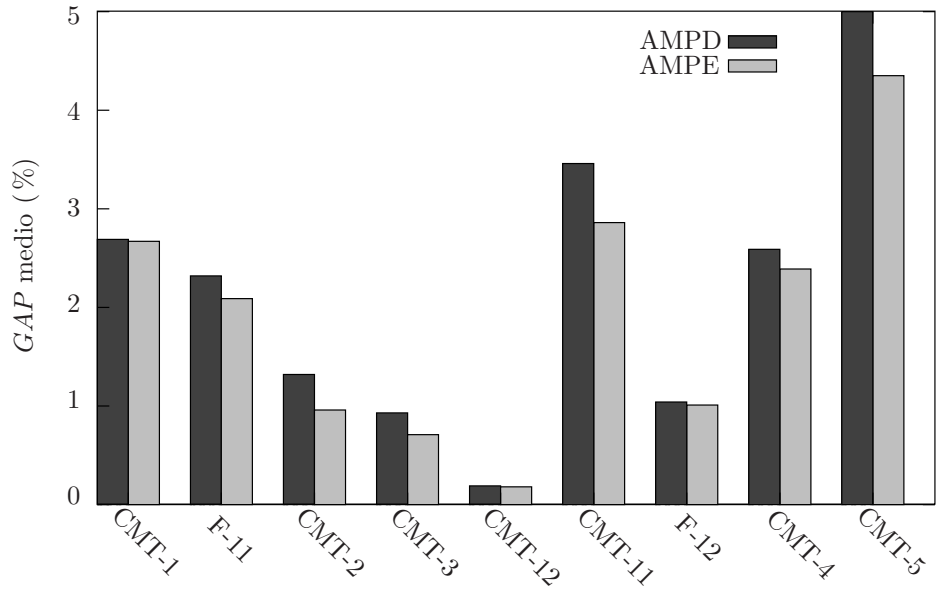
Instancias	n	GAP	Tiempo
CMT-1	50	0.98	2.09
F-11	71	0.98	1.74
CMT-2	75	0.63	1.88
CMT-3	100	0.82	1.72
CMT-12	100	0.84	1.98
CMT-11	120	0.96	1.45
F-12	134	1.04	1.26
CMT-4	150	0.86	1.84
CMT-5	199	0.89	1.71
Promedio		0.89	1.74

Tabla 6.13: Comparación entre GAP y tiempo de ejecución medio.

En la mayor parte de los casos el algoritmo AMPE obtuvo valores de GAP inferiores a los obtenidos por el algoritmo AMPD. Sólo en los casos basados en la instancia F-12 el AMPD supera, en promedio, al AMPE. En promedio, el GAP obtenido por el AMPE fue 0.89 veces el obtenido por el AMPD, lo cual podría considerarse una diferencia poco significativa. Sin embargo, debe tenerse en cuenta que todas las soluciones factibles tienen valores relativamente bajos de GAP (ver Sección 6.2.1) y por lo tanto, es de esperar que las variaciones observadas sean moderadas.

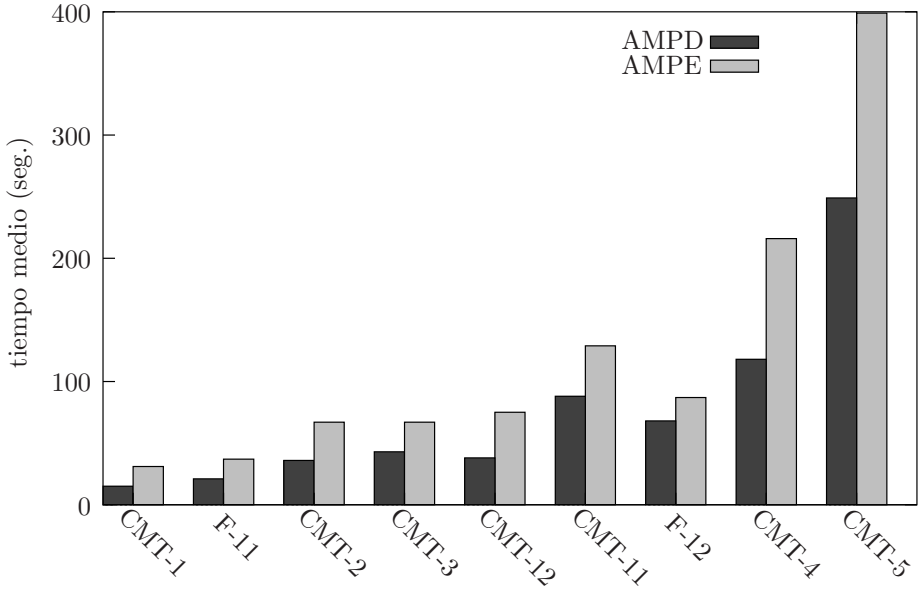


(a) Instancias con T_1.

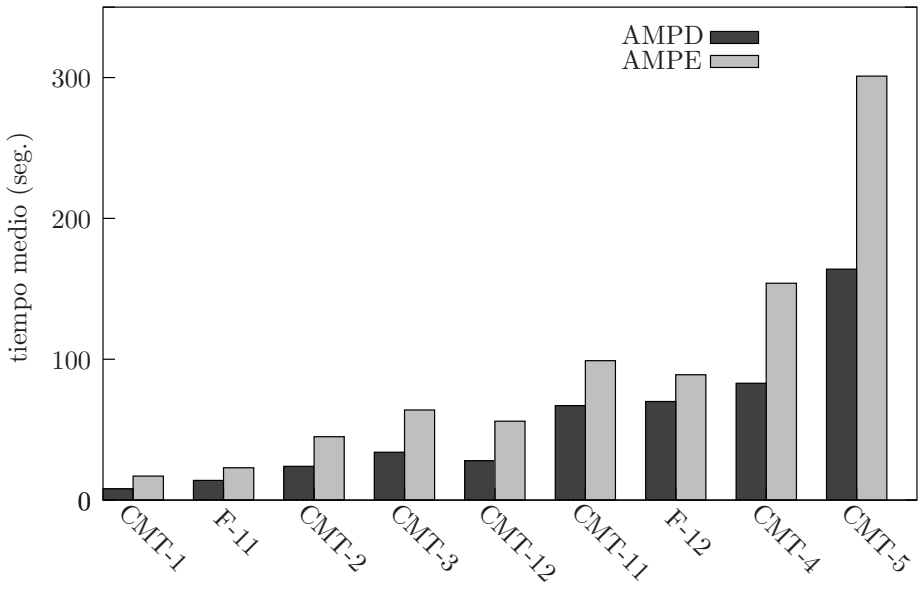


(b) Instancias con T_2.

Figura 6.5: Valores medios de GAP agrupados por instancia base.



(a) Instancias con T_1.



(b) Instancias con T_2.

Figura 6.6: Tiempos medios de ejecución agrupados por instancia base.

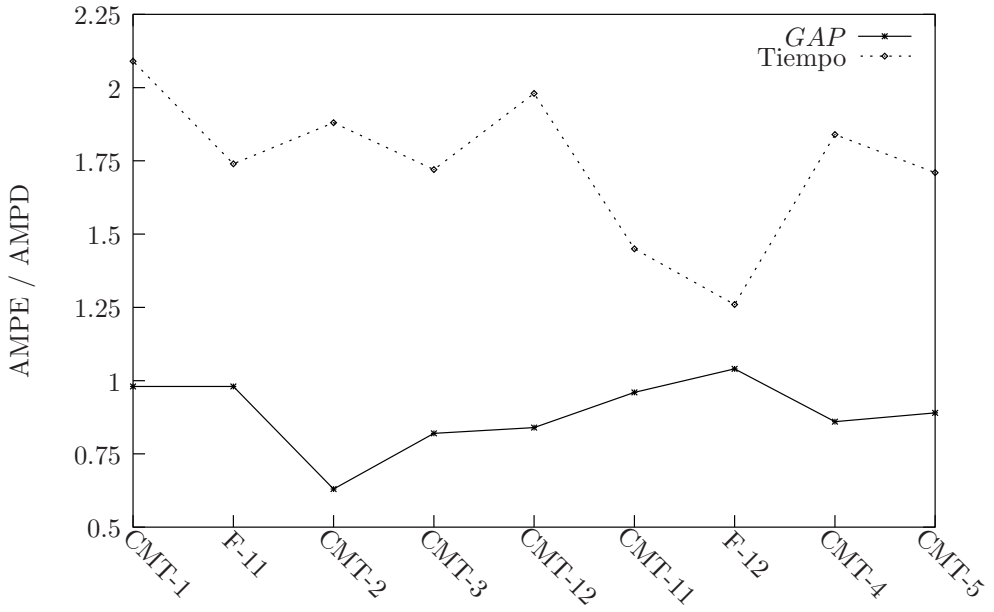


Figura 6.7: Comparación entre AMPE y AMPD para los valores medios del GAP y el tiempo de ejecución.

En todos los casos, el AMPE obtuvo tiempos de ejecución más elevados que el AMPD. En promedio el tiempo de ejecución del AMPE fue 1.74 veces el del AMPD. Esta diferencia resulta más significativa que la obtenida para el GAP. Dado que el tiempo de ejecución está gobernado por la exploración de la vecindad de ruteo, la diferencia sugiere que, para los valores de los parámetros seleccionados, en la estrategia de reducción estática de la vecindad se tienen en cuenta más movidas que en la estrategia de reducción dinámica.

6.4.3. Resultados detallados

En las Tablas 6.14, 6.15 y 6.16 se presentan los resultados obtenidos para cada una de las 104 instancias de prueba. Para cada instancia y cada algoritmo se muestra la cantidad de ejecuciones factibles, el GAP (considerando las ejecuciones factibles) y el tiempo de ejecución (considerando las 10 ejecuciones). Se muestra, cuando corresponde, los valores medios (μ) y las desviaciones estándar (σ). Finalmente, se indica para cada instancia cuáles de los trabajos previos reportan una solución factible y cuáles no.

Tanto el AMPD como el AMPE obtuvieron soluciones factibles en 6 instancias para las que todos los trabajos anteriores reportaban únicamente soluciones no factibles. Dichas instancias son CMT-2 | 6 | T_1, CMT-3 | 6 | T_1, CMT-4 | 7 | T_1, CMT-4 | 8 | T_1, CMT-11 | 4 | T_1 y F-11 | 2 | T_1. Las nuevas soluciones pueden consultarse en el Apéndice D.

Por otro lado, para 6 instancias, ninguno de los dos algoritmos propuestos pudo obtener una solución factible. Estas instancias también fueron reportadas como no factibles en todos los trabajos anteriores [120, 19, 104].

Cualquiera de las dos versiones del AMP representa un avance en la resolución del

Instancia	AMPD					AMPE					Otros		
	Fact.	*GAP*		Tiempo		Fact.	*GAP*		Tiempo		PS	BM	TL
		μ	σ	μ	σ		μ	σ	μ	σ			
CMT-1 | 1 | T_1	10	0.00	0.00	6	1	10	0.00	0.00	14	1	✓	✓	✓
CMT-1 | 2 | T_1	10	1.68	0.20	8	1	10	1.64	0.12	16	1	×	✓	✓
CMT-1 | 3 | T_1	0	–	–	24	1	0	–	–	49	4	×	×	×
CMT-1 | 4 | T_1	0	–	–	21	1	0	–	–	45	1	×	×	×
CMT-1 | 1 | T_2	10	0.00	0.00	6	1	10	0.00	0.00	14	1	✓	✓	✓
CMT-1 | 2 | T_2	10	1.06	0.19	8	1	10	1.00	0.00	16	1	✓	✓	✓
CMT-1 | 3 | T_2	10	5.52	0.24	11	1	10	5.48	0.23	22	1	✓	✓	×
CMT-1 | 4 | T_2	10	4.19	0.08	9	1	10	4.21	0.08	17	0	×	✓	✓
CMT-2 | 1 | T_1	10	0.29	0.37	19	1	10	0.05	0.06	39	1	✓	✓	✓
CMT-2 | 2 | T_1	10	0.37	0.50	21	1	10	0.14	0.27	42	2	✓	✓	✓
CMT-2 | 3 | T_1	10	0.57	0.53	22	1	10	0.09	0.13	43	1	✓	✓	✓
CMT-2 | 4 | T_1	10	0.91	0.41	23	1	10	0.35	0.33	44	1	✓	✓	✓
CMT-2 | 5 | T_1	10	1.06	0.72	23	1	10	0.67	0.49	43	2	✓	✓	✓
CMT-2 | 6 | $\mathbf{T_1}$	4	2.82	0.11	70	18	3	2.91	0.00	117	40	×	×	×
CMT-2 | 7 | T_1	0	–	–	74	4	0	–	–	140	4	×	×	×
CMT-2 | 1 | T_2	10	0.45	0.44	20	1	10	0.10	0.27	39	1	✓	✓	✓
CMT-2 | 2 | T_2	10	0.35	0.33	22	1	10	0.45	0.43	43	1	✓	✓	✓
CMT-2 | 3 | T_2	10	0.50	0.44	23	1	10	0.21	0.27	43	1	✓	✓	✓
CMT-2 | 4 | T_2	10	0.27	0.35	24	1	10	0.35	0.43	45	1	✓	✓	✓
CMT-2 | 5 | T_2	10	1.09	0.46	23	2	10	0.32	0.38	45	2	✓	✓	✓
CMT-2 | 6 | T_2	10	2.95	0.83	28	1	10	1.87	0.75	53	2	✓	✓	✓
CMT-2 | 7 | T_2	10	3.44	0.87	26	1	10	3.42	0.29	50	1	×	✓	×
CMT-3 | 1 | T_1	10	0.35	0.18	28	2	10	0.18	0.15	50	4	✓	✓	✓
CMT-3 | 2 | T_1	10	0.29	0.19	31	2	10	0.28	0.18	56	2	✓	✓	✓
CMT-3 | 3 | T_1	10	0.54	0.39	34	1	10	0.31	0.30	61	3	✓	✓	✓
CMT-3 | 4 | T_1	10	0.69	0.24	37	2	10	0.73	0.23	64	3	✓	✓	✓
CMT-3 | 5 | T_1	10	3.02	0.66	55	20	10	2.38	0.94	65	2	×	✓	×
CMT-3 | 6 | $\mathbf{T_1}$	9	3.09	0.90	68	32	9	3.19	1.07	107	51	×	×	×
CMT-3 | 1 | T_2	10	0.55	0.18	28	3	10	0.26	0.21	52	3	✓	✓	✓
CMT-3 | 2 | T_2	10	0.56	0.24	32	4	10	0.53	0.23	58	4	✓	✓	✓
CMT-3 | 3 | T_2	10	0.42	0.21	34	3	10	0.21	0.16	65	3	✓	✓	✓
CMT-3 | 4 | T_2	10	0.57	0.28	34	5	10	0.45	0.18	68	3	✓	✓	✓
CMT-3 | 5 | T_2	10	1.34	0.26	41	1	10	1.21	0.36	73	1	✓	✓	✓
CMT-3 | 6 | T_2	10	2.14	0.73	39	1	10	1.62	0.89	69	3	✓	✓	✓

Tabla 6.14: Resultados detallados sobre las instancias de prueba basadas en CMT-1, CMT-2 y CMT-3.

Instancia	AMPD					AMPE					Otros		
	Fact.	*GAP*		Tiempo		Fact.	*GAP*		Tiempo		PS	BM	TL
		μ	σ	μ	σ		μ	σ	μ	σ			
CMT-4 | 1 | T_1	10	1.53	0.34	65	3	10	1.26	0.49	127	7	✓	✓	✓
CMT-4 | 2 | T_1	10	1.60	0.36	69	3	10	1.53	0.36	138	4	✓	✓	✓
CMT-4 | 3 | T_1	10	1.96	0.19	74	2	10	1.47	0.45	141	5	✓	✓	✓
CMT-4 | 4 | T_1	10	2.05	0.50	78	3	10	1.66	0.49	144	5	×	✓	✓
CMT-4 | 5 | T_1	10	2.50	0.85	80	3	10	1.63	0.73	147	4	✓	✓	✓
CMT-4 | 6 | T_1	10	3.40	0.61	97	55	10	2.73	0.81	170	55	×	✓	✓
CMT-4 | 7 | T_1	1	3.90	–	244	4	1	4.02	–	440	44	×	×	×
CMT-4 | 8 | T_1	4	3.36	0.47	237	12	2	2.93	0.28	420	64	×	×	×
CMT-4 | 1 | T_2	10	1.44	0.36	69	4	10	1.58	0.45	133	9	✓	✓	✓
CMT-4 | 2 | T_2	10	1.92	0.32	76	5	10	1.64	0.53	143	7	✓	✓	✓
CMT-4 | 3 | T_2	10	1.91	0.55	79	4	10	2.02	0.33	147	5	✓	✓	✓
CMT-4 | 4 | T_2	10	1.85	0.27	80	5	10	1.73	0.47	154	7	✓	✓	✓
CMT-4 | 5 | T_2	10	2.06	0.67	85	3	10	1.92	0.55	161	4	✓	✓	✓
CMT-4 | 6 | T_2	10	2.42	0.61	91	3	10	2.13	0.48	168	6	✓	✓	✓
CMT-4 | 7 | T_2	10	3.87	0.43	87	3	10	2.84	0.55	168	4	×	✓	✓
CMT-4 | 8 | T_2	10	4.79	1.58	100	37	10	5.23	1.95	156	5	✓	✓	×
CMT-5 | 1 | T_1	10	2.88	0.62	132	8	10	2.02	0.54	257	14	✓	✓	✓
CMT-5 | 2 | T_1	10	3.84	0.69	144	42	10	3.57	0.68	263	12	✓	✓	✓
CMT-5 | 3 | T_1	10	3.84	0.56	151	42	10	3.33	0.80	300	92	✓	✓	✓
CMT-5 | 4 | T_1	10	3.88	0.64	195	74	10	3.50	0.78	301	83	✓	✓	✓
CMT-5 | 5 | T_1	10	3.73	0.50	197	75	10	3.23	0.73	301	92	×	✓	✓
CMT-5 | 6 | T_1	10	3.86	0.67	189	99	10	3.85	0.81	278	6	✓	✓	✓
CMT-5 | 7 | T_1	9	3.94	0.28	278	80	10	3.95	0.33	310	95	×	✓	✓
CMT-5 | 8 | T_1	8	4.06	0.53	322	95	10	3.72	0.41	369	126	×	✓	✓
CMT-5 | 9 | T_1	2	3.96	0.69	445	48	6	3.42	0.40	736	203	×	×	✓
CMT-5 | 10 | T_1	0	–	–	456	17	0	–	–	870	20	×	×	×
CMT-5 | 1 | T_2	10	2.96	0.94	145	5	10	2.58	0.67	277	7	✓	✓	✓
CMT-5 | 2 | T_2	10	4.65	0.62	144	4	10	4.36	0.61	279	12	✓	✓	✓
CMT-5 | 3 | T_2	10	5.05	0.89	147	6	10	4.09	1.02	285	7	✓	✓	✓
CMT-5 | 4 | T_2	10	4.55	0.72	152	5	10	4.22	0.45	296	14	✓	✓	✓
CMT-5 | 5 | T_2	10	4.73	0.80	153	5	10	3.71	0.63	299	8	✓	✓	✓
CMT-5 | 6 | T_2	10	5.02	0.72	156	7	10	4.20	0.72	306	11	✓	✓	✓
CMT-5 | 7 | T_2	10	5.08	0.76	153	7	10	3.93	1.18	302	17	✓	✓	✓
CMT-5 | 8 | T_2	10	4.94	0.97	161	10	10	3.78	1.03	317	17	✓	✓	✓
CMT-5 | 9 | T_2	10	5.94	1.11	172	57	10	5.90	1.07	296	8	✓	✓	✓
CMT-5 | 10 | T_2	10	6.74	1.54	250	110	10	6.77	1.20	351	128	×	✓	✓
CMT-11 | 1 | T_1	5	2.90	0.04	61	28	8	2.88	0.03	76	38	✓	✓	✓
CMT-11 | 2 | T_1	3	2.93	0.03	76	26	9	2.90	0.04	71	28	×	✓	✓
CMT-11 | 3 | T_1	9	3.86	0.24	57	24	5	3.05	1.61	127	52	×	✓	✓
CMT-11 | 4 | T_1	2	3.65	0.12	138	16	3	3.68	0.24	193	44	×	×	×
CMT-11 | 5 | T_1	1	0.01	0.01	114	25	2	0.08	0.01	178	24	×	✓	✓
CMT-11 | 1 | T_2	10	5.94	2.12	36	3	10	4.20	2.55	60	4	✓	✓	✓
CMT-11 | 2 | T_2	10	4.51	1.94	40	3	10	3.61	2.09	62	2	✓	✓	✓
CMT-11 | 3 | T_2	10	2.08	1.56	50	11	10	2.88	0.97	71	3	✓	✓	✓
CMT-11 | 4 | T_2	8	0.78	1.72	98	44	8	0.78	1.88	122	60	×	✓	✓
CMT-11 | 5 | T_2	2	3.48	0.37	118	36	3	1.36	2.13	181	58	✓	✓	✓

Tabla 6.15: Resultados detallados sobre las instancias de prueba basadas en CMT-4, CMT-5 y F-11.

Instancia	AMPD					AMPE					Otros		
	Fact.	*GAP*		Tiempo		Fact.	*GAP*		Tiempo		PS	BM	TL
		μ	σ	μ	σ		μ	σ	μ	σ			
CMT-12 | 1 | T_1	10	0.00	0.00	22	1	10	0.00	0.00	46	1	✓	✓	✓
CMT-12 | 2 | T_1	10	0.00	0.00	25	1	10	0.00	0.00	51	1	✓	✓	✓
CMT-12 | 3 | T_1	10	0.00	0.00	28	1	10	0.00	0.00	54	1	✓	✓	✓
CMT-12 | 4 | T_1	10	0.00	0.00	28	1	10	0.00	0.00	56	1	✓	×	✓
CMT-12 | 5 | T_1	10	3.31	0.27	31	2	10	3.33	0.26	58	3	✓	×	×
CMT-12 | 6 | T_1	0	–	–	92	1	0	–	–	183	4	×	×	×
CMT-12 | 1 | T_2	10	0.02	0.05	22	1	10	0.00	0.00	46	1	✓	✓	✓
CMT-12 | 2 | T_2	10	0.06	0.20	26	1	10	0.00	0.00	52	1	✓	✓	✓
CMT-12 | 3 | T_2	10	0.00	0.00	28	1	10	0.00	0.00	56	1	✓	✓	✓
CMT-12 | 4 | T_2	10	0.02	0.05	30	1	10	0.00	0.00	59	1	✓	✓	✓
CMT-12 | 5 | T_2	10	0.65	0.03	33	1	10	0.64	0.00	64	2	✓	✓	✓
CMT-12 | 6 | T_2	10	0.44	0.00	33	1	10	0.44	0.00	61	2	✓	✓	✓
F-11 | 1 | T_1	10	0.35	0.47	9	1	10	0.00	0.00	18	1	✓	✓	✓
F-11 | 2 | T_1	6	3.92	0.46	23	8	8	4.08	0.41	32	16	×	×	×
F-11 | 3 | T_1	0	–	–	32	3	0	–	–	62	3	×	×	×
F-11 | 1 | T_2	10	0.58	0.53	9	1	10	0.04	0.11	19	1	✓	✓	✓
F-11 | 2 | T_2	10	0.87	1.25	14	1	10	0.51	0.96	24	1	✓	✓	✓
F-11 | 3 | T_2	10	5.68	1.03	17	1	10	5.72	1.01	28	3	✓	✓	×
F-12 | 1 | T_1	10	0.64	0.14	56	4	10	0.60	0.15	75	3	✓	✓	✓
F-12 | 2 | T_1	10	0.92	0.12	73	4	10	0.88	0.17	90	8	✓	✓	✓
F-12 | 3 | T_1	10	0.64	0.25	76	5	10	0.91	0.27	95	8	✓	✓	✓
F-12 | 1 | T_2	10	0.96	0.31	63	5	10	1.04	0.37	78	4	✓	✓	✓
F-12 | 2 | T_2	10	0.96	0.37	72	5	10	1.12	0.60	91	7	✓	✓	✓
F-12 | 3 | T_2	10	1.15	0.43	78	9	10	0.86	0.31	99	8	✓	✓	✓

Tabla 6.16: Resultados detallados sobre las instancias de prueba basadas en CMT-12, F-11 y F-12.

VRPMT, en la medida que permitieron resolver una mayor cantidad de instancias de prueba del problema.

Para la mayoría de las instancias en las que se obtuvieron soluciones factibles, la desviación estándar del GAP fue moderada (el valor máximo fue 2.55 %). En algunas instancias, el tiempo de ejecución presenta una desviación estándar algo elevada. Esto se debe a que el criterio de terminación del algoritmo provoca que en algunos casos se ejecuten 100 iteraciones, en otros 200 y en otros 300, según si se encontró una solución factible o no.

6.4.4. Análisis de las instancias no factibles

Ambos algoritmos reportaron soluciones no factibles para las siguientes 6 instancias:

1. CMT-1 | 3 | T_1,
2. CMT-1 | 4 | T_1,
3. CMT-2 | 7 | T_1,
4. CMT-5 | 10 | T_1,
5. CMT-12 | 6 | T_1,
6. F-11 | 3 | T_1,

En lo que queda de este capítulo, se analizan las soluciones obtenidas por los algoritmos AMPD y AMPE para estas instancias. Dadas las 10 ejecuciones de cada algoritmo sobre cada instancia (todas las cuales resultaron en soluciones no factibles), se consideran las medidas definidas en la Sección 4.7. Para cada medida se reporta el mejor caso, el peor caso y el caso promedio. Dichos valores son comparados con los reportados en los trabajos anteriores (en los que se da siempre la solución obtenida por la mejor ejecución, es decir, el mejor caso).

Medida LTR

En la Tabla 6.17 se muestran los resultados respecto a la medida LTR (para cada instancia se resaltan los mejores valores). Los resultados obtenidos por el AMPD y el AMPE son similares. Se observa que el AMPD obtiene un valor medio levemente inferior en el mejor caso y el AMPE, en cambio, obtiene un mejor valor para el peor caso y el caso promedio. Las diferencias entre ambos algoritmos respecto a esta medida no deben considerarse como relevantes, dada la poca cantidad de casos no factibles y lo pequeño de dichas diferencias.

En comparación con los resultados obtenidos por otros autores, ambos algoritmos reportan mejoras respecto a esta medida en 4 de las 6 instancias analizadas. Las diferencias

Inst.	AMPD			AMPE			Otros		
	Min	Max	Prom.	Min	Max	Prom.	BM	TL	PS
1	1.032	1.038	1.033	1.032	1.032	1.032	1.041	1.115	**1.026**
2	**1.027**	**1.027**	**1.027**	**1.027**	**1.027**	**1.027**	**1.027**	**1.027**	1.085
3	1.008	1.064	1.029	**1.004**	1.050	1.017	1.088	1.073	1.064
4	**1.021**	1.097	1.066	1.031	1.075	1.051	1.051	1.024	1.064
5	**1.014**	1.029	1.017	**1.014**	1.029	1.017	1.072	1.064	1.029
6	1.020	1.048	1.031	1.020	1.048	1.035	**1.011**	1.070	1.020
Prom.	**1.020**	1.051	1.034	1.021	1.044	1.030	1.048	1.062	1.048

Tabla 6.17: Comparación de LTR para las instancias reportadas como no factibles.

	AMPD			AMPE		
	BM	TL	PS	BM	TL	PS
Min	−28.4	−53.9	−42.0	−25.9	−47.9	−40.4
Prom.	8.1	−14.3	−13.2	6.5	−26.7	−17.6
Max	55.4	23.1	28.3	43.1	3.8	15.0

Tabla 6.18: Diferencia promedio (%) entre los valores de LTR obtenidos en el mejor caso, caso promedio y peor caso y los valores reportados por otros autores.

promedio entre los algoritmos propuestos en este trabajo y los propuestos por los demás autores, en términos porcentuales[3], se muestran en la Tabla 6.18. Los mejores valores obtenidos por el AMPD y el AMPE superan considerablemente a los reportados en la literatura. Además, los valores medios de LTR obtenidos superan a los mejores valores obtenidos por Taillard et al. [120] y Petch y Salhi [104]. Solo los peores casos reportados en este trabajo son superados de forma considerable por los mejores valores reportados por otros autores.

Medida OT

Los valores comparativos respecto al overtime normalizado (OT) se muestran en la Tabla 6.19. No se incluyen resultados de Petch y Salhi [104] pues en dicho trabajo no se reportan datos respecto a esta medida. El algoritmo AMPE obtiene mejores resultados que el AMPD, aunque al igual que para la medida LTR, las diferencias no parecen muy significativas.

Respecto a esta medida, en 4 de las 6 instancias analizadas se obtienen mejores resultados que los reportados hasta el momento. Las diferencias porcentuales respecto a los resultados obtenidos por Brandão y Mercer [19] y Taillard et al. [120] se muestran en la Tabla 6.20. Al igual que para la medida LTR, los mejores resultados obtenidos en este trabajo superan, en promedio, a los mejores resultados reportados y las diferencias resultan significativas. El peor caso reportado por el AMPE supera a las mejores soluciones

[3]Si v es el valor reportado en este trabajo para cierta medida y a es el valor reportado en otro trabajo, las *comparaciones en términos porcentuales* se calculan como $\frac{v-a}{a}$.

Inst.	AMPD			AMPE			Otros	
	Min	Max	Prom.	Min	Max	Prom.	BM	TL
1	**3.56**	4.03	3.63	**3.56**	**3.56**	**3.56**	5.27	12.63
2	**4.80**	**4.80**	**4.80**	**4.80**	**4.80**	**4.80**	6.15	6.88
3	1.23	7.79	3.66	**0.45**	6.12	2.82	10.60	13.88
4	**5.45**	21.97	14.14	5.73	18.67	12.08	8.40	5.55
5	2.66	3.10	2.75	2.66	3.10	2.75	**2.23**	9.06
6	2.16	4.81	3.31	2.16	4.81	3.58	**1.81**	7.48
Prom.	3.31	7.75	5.38	**3.23**	6.84	4.93	5.74	9.25

Tabla 6.19: Comparación de la medida OT (%) para las instancias reportadas como no factibles.

	AMPD		AMPE	
	BM	TL	BM	TL
Min	−23.2	−56.1	−23.9	−56.2
Prom.	9.3	−24.3	6.1	−31.0
Max	49.0	8.7	38.4	−3.9

Tabla 6.20: Diferencia promedio (%) entre los valores de OT obtenidos en el mejor caso, caso promedio y peor caso y los valores reportados por otros autores.

obtenidas por Taillard et al. [120], respecto a esta medida. Los valores reportados por Brandão y Mercer [19], en cambio, superan al caso promedio (y en consecuencia, al peor caso) del AMPD y del AMPE.

Medida PC_δ

Finalmente, en la Tabla 6.21 se reportan los resultados respecto a la medida PC_δ para $\delta \in \{2, 3\}$. Dado que ésta no es una medida normalizada, sino que su valor depende de la magnitud de los costos c_{ij}, no se reportan los valores medios. Los resultados obtenidos por ambos algoritmos son similares. El AMPE obtiene mejores valores que el AMPD en la mayor parte de los casos, pero las diferencias en ningún caso superan el 1.2 %. En 3 de las 6 instancias, los algoritmos propuestos en este trabajo obtienen mejores resultados que los reportados anteriormente por Brandão y Mercer [19] y Taillard et al. [120].

Al analizar los valores de la medida PC_δ debe tenerse presente que, como se mencionó en la Sección 4.7.2, respetar las restricciones de overtime y minimizar el costo total pueden ser objetivos encontrados. Soluciones con valores bajos de la medida PC_δ pueden conseguirse incurriendo en violaciones a la restricción de overtime. En los algoritmos propuestos en este trabajo, al comparar dos soluciones, en primera instancia se analiza la medida $O(s)$. Sólo se tiene en cuenta el costo de las soluciones cuando ambas son factibles. Esta estrategia supone, como es usual en los problemas de optimización, que es preferible una solución factible con costo elevado que una solución no factible de bajo costo. Comparar estos algoritmos con otros que no se basan en este principio puede conducir

Ins.	δ	AMPE			AMPD			Otros	
		Min	Max	Prom.	Min	Max	Prom.	BM	TL
1	2	**562.76**	**562.76**	**562.76**	**562.76**	565.95	563.20	575.73	579.48
	3	**569.30**	**569.30**	**569.30**	**569.30**	573.36	569.88	585.43	602.72
2	2	**561.71**	**561.71**	**561.71**	**561.71**	**561.71**	**561.71**	564.07	565.27
	3	**568.34**	**568.34**	**568.34**	**568.34**	**568.34**	**568.34**	572.56	574.76
3	2	**864.64**	885.88	876.03	874.25	890.44	878.15	896.57	878.29
	3	**872.30**	892.99	879.56	875.78	900.18	882.72	909.82	895.64
4	2	1356.58	1404.64	1380.70	1361.49	1407.25	1388.79	1371.17	**1338.64**
	3	1367.01	1429.00	1397.12	1371.64	1437.13	1408.03	1371.17	**1338.64**
5	2	856.80	857.52	856.94	856.80	857.52	856.94	**825.94**	845.48
	3	860.61	861.96	860.88	860.61	861.96	860.88	**829.13**	858.44
6	2	257.91	259.72	258.79	257.91	259.73	258.93	257.47	**257.31**
	3	259.74	263.81	261.83	259.74	263.81	261.74	**259.01**	263.67

Tabla 6.21: Comparación de la medida PC_δ para $\delta \in \{2, 3\}$ en las instancias reportadas como no factibles.

	AMPD		AMPE	
	BM	TL	BM	TL
Min	−0.42	−0.70	−0.60	−0.88
Prom.	0.18	−0.08	0.00	−0.27
Max	0.94	0.69	0.66	0.41

Tabla 6.22: Diferencia promedio (%) entre los valores de PC_δ obtenidos en el mejor caso, caso promedio y peor caso y los valores reportados por otros autores (para $\delta \in \{2, 3\}$).

a resultados engañosos. De todos modos, por completitud del análisis, se prefirió incluir estos resultados junto con la advertencia precedente.

Las diferencias porcentuales se muestran en la Tabla 6.22. Al igual que con las medidas anteriores, tanto el AMPD como el AMPE obtienen, en el mejor caso, mejores valores que los otros autores. La magnitud de las diferencias es muy pequeña en términos porcentuales, porque los valores de esta medida son grandes en comparación con las diferencias observadas. Los resultados obtenidos por Brandão y Mercer [19] en el mejor caso superan al caso promedio y al peor caso reportados en este trabajo.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

En este trabajo se estudia el Problema de Ruteo de Vehículos con Múltiples Viajes o *Vehicle Routing Problem with Multiple Trips* (VRPMT). Esta es una importante variante del problema clásico de ruteo de vehículos. La característica de poder realizar múltiples viajes introduce una nueva dimensión en los problemas. Además de una componente de ruteo que busca determinar las secuencias de clientes a visitar, existe una componente de asignación que se ocupa de distribuir las rutas entre los vehículos. La ausencia de esta componente en la mayor parte de los modelos abordados en la literatura académica, disminuye su utilidad en algunos contextos de planificación operativa. De poco sirve, en ese contexto, diseñar un conjunto de rutas que no pueden ser llevadas a la práctica por los vehículos disponibles.

Además de revestir gran interés práctico, el VRPMT es complejo desde el punto de vista computacional. El problema de encontrar una solución óptima pertenece a la clase NP-Hard, al igual que muchos otros problemas de optimización combinatoria. Lo que hace que el VRPMT sea realmente difícil de tratar es que encontrar una solución factible es también NP-Hard. Por lo tanto, es de esperar que las heurísticas constructivas simples, basadas en un conjunto reducido de reglas, fallen en su intento por resolver el problema. Esta característica exige, sobre todo para instancias muy ajustadas, el uso de métodos de solución que realicen una buena exploración del espacio de soluciones.

Los antecedentes encontrados en la literatura académica acerca de la resolución de este problema son escasos. Esto resulta sorprendente si se tiene en cuenta la relevancia de este problema, al menos a nivel operativo.

En este trabajo se propone un algoritmo basado en el *Adaptive Memory Procedure* [112] para resolver el VRPMT. Esta estrategia consiste en mantener una memoria con rutas que pertenecen a algunas de las mejores soluciones visitadas por el algoritmo. Las rutas en la memoria se utilizan para construir soluciones iniciales para un algoritmo de búsqueda

local. A su vez, la solución obtenida por la búsqueda local se utiliza para actualizar la información de la memoria.

La búsqueda local utilizada es una implementación de la metaheurística *Tabu Search* [66] y toma elementos de dos algoritmos propuestos originalmente para el resolver el VRP: *Taburoute* [60] y *Granular Tabu Search* [127].

A diferencia de las propuestas anteriores para resolver el VRPMT, las componentes de asignación y de ruteo se consideran simultáneamente. Para ello, se define una vecindad que modifica la secuencia de nodos de las rutas y otra que modifica la asignación de las rutas a los vehículos. En cada iteración se examinan ambas vecindades y se selecciona la mejor solución. Por lo tanto, en algunos casos se modifican las rutas y en otros se modifica la asignación, según lo que sea más conveniente. Esto aporta flexibilidad al algoritmo.

Durante la ejecución del algoritmo se permite visitar soluciones no factibles. Estas soluciones son penalizadas en la función objetivo, utilizando factores de penalización que se ajustan automáticamente mediante reglas sencillas. Se produce así una oscilación entre soluciones factibles y no factibles. Esta estrategia trae consigo dos ventajas importantes. Por un lado, disminuye las dificultades introducidas por la presencia de soluciones no factibles, que suelen provocar que ciertas zonas del espacio de soluciones no puedan ser alcanzadas. Además, simplifica la incorporación de restricciones al problema. Para considerar una nueva restricción no es necesario conocer su impacto en el modelo, sino tan solo definir una medida para cuantificar las soluciones que no la satisfacen.

El uso de estrategias de reducción de la vecindad hizo viable la ejecución del algoritmo en tiempos moderados. Las dos estrategias propuestas obtuvieron resultados similares en cuanto a la calidad de las soluciones obtenidas. La estrategia de reducción estática de la vecindad, sin embargo, registró tiempos que duplicaron a los obtenidos por la estrategia de reducción dinámica. No obstante, parece apresurado inferir alguna conclusión general al respecto, ya que seguramente este comportamiento esté ligado a los valores particulares seleccionados para los parámetros.

Las soluciones obtenidas por los algoritmos propuestos pueden considerarse de calidad aceptable. Los tiempos de ejecución fueron relativamente moderados considerando la complejidad del problema y lo ajustado de las instancias de prueba utilizadas. Se encontraron soluciones factibles para instancias en las que todos los trabajos anteriores fallaban. En las instancias para las que no se pudo encontrar soluciones, se logró un buen compromiso entre el costo de la solución y la cercanía a la factibilidad, mejorando incluso algunos resultados previos.

7.2. Trabajo futuro

Por su relevancia práctica y por el desafío que implica para las técnicas de optimización, creemos que los problemas de ruteo de vehículos que consideran una componente de asignación merecen más atención. La inclusión de esta nueva dimensión de manera sistemática contribuiría a reducir la brecha existente entre los problemas estudiados en la academia y los problemas que surgen en la realidad.

En ese sentido, una posible línea de trabajo consiste en agregar restricciones al VRPMT, formulando modelos más realistas. En primer lugar, podría estudiarse una versión del problema con flota heterogénea, es decir, en la que los vehículos tienen diferentes capacidades, costos de viaje y velocidades. Esta característica podría incluirse en los algoritmos propuestos en este trabajo modificando la evaluación de la vecindad de asignación. También puede buscarse adaptar el algoritmo para considerar ventanas de tiempo, aunque se advierte que esa restricción implica una mayor complejidad dado que la duración de un conjunto de rutas dependería del orden en que estas son recorridas. Considerar múltiples depósitos es otra alternativa a tener en cuenta. En cualquiera de los casos debe diseñarse un conjunto de instancias de prueba.

Una característica cuestionable del modelo planteado en este trabajo es que la restricción de horizonte de tiempo puede resultar demasiado rígida. En la práctica, puede ser posible admitir que el día de trabajo finalice algo más tarde si eso contribuye a disminuir los costos de la distribución. Una alternativa para contemplar esta realidad consiste en formular el VRPMT como un problema multi-objetivo [40] en el que se busca, por un lado, minimizar los costos de las rutas y, por otro lado, minimizar alguna de las medidas que en este trabajo se utilizan para cuantificar las infactibilidades respecto a la restricción de overtime (LTR, O, OT).

El comportamiento de los algoritmos presentados depende de una gran cantidad de parámetros. Encontrar valores para los parámetros requiere de un gran esfuerzo que ofrece pocas garantías. Esto deteriora su portabilidad e introduce una dificultad en su uso. Sería interesante encontrar alternativas que disminuyan la cantidad de parámetros a ajustar. Una posibilidad es la incorporación de mecanismos para que los parámetros se ajusten de manera automática durante la ejecución del algoritmo.

Algunos algoritmos de optimización, en particular algunos métodos exactos, obtienen resultados muy diferentes dependiendo del tipo de instancias que se resuelven. En particular, suele ocurrir que algunos métodos que obtienen muy buenos resultados para instancias de prueba ajustadas (es decir, con pocas soluciones factibles), ven deteriorado su desempeño al trabajar con instancias de diferentes características. Dado que en este trabajo la experimentación se realizó sobre instancias muy ajustadas, interesa estudiar el comportamiento de los algoritmos para otro tipo de instancias.

Bibliografía

[1] E. Aarts, J. Korst, P. van Laarhoven. Simulated Annealing. En *Local search in Combinatorial Optimization*, E. Aarts, J. Lenstra, editores, páginas 91–120. John Wiley and Sons, 1997.

[2] E. Aarts, J. Lenstra. *Local search in Combinatorial Optimization*. Wiley, Chichester, 1997.

[3] Y. Agarwal, K. Mathur, H. Salkin. A set-partitioning-based algorithm for the vehicle routing problem. *Networks*, 19:731–750, 1989.

[4] R. Agarwala, D. Applegate, D. Maglott, G. Schuler, A. Schäffer. A fast and scalable radiation hybrid map construction and integration strategy. *Genome Research*, 10:350–364, 2000.

[5] S. Ahmadi, I. Osman. Greedy random adaptive memory programming search for the capacitated clustering problem. *European Journal of Operational Research*, 162:30–44, 2005.

[6] K. Altinkemer, B. Gavish. Parallel savings based heuristics for the delivery problem. *Operations Research*, 39:456–469, 1991.

[7] C. Bailey, T. McLain, R. Beard. Fuel saving strategies for dual spacecraft interferometry missions. *Journal of the Astronautical Sciences*, 49:469–488, 2001.

[8] B. Baker, M. Ayechew. A genetic algorithm for the vehicle routing problem. *Computers and Operational Research*, 30:787–800, 2003.

[9] M. Balinski, R. Quandt. On an integer program for a delivery problem. *Operations Research*, 12:300–304, 1964.

[10] J. Beasley. Route first – cluster second methods for vehicle routing. *Omega*, 11:403–408, 1983.

[11] J. Berger, M. Barkaoui, O. Bräysy. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *INFOR*, 41:179–194, 2003.

[12] J. Blanton, R. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. En *Proceedings of the 5th International Conference on Genetic Algorithms*, páginas 452–459, 1993.

[13] C. Blum, A. Roli. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.

[14] F. Boctor, J. Renaud. The column-circular subsets-selection problem: complexity and solutions. *Computers and Operations Research*, 27:383–398, 2000.

[15] L. Bodin, B. Golden, A. Assad, M. Ball. Routing and scheduling of vehicles and crews – the state of the art. *Computers and Operations Research*, 10(2):63–211, 1983.

[16] B. Bozkaya, E. Erkut, G. Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, 144:12–26, 2003.

[17] J. Bramel, D. Simchi-Levi. A location based heuristic for general routing problems. *Operations Research*, 43(4):649–660, 1995.

[18] J. Brandão, A. Mercer. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research*, 100:180–191, 1997.

[19] J. Brandão, A. Mercer. The multi-trip vehicle routing problem. *Journal of the Operational Research Society*, 49:799–805, 1998.

[20] O. Bräysy, W. Dullaert. A fast evolutionary metaheuristic for the vehicle routing problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(2):153–172, 2003.

[21] B. Bullnheimer, R. Hard, C. Strauss. An improved ant system for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[22] B. Bullnheimer, R. Hartl, C. Strauss. Applying the ant system to the vehicle routing problem. En *Proceedings of the 2nd International Conference on Metaheuristics (MIC'97)*, Sophia-Antipolis, France, 1997.

[23] B. Bullnheimer, R. Hartl, C. Strauss. A new rank based version of the ant system – a computational study. Reporte técnico, University of Viena, Institute of Management Science, 1997.

[24] N. Christofides. The vehicle routing problem. *RAIRO (recherche opérationnelle)*, 26:736–743, 1980.

[25] N. Christofides, S. Eilon. An algorithm for the vehicle dispatching problem. *Operational Research Quarterly*, 20:309–318, 1969.

[26] N. Christofides, A. Mingozzi, P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree shortest path relaxations. *Mathematical Programming*, 20:255–282, 1981.

[27] N. Christofides, A. Mingozzi, P. Toth. Space state relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[28] N. Christofides, A. Mingozzi, P. Toth. The Vehicle Routing Problem. En *Combinatorial Optimization*, N. Christofides, A. Mingozzi, P. Toth, C. Sandi, editores, páginas 315–338. Wiley, Chichester, 1979.

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

[30] A. Colorni, M. Dorigo, V. Maniezzo. Distributed optimization by ant colonies. En F. Varela, P. Bourgine, editores, *Proceedings of the European Conference on Artificial Life*, páginas 134–142. Elsevier, Amsterdam, 1991.

[31] F. Cordeau, G. Desaulniers, J. Desrosiers, M. Solomon, F. Soumis. The VRP with time windows. Reporte Técnico Cahiers du GERAD G-99-13, École des Hautes Études Commerciales de Montréal, 1999.

[32] F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, J. Sormany. New heuristics for the vehicle routing problem. Reporte Técnico CRT-2004-33, Centre de recherche sur les transports, Montreal, 2004.

[33] F. Cordeau, G. Laporte. Modeling and optimization of vehicle routing and arc routing problems. Reporte Técnico CRT-2002-30, Centre de recherche sur les transports, Montreal, 2002.

[34] F. Cordeau, G. Laporte, A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

[35] T. Crainic, G. Laporte. *Fleet management and logistics*. Kluwer Academic Publishers, 1998.

[36] B. Crevier, F. Cordeau, E. Taillard. The multi-depot vehicle routing problem with inter-depot routes. Reporte Técnico CRT-2004-10, Centre de recherche sur les transports, Montreal, 2004.

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

[38] G. Dantzig, J. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[39] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[40] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

[41] M. Desrochers, J. Desrosiers, M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–355, 1992.

[42] M. Desrochers, T. Verhoog. A matching based savings algorithm for the vehicle routing problem. Reporte Técnico Cahiers du GERAD G-89-04, École des Hautes Études Commerciales de Montréal, 1989.

[43] J. Desrosiers, F. Soumis, M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[44] M. Dorigo, V. Maniezzo, A. Colorni. The Ant System: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[45] W. Dullaert, G. Janssens, K. Sorensen, B. Vernimmen. New heuristics for the fleet size and mix vehicle routing problem with time windows. *Journal of the Operational Research Society*, 53:1232–1238, 2002.

[46] S. Eilon, C. Watson-Gandy, N. Christofides. *Distribution management: mathematical modelling and practical analysis*. Griffin, 1971.

[47] T. Feo, M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[48] M. Fischetti, P. Toth, D. Vigo. A branch and bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42:846–859, 1994.

[49] M. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994.

[50] M. Fisher, R. Jaikumar. A generalized assignment heuristic for the vehicle routing problem. *Networks*, 11:109–124, 1981.

[51] B. Fleischmann. The vehicle routing problem with multiple use of vehicles. Working Paper. Fachbereich Wirtschaftswissenschaften, Universität Hamburg, 1990.

[52] C. Fleurent, F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11:198–204, 1999.

[53] B. Foster, D. Ryan. An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27(3):367–384, 1976.

[54] L. Gambardella, M. Dorigo. Ant-Q: a reinforcement learning approach to the traveling salesman problem. En *International Conference on Machine Learning*, páginas 252–260, 1995.

[55] L. Gambardella, M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. En *IEEE Conference on Evolutionary Computation*, páginas 622–627. IEEE Press, 1996.

[56] L. Gambardella, E. Taillard, G. Agazzi. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. En *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover, editores. McGraw-Hill, 1999.

[57] M. Garey, D. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

[58] T. Gaskell. Bases for vehicle fleet scheduling. *Operational Research Quarterly*, 18:281–295, 1967.

[59] M. Gendreau, A. Hertz, G. Laporte. New insertion and post optimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[60] M. Gendreau, A. Hertz, G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[61] M. Gendreau, A. Hertz, G. Laporte, M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 43(3):330–335, 1998.

[62] M. Gendreau, G. Laporte, J.-Y. Potvin. Vehicle Routing: Modern Heuristics. En *Local search in Combinatorial Optimization*, E. Aarts, J. Lenstra, editores, páginas 311–336. Wiley, Chichester, 1997.

[63] M. Gendreau, G. Laporte, J.-Y. Potvin. Metaheuristics for the Vehicle Routing Problem. En *The Vehicle Routing Problem*, P. Toth, D. Vigo, editores, páginas 129–154. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[64] B. Gillett, L. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22:340–349, 1974.

[65] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[66] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[67] F. Glover. Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Reporte técnico, Graduate School of Business and Administration, University of Colorado, June 1991.

[68] F. Glover. Scatter Search and Path Relinking. En *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover, editores. McGraw-Hill, 1999.

[69] F. Glover, M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[70] F. Glover, M. Laguna, R. Martí. Scatter Search and Path Relinking: advances and applications. En *Handbook of Metaheuristics*, F. Glover, G. Kochenberger, editores. Kluwer Academic Publishers, 2002.

[71] D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[72] B. Golden, T. Magnanti, H. Nguyen. Implementing vehicle routing algorithms. *Networks*, 7:113–148, 1977.

[73] B. Golden, A. Assad. *Vehicle routing: methods and studies*. North-Holland, Amsterdam, 1988.

[74] B. Golden, A. Assad, L. Levy, F. Gheysens. The fleet size and mix vehicle routing problem. *Computers and Operations Research*, 11(1):49 – 66, 1984.

[75] B. Golden, G. Laporte, E. Taillard. An adaptive memory heuristic for a class of vehicle routing problems with min-max objective. *Computers and Operations Research*, 24:445–452, 1997.

[76] B. Golden, E. Wasil, J. Kelly, I.-M. Chao. Metaheuristics in vehicle routing. En *Fleet management and logistics*, T. Crainic, G. Laporte, editores, páginas 33–56. Kluwer Academic Publishers, 1998.

[77] R. Hall. On the road to recovery. *ORMS Today*, 31:40–49, 2004.

[78] P. Hansen, N. Mladenović. Variable Neighborhood Search. En *Handbook of Metaheuristics*, F. Glover, G. Kochenberger, editores, páginas 145–184. Kluwer Academic Publishers, 2003.

[79] J. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, 1975.

[80] J. Hooker. Needed: an empirical science of algorithms. *Operations Research*, 42:201–212, 1994.

[81] J. Hooker. Testing heuristics: we have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[82] H. Hoos, T. Stützle. *Stochastic local search foundations and applications.* Elsevier / Morgan Kaufmann, 2004.

[83] D. Johnson, L. McGeoch. The Traveling Salesman Problem: a case study in local optimization. En *Local search in Combinatorial Optimization*, E. Aarts, J. Lenstra, editores, páginas 215–310. John Wiley and Sons, 1997.

[84] S. Kirkpatrick, C. Gelatt, M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[85] G. Laporte. The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[86] G. Laporte, G. Mercure, Y. Norbert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16:33–46, 1986.

[87] G. Laporte, G. Mercure, Y. Norbert. A branch-and-bound algorithm for a class of asymmetrical vehicle routing problems. *Journal of the Operational Research Society*, 43:469–481, 1992.

[88] G. Laporte, Y. Norbert, M. Desrochers. Optimal routing under capacity and distance restrictions. *Operations Research*, 33:1050–1073, 1985.

[89] G. Laporte, Y. Norbert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.

[90] G. Laporte, F. Semet. Classical heuristics for the capacitated VRP. En *The vehicle routing problem*, P. Toth, D. Vigo, editores, páginas 109–128. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[91] J. Lenstra, A. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

[92] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[93] J. Little, K. Murty, D. Sweeney, C. Karel. An algorithm for the tavelling salesman problem. *Operations Research*, 11:972–989, 1963.

[94] H. Lourenço, O. Martin, T. Stützle. Iterated Local Search. En *Handbook of Metaheuristics*, F. Glover, G. Kochenberger, editores, páginas 321–353. Kluwer Academic Publishers, 2002.

[95] D. Luenberger. *Linear and nonlinear optimization.* Addison-Wesley, 1984.

[96] S. Martello, P. Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley and Sons, 1990.

[97] D. Miller. A matching based exact algorithm for capacitated vehicle routing problems. *ORSA Journal on Computing*, 7:1–9, 1995.

[98] N. Mladenović, P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[99] R. Mole, S. Jameson. A sequential route-building algorithm employing a generalised savings criterion. *Operational Research Quarterly*, 27:503–511, 1976.

[100] G. Nemhauser, L. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[101] I. Or. Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking, 1976.

[102] C. Orloff. Route constrained fleet scheduling. *Transportation Science*, 10:149–168, 1976.

[103] I. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[104] R. Petch, S. Salhi. A multi-phase constructive heuristic for the vehicle routing problem with multiple trips. *Discrete Applied Mathematics*, 133:69–92, 2004.

[105] L. Pitsoulis, M. Resende. Greedy Randomized Adaptive Search Procedures. En *Handbook of Applied Optimization*, P. Pardalos, M. Resende, editores, páginas 178–183. Oxford University Press, 2002.

[106] J.-Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:339–370, 1996.

[107] J.-Y. Potvin, S. Bengio. The vehicle routing problem with time windows – part II: genetic search. *INFORMS Journal on Computing*, 8:165–172, 1996.

[108] J.-Y. Potvin, J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66:331–340, 1993.

[109] C. Rego, C. Roucariol. A parallel tabu search algorithm using ejection chains for the Vehicle Routing Problem. En *Meta-Heuristics: theory and applications*, I. Osman, J. Kelly, editores, páginas 661–675. Kluwer Academic Publishers, 1996.

[110] J. Renaud, F. Boctor, G. Laporte. A fast composite heuristic for the symmetric travelling salesman problem. *INFORMS Journal on Computing*, 8(2):134–143, 1996.

[111] J. Renaud, F. Boctor, G. Laporte. An improved petal heuristic for the vehicle routing problem. *Journal of Operational Research Society*, 47:329–336, 1996.

[112] Y. Rochat, E. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[113] D. Ryan, C. Hjorring, F. Glover. Extensions of the petal method for vehicle routing. *Journal of Operational Research Society*, 44:289–296, 1993.

[114] M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35(2):254–264, 1987.

[115] T. Stützle, H. Hoos. Improving the ant system: a detailed report on the MAX-MIN ant system. Reporte Técnico AIDA-96-12, 1996.

[116] E. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[117] E. Taillard. A heuristic column generation method for the heterogeneous VRP. *Operations Research – Recherche opérationnelle*, 33:1–14, 1999.

[118] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31:170–186, 1997.

[119] E. Taillard, L. Gambardella, M. Gendreau, J.-Y. Potvin. Adaptive memory programming: a unified view of metaheuristics. *European Journal of Operational Research*, 135:1–16, 2001.

[120] E. Taillard, G. Laporte, M. Gendreau. The vehicle routing problem with multiple use of vehicles. *Journal of the Operational Research Society*, 47:1065–1070, 1996.

[121] H. Tang, E. Miller-Hooks. A TABU search heuristic for the team orienteering problem. *Computers and Operations Research*, 32:1379–1407, 2005.

[122] C. Tarantilis. Solving the vehicle routing problem with adaptive memory programming methodology. *Computers and Operations Research, (in press).*

[123] C. Tarantilis, C. Kiranoudis. BoneRoute: An adaptive memory-based method for effective fleet management. *Annals of Operations Research*, 114:227–241, 2002.

[124] S. Thangiah. Vehicle Routing with Time Windows using Genetic Algorithms. En *Application handbook of Genetic Algorithms: new frontiers, volume II*, L. Chambers, editor, páginas 253–277. CRC Press, 1995.

[125] P. Toth, D. Vigo. Exact solution of the Vehicle Routing Problem. En *Fleet management and logistics*, T. Crainic, G. Laporte, editores, páginas 1–31. Kluwer Academic Publishers, 1998.

[126] P. Toth, D. Vigo. An overview of vehicle routing problems. En *The Vehicle Routing Problem*, P. Toth, D. Vigo, editores, Monographs on Discrete Mathematics and Applications, páginas 1–26. SIAM, 2002.

[127] P. Toth, D. Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

[128] A. Van Breedam. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, 86:480–490, 1995.

[129] T. Volgenat, R. Jonker. The symmetric traveling salesman problem and edge exchanges in minimal 1-tree. *European Journal of Operational Research*, páginas 394–403, 1983.

[130] S. Voß. Meta-heuristics: the state of the art. En *ECAI '00: Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers*, páginas 1–23. Springer-Verlag, 2001.

[131] C. Voudouris, E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[132] P. Wark, J. Holt. A repeated matching heuristic for the vehicle routing problem. *Journal of Operational Research Society*, 45:1156–1167, 1994.

[133] A. Wren. *Computers in transport planning and operation.* Ian Allan, 1971.

[134] A. Wren, A. Holliday. Computer scheduling of vehicles form one or more depots to a number of delivery points. *Operational Research Quarterly*, 23:333–344, 1972.

[135] J. Xu, J. Kelly. A network-flow based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30:379–393, 1996.

[136] P. Yellow. A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly*, 21:281–283, 1970.

[137] Q. Zhao, S. Wang, K. Lai, G. Xia. A vehicle routing problem with multiple use of vehicles. *Advanced Modeling and Optimization*, 4:21–40, 2002.

[138] K. Zhu. A new genetic algorithm for VRPTW. Presentado en IC-AI 2000, Las Vegas, USA, 2000.

Apéndice A

Metaheurísticas para problemas de optimización combinatoria

A.1. Introducción

En los últimos 15 años se ha desarrollado un conjunto de algoritmos aproximados que pretenden superar las limitaciones de las heurísticas más simples. Este tipo de métodos suelen combinar heurísticas básicas para lograr una exploración efectiva del espacio de soluciones. Se caracterizan mediante descripciones de alto nivel llamadas *metaheurísticas*. Si bien no existe una definición aceptada para el término, resulta adecuado a los efectos de este trabajo entender a una metaheurística como una estrategia de alto nivel para explorar espacios de búsqueda en problemas de optimización combinatoria [13].

Las metaheurísticas suelen estar diseñadas en base a ideas intuitivas sobre cómo obtener las mejores soluciones, más que en conceptos con bases teóricas que los sustenten. Esta característica permite obtener métodos más flexibles y fáciles de adaptar que los algoritmos de optimización clásicos. Como contrapartida, al utilizar metaheurísticas no se dispone de resultados teóricos de utilidad que garanticen la convergencia a soluciones óptimas. De todos modos, la experiencia sugiere que las metaheurísticas pueden resultar competitivas con otros enfoques, pues han permitido obtener soluciones de calidad aceptable para problemas complejos, en tiempos de ejecución moderados (o, al menos, controlables).

En este apéndice se da un breve relevamiento de algunas metaheurísticas para resolver un problema de optimización genérico, de la forma

$$\begin{aligned} \min \quad & f(x) \\ \text{s.a.} \quad & x \in X \end{aligned}$$

donde X es un conjunto discreto de soluciones factibles y $f : X \to \mathbb{R}$ es la función objetivo. Este relevamiento, más que ser exhaustivo y profundo, pretende dar un panorama

Paso 1 (inicialización).
Construir una solución inicial s_0 y hacer $k \leftarrow 0$.

Paso 2 (búsqueda local).
Determinar $s_{k+1} \in N(s_k)$ tal que $f(s_{k+1}) < f(s_k)$.
Si existe tal s_{k+1}, hacer $k \leftarrow k+1$ y repetir el paso 2.
Si no, devolver s_k y terminar.

Figura A.1: Búsqueda local de descenso.

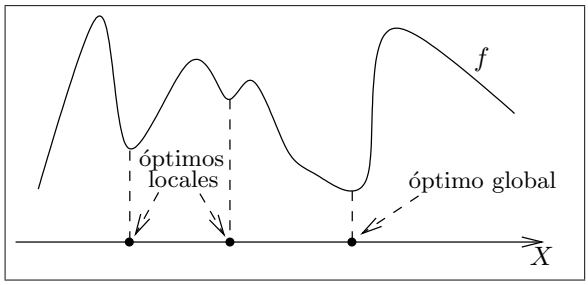


Figura A.2: Óptimos locales y globales.

general de los aspectos más relevantes y de las decisiones más importantes a la hora de diseñar métodos de este tipo. Cada una de las metheurísticas presentadas admite numerosas variantes y, además, es posible combinar dos o más técnicas obteniendo metaheurísticas híbridas.

A.2. Búsqueda local

Un esquema de búsqueda local básica requiere definir, para cada solución s, un conjunto de soluciones $N(s)$ llamado *vecindad* o *conjunto de soluciones vecinas*. Se llama *movida* a la operación necesaria para transformar una solución s en una solución vecina. Tipicamente, una movida implica la realización de un cambio pequeño en la estructura de la solución.

El seudocódigo de un algoritmo de búsqueda local de descenso se muestra en la Figura A.1. Se parte de una solución inicial s_0 y en cada iteración k, se selecciona una solución $s_{k+1} \in N(s_k)$ tal que $f(s_{k+1}) < f(s_k)$. Es decir, se busca una solución de menor costo en la vecindad. La secuencia de soluciones visitadas por el algoritmo $(s_0, s_1, \ldots)$ define una trayectoria en el espacio de soluciones.

En caso de que todas las soluciones vecinas tengan un costo mayor que s_k, el algoritmo termina y la solución s_k es un óptimo local (ver Figura A.2) respecto a la definición de vecindad. Es importante notar que, en el contexto de estos algoritmos, la optimalidad local es un concepto que depende de la definición de vecindad. En efecto, es posible definir estructuras de vecindad N_1 y N_2 sobre el mismo problema, de modo que los óptimos locales respecto a N_1 no lo sean respecto a N_2.

Paso 1 (inicialización).
Construir una solución inicial s_0 y hacer $s^{\text{best}} \leftarrow s_0$.
Inicializar T. Hacer $k \leftarrow 0$.

Paso 2 (búsqueda local).
Seleccionar aleatoriamente $s \in N(s_k)$.
Si $f(s) < f(s_k)$, ir al paso 3.
Si no, con probabilidad $p(s, s_k, T)$ ir al paso 3 y si no repetir el paso 2.

Paso 3 (actualización).
Si $f(s) < f(s^{\text{best}})$, hacer $s^{\text{best}} \leftarrow s$.
Hacer $s_{k+1} \leftarrow s$ y $k \leftarrow k + 1$.

Paso 4 (terminación).
Si $k < K^{\max}$, actualizar T e ir al paso 2.
Si no, devolver s^{best} y terminar.

Figura A.3: Simulated Annealing.

A.3. Simulated Annealing

La metaheurística *Simulated Annealing* (SA) fue propuesta por Kirkpatrick [84] y consiste en una búsqueda local dotada de un mecanismo para escapar de los óptimos locales. Es considerada una de las metaheurísticas más antiguas y, como muchas otras, está inspirada en un fenómeno de la naturaleza. La idea está basada un procedimiento utilizado para enfriar ciertos tipos de vidrio, en el cual el enfriamiento debe hacerse de manera lenta para obtener configuraciones moleculares más resistentes.

La idea principal en este método es aceptar movidas que deterioran el valor de la función objetivo. Una descripción de alto nivel se muestra en la Figura A.3. En la iteración k se selecciona aleatoriamente una solución $s \in N(s_k)$. Si s mejora s_k en términos del valor objetivo, se acepta como nueva solución. En caso que $f(s_k) \leq f(s)$, la decisión de aceptarla o no se realiza de manera no determinística con probabilidad $p(s, s_k, T)$ donde T es un valor, llamado *temperatura*, que decrece a lo largo de la ejecución del algoritmo. La probabilidad de aceptación suele definirse como

$$p(s, s_k, T) = \exp\left(-\frac{f(s) - f(s_k)}{T}\right).$$

Las soluciones con valores más próximos a $f(s_k)$ reciben las mayores probabilidades de aceptación. La temperatura se utiliza para regular la exigencia en el criterio de aceptación: es más probable aceptar una solución para valores altos de T. El valor de T es modificado durante la ejecución del algoritmo, usualmente en forma decreciente.

El hecho de que en el paso 2 las soluciones vecinas se obtengan mediante un sorteo aleatorio, hace innecesario recorrer toda la vecindad en cada paso. Debido a esto, los

Paso 1 (inicialización).
Construir una solución inicial s y hacer $s^{\text{best}} \leftarrow s$.
Hacer $k \leftarrow 1$.

Paso 2 (agitado).
Si $k > K$, devolver s^{best} y terminar.
Si no, seleccionar aleatoriamente una solución $s' \in N_k(s)$.

Paso 3 (búsqueda local).
Realizar una búsqueda local (no necesariamente sobre $N_1, N_2, \ldots, N_K$).
Sea s'' el óptimo local obtenido.

Paso 4 (actualización).
Si $f(s'') < f(s^{\text{best}})$, hacer $s^{\text{best}} \leftarrow s''$.

Paso 5 (cambio de vecindad).
Si $f(s'') < f(s)$, hacer $s \leftarrow s''$, $k \leftarrow 1$ e ir al paso 2.
Si no, hacer $k \leftarrow k + 1$ e ir al paso 2.

Figura A.4: Variable Neighborhood Search.

algoritmos basados en SA son simples de implementar y con ellos pueden lograrse bajos tiempos de ejecución. Por otro lado, esa misma característica provoca que este método suela ser superado por otras estrategias que realizan búsquedas más intensas.

A.4. Tabu Search

La metaheurística *Tabu Search* (TS) fue propuesta por Glover [66] y es otro ejemplo de algoritmos de búsqueda local en los que se aceptan movidas que deterioran la solución como manera de evitar la convergencia en óptimos locales. Dado que TS es una de las técnicas utilizadas en este trabajo, los detalles sobre este método pueden consultarse en el capítulo 3.4.

A.5. Variable Neighborhood Search

La metaheurística *Variable Neighborhood Search* (VNS) [98, 78], explota el hecho de que la optimalidad local es un concepto que depende de la definición de la vecindad, para continuar la búsqueda luego de encontrar un óptimo local. Se utilizan K estructuras de vecindad $N_1, N_2, \ldots, N_K$, sobre las que se varía la búsqueda. Un seudocódigo de esta metaheurística se da en la Gigura A.4.

Las diferentes vecindades son utilizadas para generar soluciones iniciales para un algoritmo de búsqueda local. Cuando la búsqueda del paso 3 encuentra un óptimo local s'', se busca una solución vecina de s'' respecto una de las estructuras $N_1, \ldots, N_K$. Con ese

Paso 1 (inicialización).
Hacer $z^{\text{best}} \leftarrow \infty$.

Paso 2 (construcción de la solución).

a. Crear una solución parcial s vacía.

b. Construir la lista de candidatos RCL.

c. Seleccionar probabilísticamente un elemento de RCL y agregarlo a s.

d. Si s es una solución completa, ir al paso 3. Si no, ir al paso 2.b.

Paso 3 (búsqueda local).
Aplicar una búsqueda local partiendo de s. Sea s^* la solución obtenida.

Paso 4 (actualización).
Si $f(s^*) < z^{\text{best}}$, hacer $z^{\text{best}} \leftarrow f(s^*)$ y $s^{\text{best}} \leftarrow s^*$.

Paso 5 (terminación).
Si $k < K^{\max}$ ir al paso 2.
Si no, devolver s^{best} y terminar.

Figura A.5: GRASP.

cambio se pretende obtener una solución que permita reanudar la búsqueda local en una región cercana a la que se venía explorando.

A.6. GRASP

La metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) fue propuesta por Feo y Resende [47] y consiste en la ejecución repetida de dos fases: construcción de una solución y búsqueda local.

Para la construcción de la solución suele operarse iterativamente, utilizando una estrategia ávida aleatorizada. Se comienza con una solución vacía. En cada paso de la construcción se selecciona una lista de L elementos candidatos para insertar en la solución. De esta lista de candidatos se selecciona uno según una regla probabilística y se lo agrega a la solución. El proceso se repite hasta construir una solución completa, sobre la que se aplica la búsqueda local. En la Figura A.5 se da la estructura de este método.

Entre las características más relevantes de este algoritmo se encuentran su simplicidad y el hecho que su comportamiento depende de pocos parámetros (la cantidad de iteraciones y el largo de la lista de candidatos). Por otro lado, dado que cada iteración es independiente de las anteriores, en general es posible obtener mejores soluciones utilizando algoritmos que incorporan alguna forma de memoria.

A.7. Algoritmos Genéticos

En los *Algoritmos Genéticos* (AG) [79, 39] se utilizan ideas de la evolución natural de los seres vivos para resolver problemas de optimización y búsqueda. En general se trabaja sobre una representación de las soluciones en algún esquema de codificación (por ejemplo, vectores, matrices o árboles). A diferencia de las metaheurísticas anteriores, en cada iteración no se opera sobre una única solución. El algoritmo trabaja sobre una *población* P de soluciones codificadas, llamadas individuos. Para cada individuo $i \in P$ se define una *función de fitness* o aptitud, que denotaremos $F(i)$, de modo que cuánto mayor es el fitness de un individuo, mejor es la solución que éste representa[1].

En cada iteración se aplican ciertos operadores evolutivos que combinan y modifican a los individuos de la población. En el esquema más simple se opera en tres fases: selección, cruzamiento y mutación. El operador de *selección* se encarga de elegir algunos individuos de la población que tendrán la posibilidad de reproducirse (usualmente se busca incluir en esta selección a algunos de los individuos con mejores valores del fitness). Luego se aplica repetidas veces el operador de *cruzamiento*, que toma dos individuos (llamados padres) y los combina para generar dos individuos (llamados hijos). Finalmente, el operador de *mutación* suele aplicarse con baja probabilidad para realizar pequeñas modificaciones sobre los individuos. Los operadores de cruzamiento y mutación trabajan sobre los individuos, es decir, sobre la codificación de las soluciones. El algoritmo se ejecuta una cantidad prefijada de iteraciones $K^{\max}$ y a cada una de las poblaciones sucesivas se le llama generación. Un esquema de este tipo de algoritmos se presenta en la Figura A.6.

A.8. Colonias de Hormigas

Los *Ant Systems* (AS) [30] se inspiran en la estrategia utilizada por las colonias de hormigas para buscar alimentos. Cuando una hormiga encuentra un camino hacia una fuente de alimento, deposita en el trayecto una sustancia llamada *feromona*. La cantidad de feromona depositada depende de la longitud del camino y de la calidad del alimento encontrado. Si una hormiga no detecta la presencia de feromona se mueve aleatoriamente; pero si percibe dicha sustancia, decidirá con alta probabilidad moverse por los trayectos con más cantidad, lo que a su vez provocará una aumento de la feromona depositada en esa zona. De este proceso emerge una comportamiento denominado autocatalítico: cuanto más hormigas sigan cierto trayecto, más atractivo éste se vuelve para ellas.

En los AS se trabaja sobre un grafo $G = (C, L)$ en el cual los nodos representan los componentes de las soluciones y los arcos representan las conexiones posibles, de modo que cada camino se corresponde con una secuencia de pasos que construyen una solución.

[1] En algunos problemas de minimización, se define un fitness proporcional a la función objetivo, de modo que en esos casos se desea *minimizar* el fitness.

Paso 1 (inicialización).
Construir una población inicial P_0 y hacer $k \leftarrow 0$.
Hacer $s^{\text{best}} \leftarrow \arg\min_{i \in P_0} f(i)$.

Paso 2 (selección).
Seleccionar un conjunto de individuos $P'_k \subseteq P_k$.

Paso 3 (cruzamiento).
Aplicar el operador de cruzamiento sobre individuos de P'_k.

Paso 4 (mutación).
Aplicar el operador de mutación sobre individuos de P'_k.

Paso 5 (actualización).
Hacer $s_k^{\text{best}} \leftarrow \arg\min_{i \in P_k} f(i)$.
Si $f(s_k^{\text{best}}) < f(s^{\text{best}})$, hacer $s^{\text{best}} \leftarrow s_k^{\text{best}}$.

Paso 6 (terminación).
Si $k < K^{\max}$, hacer $P_{k+1} \leftarrow P'_k$, $k \leftarrow k+1$ e ir al paso 2.
Si no, devolver s^{best}.

Figura A.6: Algoritmo Genético.

La idea consiste en simular el comportamiento de una colonia de M hormigas moviéndose por G (es decir, construyendo soluciones). Cada arco de $(i,j) \in L$ tiene asociado una medida η_{ij} que representa el costo de incorporar el componente j habiendo incorporado el i previamente, y una cantidad de feromona τ_{ij} que resume la información histórica obtenida por las hormigas en iteraciones anteriores.

En cada iteración, las hormigas se mueven en G mediante reglas probabilísticas que combinan los valores de η_{ij} y τ_{ij}. Luego, se actualizan los valores de τ_{ij}, buscando que los caminos recorridos por las hormigas que obtuvieron las mejores soluciones sean los que incrementen en mayor medida la cantidad de feromona depositada. La Figura A.7 muestra un seudocódigo de este método.

Paso 1 (inicialización).
Inicializar τ_{ij}^0. Hacer $k \leftarrow 0$ y $z^{\text{best}} \leftarrow \infty$.

Paso 2 (construcción de soluciones).
Colocar M hormigas en el grafo.
Calcular las probabilidades de transición en función de τ_{ij}^k y η_{ij}.
Construir una solución para cada hormiga.
Sean $S^k = \{s_1^k, \ldots, s_M^k\}$ dichas soluciones y $\hat{s}^k \leftarrow \arg\min_{s \in S^k} f(s)$.

Paso 3 (actualización).
Si $f(\hat{s}^k) < z^{\text{best}}$, hacer $s^{\text{best}} \leftarrow \hat{s}^k$ y $z^{\text{best}} \leftarrow f(\hat{s}^k)$.

Paso 4 (terminación).
Si $k \geq K^{\max}$, devolver s^{best} y terminar.

Paso 5 (actualización de las feromonas).
Calcular τ_{ij}^{k+1} utilizando las soluciones de S^k y los valores de τ_{ij}^k.
Hacer $k \leftarrow k + 1$ e ir al paso 2.

Figura A.7: Algoritmo de Hormigas.

Apéndice B

Problemas relacionados

B.1. Introducción

En este apéndice se dan las definiciones de algunos problemas de optimización combinatoria relacionados de alguna manera con el problema estudiado en la tesis.

B.2. El Set Partitioning Problem

Dado un conjunto finito S y subconjuntos $S_1, \ldots, S_n$ de S, donde el conjunto S_i tiene asociado un costo c_i, en el *Set Partitioning Problem* (SPP) [100] se desea encontrar $I \subset [1, n]$ de modo que $\bigcup_{i \in I} S_i = S$ y $\sum_{i \in I} c_i$ sea mínimo. Es decir, seleccionar algunos de los subconjuntos de modo que sean una partición de S y se minimice el costo total.

Definiendo para cada $i \in 1..n$ y $j \in S$, un parámetro a_{ij} que tome el valor 1 si $j \in S_i$ y 0 en caso contrario, el problema puede formularse como:

$$\min \sum_{i=1}^{n} c_i x_i \tag{B.1}$$

$$\text{s.a.} \sum_{i=1}^{n} a_{ij} x_i = 1 \qquad \forall\, j \in S \tag{B.2}$$

$$x_i \in \{0, 1\} \qquad \forall\, i = 1, \ldots, m$$

Las variables x_i indican si el subconjunto i es seleccionado o no. La función objetivo B.1 mide el costo de cada selección. El hecho de que los conjuntos seleccionados formen una partición se impone en B.2.

El SPP es un problema de la clase NP-Hard [57].

B.3. El Bin Packing Problem

En el *Bin Packing Problem* (BPP) [100] existe un conjunto de items $I = \{1, \ldots, n\}$, cada uno de los cuales tiene asociado un peso w_i. Se dispone de un conjunto de infinitos recipientes idénticos de capacidad B. El problema consiste en determinar la mínima cantidad de recipientes necesarios para almacenar los items de modo que el peso total de los items almacenados en un mismo recipiente no supere su capacidad.

Siendo $J = \{1, \ldots, n\}$ un conjunto con suficiente cantidad de recipientes (por ejemplo, con $n = m$), el problema puede formularse como:

$$\min \sum_{j \in J} y_j \tag{B.3}$$

$$\text{s.a.} \sum_{i \in I} w_i x_{ij} \leq B y_j \qquad \forall\, j \in J \tag{B.4}$$

$$\sum_{j \in J} x_{ij} = 1 \qquad \forall\, i \in I \tag{B.5}$$

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

$$y_j \in \{0, 1\} \qquad \forall\, j \in J$$

Las variables x_{ij} indican si el item i es asignado al recipiente j, mientras que las y_j establecen si el recipiente j se utiliza o no en la solución. El objetivo consiste en minimizar la cantidad de recipientes utilizados (B.3). En (B.4) se impone que el peso de los items asignados a un mismo recipiente no supere su capacidad y, finalmente, en (B.5) se exige que cada item sean asignado a exactamente un recipiente.

El BPP pertenece a la clase de problemas NP-Hard [57]. Este problema también puede formularse como un problema de decisión, en el que, dados los datos del problema, se busca determinar si existe una solución que respete las restricciones del problema y utilizando a lo sumo U recipientes. Esta versión es un problema NP-Completo [57].

Apéndice C

Instancias de prueba

En este apéndice se detallan los datos de cada instancia del VRP utilizada como base para la construcción de las instancias de prueba del VPRMT. Para cada una de ellas, se provee una tabla con las coordenadas de cada nodo (x_i,y_i), las demandas de los clientes (d_i) y la capacidad de cada vehículo Q. Además, se muestra una gráfica con las posiciones de los clientes y el depoósito.

Nodos de la instancia CMT-1

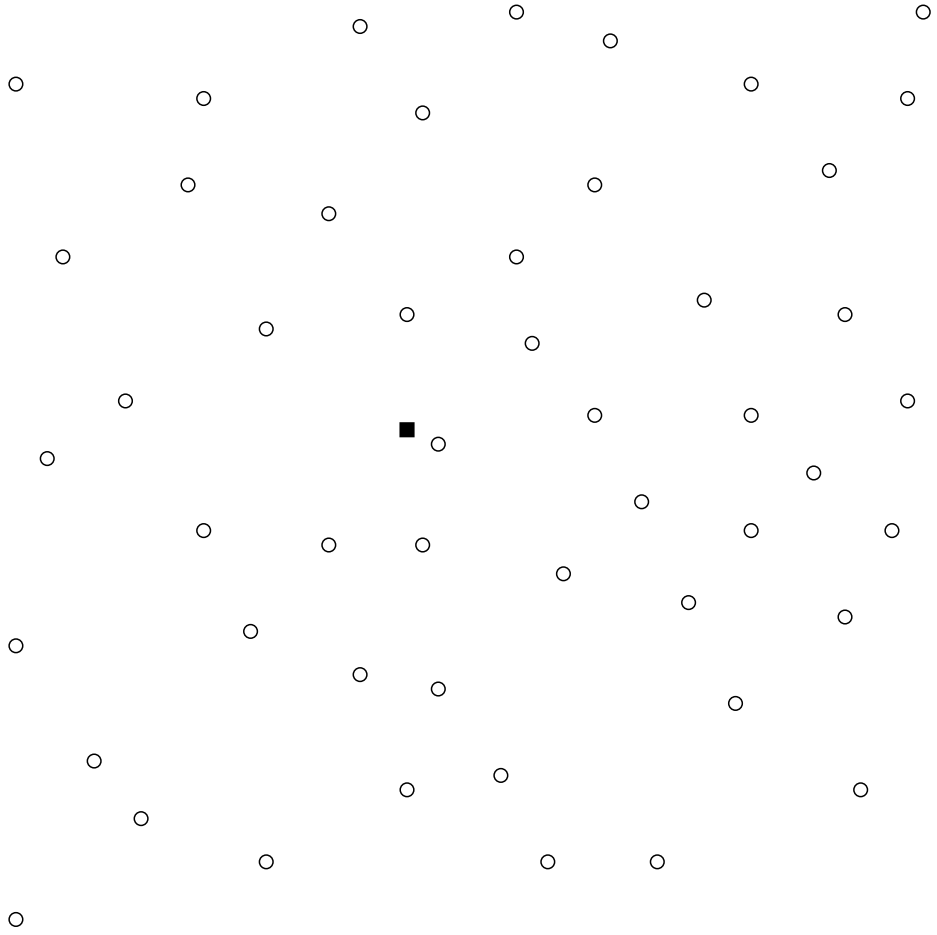
La instancia CMT-1 fue propuesta por Christofides et al. [28], tiene 50 nodos y $Q = 160$.

i	x_i	y_i	d_i
0	30	40	–
1	37	52	7
2	49	49	30
3	52	64	16
4	20	26	9
5	40	30	21
6	21	47	15
7	17	63	19
8	31	62	23
9	52	33	11
10	51	21	5
11	42	41	19
12	31	32	29
13	5	25	23
14	12	42	21
15	36	16	10
16	52	41	15

i	x_i	y_i	d_i
17	27	23	3
18	17	33	41
19	13	13	9
20	57	58	28
21	62	42	8
22	42	57	8
23	16	57	16
24	8	52	10
25	7	38	28
26	27	68	7
27	30	48	15
28	43	67	14
29	58	48	6
30	58	27	19
31	37	69	11
32	38	46	12
33	46	10	23

i	x_i	y_i	d_i
34	61	33	26
35	62	63	17
36	63	69	6
37	32	22	9
38	45	35	15
39	59	15	14
40	5	6	7
41	10	17	27
42	21	10	13
43	5	64	11
44	30	15	16
45	39	10	10
46	32	39	5
47	25	32	25
48	25	55	17
49	48	28	18
50	56	37	10

Gráfico de la instancia CMT-1



Nodos de la instancia CMT-2

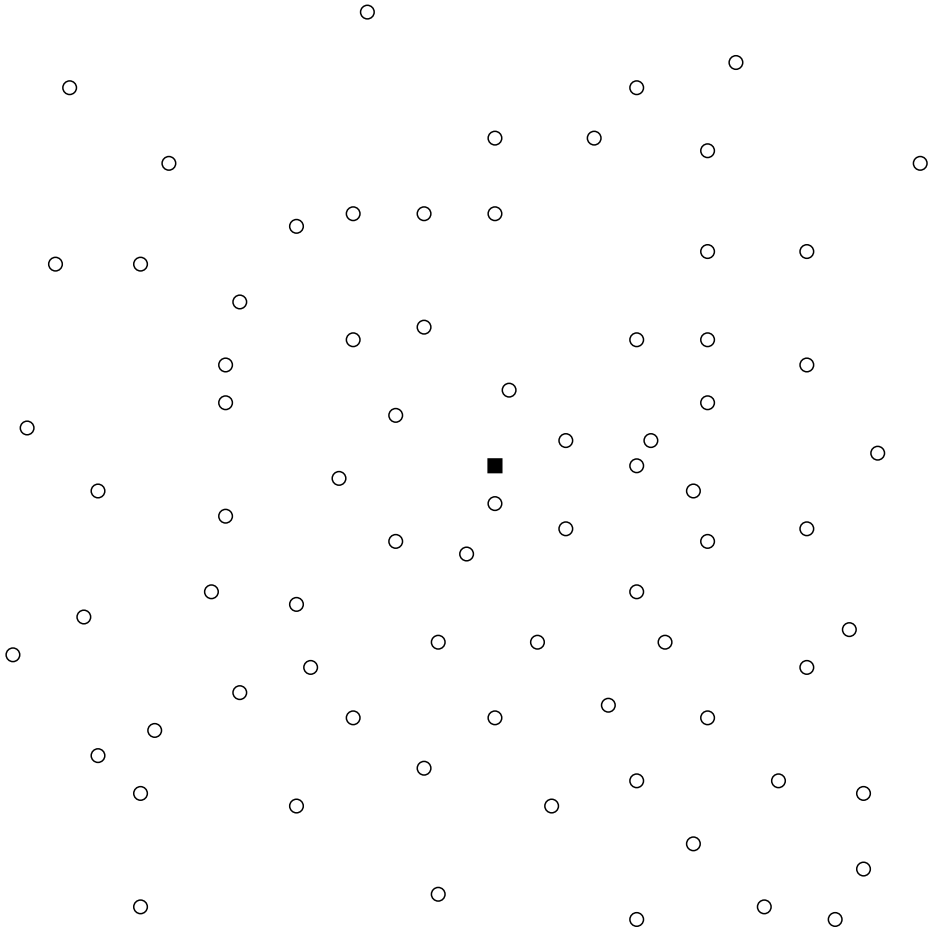
La instancia CMT-2 fue propuesta por Christofides et al. [28], tiene 75 nodos y $Q = 140$.

i	x_i	y_i	d_i
0	40	40	–
1	22	22	18
2	36	26	26
3	21	45	11
4	45	35	30
5	55	20	21
6	33	34	19
7	50	50	15
8	55	45	16
9	26	59	29
10	40	66	26
11	55	65	37
12	35	51	16
13	62	35	12
14	62	57	31
15	62	24	8
16	21	36	19
17	33	44	20
18	9	56	13
19	62	48	15
20	66	14	22
21	44	13	28
22	26	13	12
23	11	28	6
24	7	43	27
25	17	64	14

i	x_i	y_i	d_i
26	41	46	18
27	55	34	17
28	35	16	29
29	52	26	13
30	43	26	22
31	31	76	25
32	22	53	28
33	26	29	27
34	50	40	19
35	55	50	10
36	54	10	12
37	60	15	14
38	47	66	24
39	30	60	16
40	30	50	33
41	12	17	15
42	15	14	11
43	16	19	18
44	21	48	17
45	50	30	21
46	51	42	27
47	50	15	19
48	48	21	20
49	12	38	5
50	15	56	22

i	x_i	y_i	d_i
51	29	39	12
52	54	38	19
53	55	57	22
54	67	41	16
55	10	70	7
56	6	25	26
57	65	27	14
58	40	60	21
59	70	64	24
60	64	4	13
61	36	6	15
62	30	20	18
63	20	30	11
64	15	5	28
65	50	70	9
66	57	72	37
67	45	42	30
68	38	33	10
69	50	4	8
70	66	8	11
71	59	5	3
72	35	60	1
73	27	24	6
74	40	20	10
75	40	37	20

Gráfico de la instancia CMT-2



Nodos de la instancia CMT-3

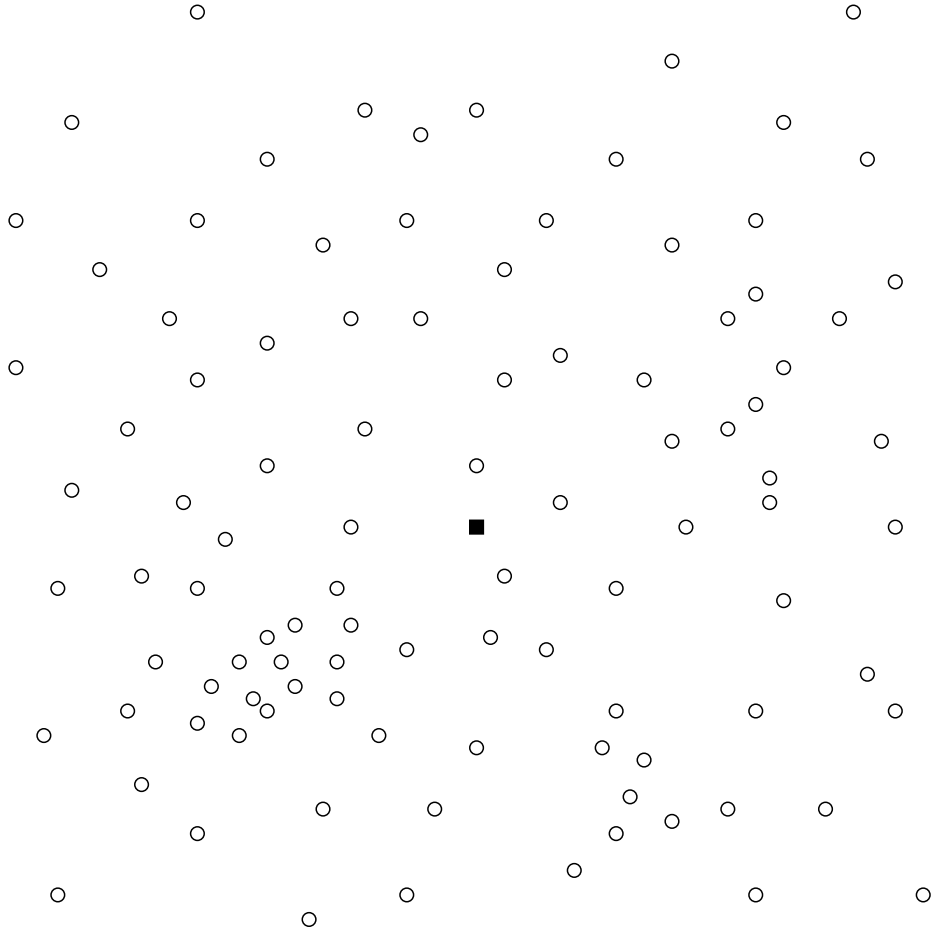
La instancia CMT-3 fue propuesta por Christofides et al. [28], tiene 100 nodos y $Q = 200$.

i	x_i	y_i	d_i
0	35	35	–
1	41	49	10
2	35	17	7
3	55	45	13
4	55	20	19
5	15	30	26
6	25	30	3
7	20	50	5
8	10	43	9
9	55	60	16
10	30	60	16
11	20	65	12
12	50	35	19
13	30	25	23
14	15	10	20
15	30	5	8
16	10	20	19
17	5	30	2
18	20	40	12
19	15	60	17
20	45	65	9
21	45	20	11
22	45	10	18
23	55	5	29
24	65	35	3
25	65	20	6
26	45	30	17
27	35	40	16
28	41	37	16
29	64	42	9
30	40	60	21
31	31	52	27
32	35	69	23
33	53	52	11

i	x_i	y_i	d_i
34	65	55	14
35	63	65	8
36	2	60	5
37	20	20	8
38	5	5	16
39	60	12	31
40	40	25	9
41	42	7	5
42	24	12	5
43	23	3	7
44	11	14	18
45	6	38	16
46	2	48	1
47	8	56	27
48	13	52	36
49	6	68	30
50	47	47	13
51	49	58	10
52	27	43	9
53	37	31	14
54	57	29	18
55	63	23	2
56	53	12	6
57	32	12	7
58	36	26	18
59	21	24	28
60	17	34	3
61	12	24	13
62	24	58	19
63	27	69	10
64	15	77	9
65	62	77	20
66	49	73	25
67	67	5	25

i	x_i	y_i	d_i
68	56	39	36
69	37	47	6
70	37	56	5
71	57	68	15
72	47	16	25
73	44	17	9
74	46	13	8
75	49	11	18
76	49	42	13
77	53	43	14
78	61	52	3
79	57	48	23
80	56	37	6
81	55	54	26
82	15	47	16
83	14	37	11
84	11	31	7
85	16	22	41
86	4	18	35
87	28	18	26
88	26	52	9
89	26	35	15
90	31	67	3
91	15	19	1
92	22	22	2
93	18	24	22
94	26	27	27
95	25	24	20
96	22	27	11
97	25	21	12
98	19	21	10
99	20	26	9
100	18	18	17

Gráfico de la instancia CMT-3



Nodos de la instancia CMT-4

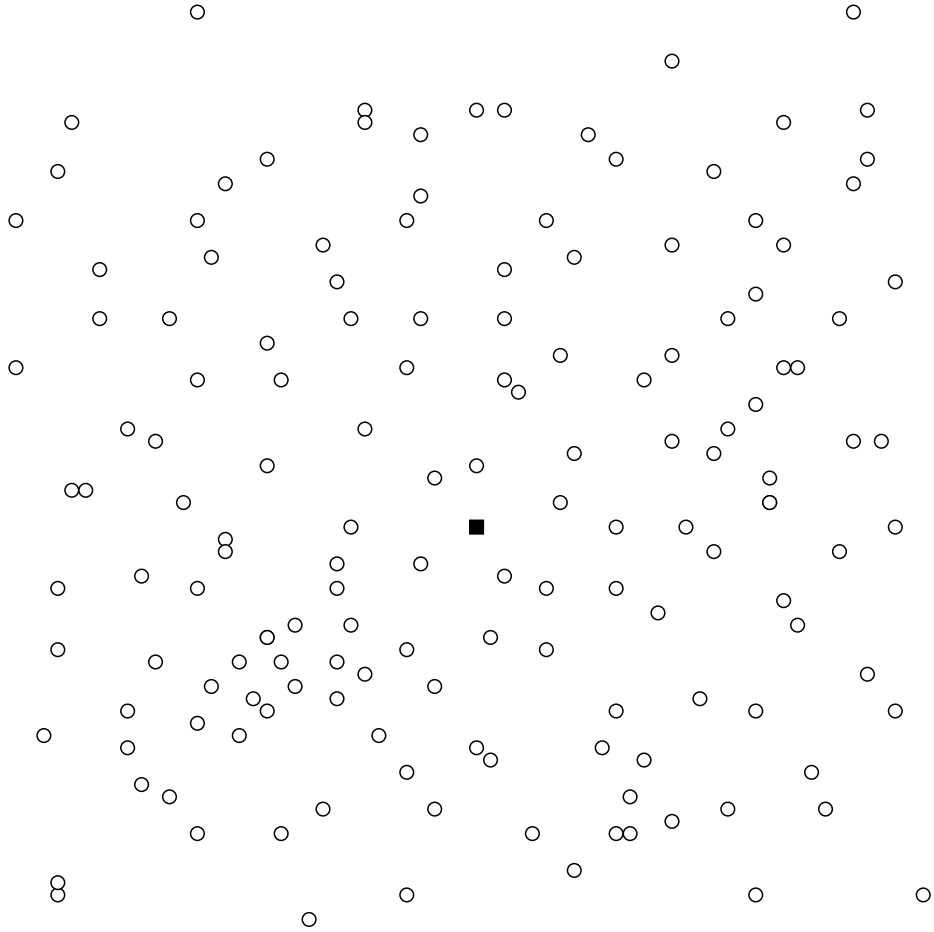
La instancia CMT-4 fue propuesta por Christofides et al. [28], tiene 150 nodos y $Q = 200$.

i	x_i	y_i	d_i
0	35	35	–
1	41	49	10
2	35	17	7
3	55	45	13
4	55	20	19
5	15	30	26
6	25	30	3
7	20	50	5
8	10	43	9
9	55	60	16
10	30	60	16
11	20	65	12
12	50	35	19
13	30	25	23
14	15	10	20
15	30	5	8
16	10	20	19
17	5	30	2
18	20	40	12
19	15	60	17
20	45	65	9
21	45	20	11
22	45	10	18
23	55	5	29
24	65	35	3
25	65	20	6
26	45	30	17
27	35	40	16
28	41	37	16
29	64	42	9
30	40	60	21
31	31	52	27
32	35	69	23
33	53	52	11
34	65	55	14
35	63	65	8
36	2	60	5
37	20	20	8
38	5	5	16
39	60	12	31
40	40	25	9
41	42	7	5
42	24	12	5
43	23	3	7
44	11	14	18
45	6	38	16
46	2	48	1
47	8	56	27
48	13	52	36
49	6	68	30
50	47	47	13

i	x_i	y_i	d_i
51	49	58	10
52	27	43	9
53	37	31	14
54	57	29	18
55	63	23	2
56	53	12	6
57	32	12	7
58	36	26	18
59	21	24	28
60	17	34	3
61	12	24	13
62	24	58	19
63	27	69	10
64	15	77	9
65	62	77	20
66	49	73	25
67	67	5	25
68	56	39	36
69	37	47	6
70	37	56	5
71	57	68	15
72	47	16	25
73	44	17	9
74	46	13	8
75	49	11	18
76	49	42	13
77	53	43	14
78	61	52	3
79	57	48	23
80	56	37	6
81	55	54	26
82	15	47	16
83	14	37	11
84	11	31	7
85	16	22	41
86	4	18	35
87	28	18	26
88	26	52	9
89	26	35	15
90	31	67	3
91	15	19	1
92	22	22	2
93	18	24	22
94	26	27	27
95	25	24	20
96	22	27	11
97	25	21	12
98	19	21	10
99	20	26	9
100	18	18	17

i	x_i	y_i	d_i
101	37	52	7
102	49	49	30
103	52	64	16
104	20	26	9
105	40	30	21
106	21	47	15
107	17	63	19
108	31	62	23
109	52	33	11
110	51	21	5
111	42	41	19
112	31	32	29
113	5	25	23
114	12	42	21
115	36	16	10
116	52	41	15
117	27	23	3
118	17	33	41
119	13	13	9
120	57	58	28
121	62	42	8
122	42	57	8
123	16	57	16
124	8	52	10
125	7	38	28
126	27	68	7
127	30	48	15
128	43	67	14
129	58	48	6
130	58	27	19
131	37	69	11
132	38	46	12
133	46	10	23
134	61	33	26
135	62	63	17
136	63	69	6
137	32	22	9
138	45	35	15
139	59	15	14
140	5	6	7
141	10	17	27
142	21	10	13
143	5	64	11
144	30	15	16
145	39	10	10
146	32	39	5
147	25	32	25
148	25	55	17
149	48	28	18
150	56	37	10

Gráfico de la instancia CMT-4



Nodos de la instancia CMT-5

La instancia CMT-5 fue propuesta por Christofides et al. [28], tiene 200 nodos y $Q = 200$.

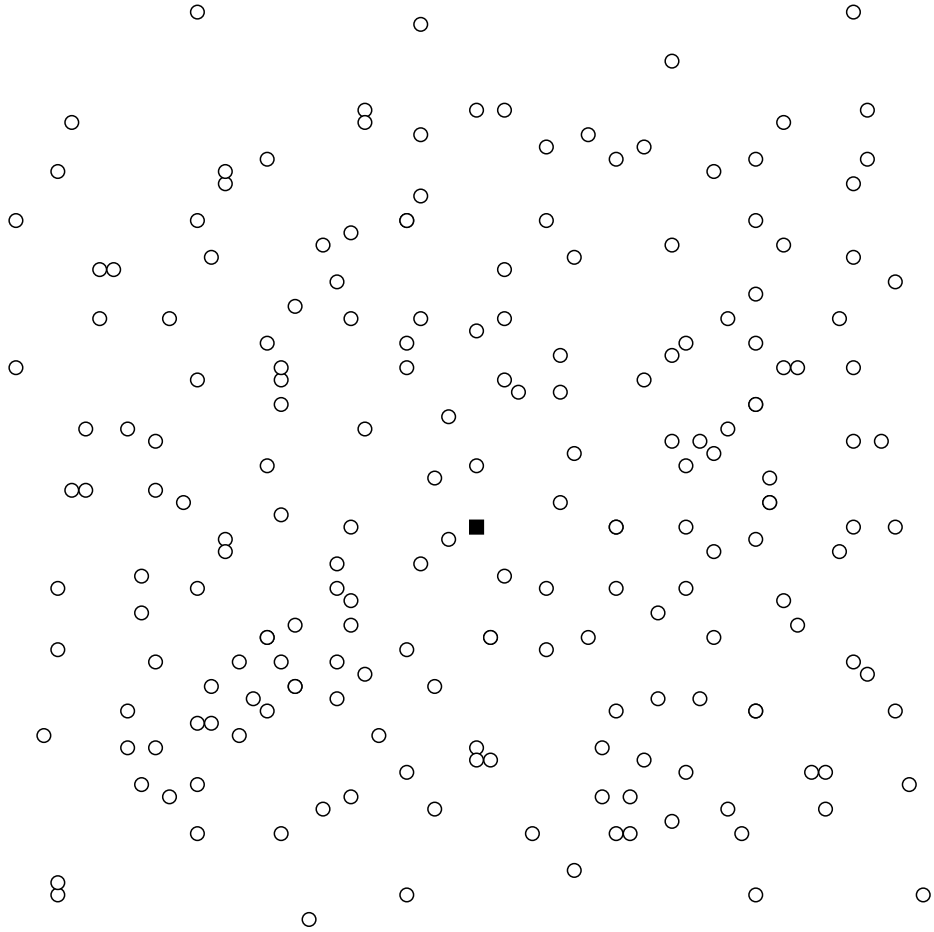
i	x_i	y_i	d_i
0	35	35	–
1	41	49	10
2	35	17	7
3	55	45	13
4	55	20	19
5	15	30	26
6	25	30	3
7	20	50	5
8	10	43	9
9	55	60	16
10	30	60	16
11	20	65	12
12	50	35	19
13	30	25	23
14	15	10	20
15	30	5	8
16	10	20	19
17	5	30	2
18	20	40	12
19	15	60	17
20	45	65	9
21	45	20	11
22	45	10	18
23	55	5	29
24	65	35	3
25	65	20	6
26	45	30	17
27	35	40	16
28	41	37	16
29	64	42	9
30	40	60	21
31	31	52	27
32	35	69	23
33	53	52	11
34	65	55	14
35	63	65	8
36	2	60	5
37	20	20	8
38	5	5	16
39	60	12	31
40	40	25	9
41	42	7	5
42	24	12	5
43	23	3	7
44	11	14	18
45	6	38	16
46	2	48	1
47	8	56	27
48	13	52	36
49	6	68	30
50	47	47	13

i	x_i	y_i	d_i
51	49	58	10
52	27	43	9
53	37	31	14
54	57	29	18
55	63	23	2
56	53	12	6
57	32	12	7
58	36	26	18
59	21	24	28
60	17	34	3
61	12	24	13
62	24	58	19
63	27	69	10
64	15	77	9
65	62	77	20
66	49	73	25
67	67	5	25
68	56	39	36
69	37	47	6
70	37	56	5
71	57	68	15
72	47	16	25
73	44	17	9
74	46	13	8
75	49	11	18
76	49	42	13
77	53	43	14
78	61	52	3
79	57	48	23
80	56	37	6
81	55	54	26
82	15	47	16
83	14	37	11
84	11	31	7
85	16	22	41
86	4	18	35
87	28	18	26
88	26	52	9
89	26	35	15
90	31	67	3
91	15	19	1
92	22	22	2
93	18	24	22
94	26	27	27
95	25	24	20
96	22	27	11
97	25	21	12
98	19	21	10
99	20	26	9
100	18	18	17

i	x_i	y_i	d_i
101	37	52	7
102	49	49	30
103	52	64	16
104	20	26	9
105	40	30	21
106	21	47	15
107	17	63	19
108	31	62	23
109	52	33	11
110	51	21	5
111	42	41	19
112	31	32	29
113	5	25	23
114	12	42	21
115	36	16	10
116	52	41	15
117	27	23	3
118	17	33	41
119	13	13	9
120	57	58	28
121	62	42	8
122	42	57	8
123	16	57	16
124	8	52	10
125	7	38	28
126	27	68	7
127	30	48	15
128	43	67	14
129	58	48	6
130	58	27	19
131	37	69	11
132	38	46	12
133	46	10	23
134	61	33	26
135	62	63	17
136	63	69	6
137	32	22	9
138	45	35	15
139	59	15	14
140	5	6	7
141	10	17	27
142	21	10	13
143	5	64	11
144	30	15	16
145	39	10	10
146	32	39	5
147	25	32	25
148	25	55	17
149	48	28	18
150	56	37	10

i	x_i	y_i	d_i
151	22	22	18
152	36	26	26
153	21	45	11
154	45	35	30
155	55	20	21
156	33	34	19
157	50	50	15
158	55	45	16
159	26	59	29
160	40	66	26
161	55	65	37
162	35	51	16
163	62	35	12
164	62	57	31
165	62	24	8
166	21	36	19
167	33	44	20
168	9	56	13
169	62	48	15
170	66	14	22
171	44	13	28
172	26	13	12
173	11	28	6
174	7	43	27
175	17	64	14
176	41	46	18
177	55	34	17
178	35	16	29
179	52	26	13
180	43	26	22
181	31	76	25
182	22	53	28
183	26	29	27
184	50	40	19
185	55	50	10
186	54	10	12
187	60	15	14
188	47	66	24
189	30	60	16
190	30	50	33
191	12	17	15
192	15	14	11
193	16	19	18
194	21	48	17
195	50	30	21
196	51	42	27
197	50	15	19
198	48	21	20
199	12	38	5

Gráfico de la instancia CMT-5



Nodos de la instancia CMT-11

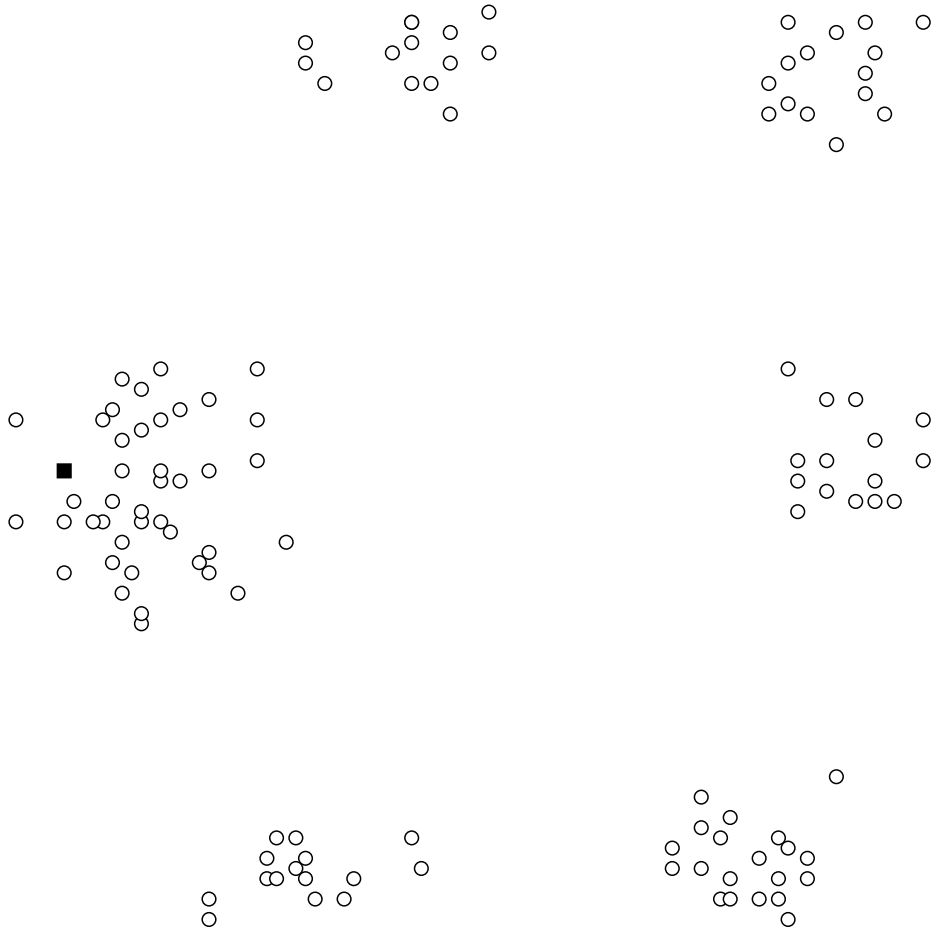
La instancia CMT-11 fue propuesta por Christofides et al. [28], tiene 120 nodos y $Q = 200$.

i	x_i	y_i	d_i
0	10	45	–
1	25	1	25
2	25	3	7
3	31	5	13
4	32	5	6
5	31	7	14
6	32	9	5
7	34	9	11
8	46	9	19
9	35	7	5
10	34	6	15
11	35	5	15
12	47	6	17
13	40	5	13
14	39	3	12
15	36	3	18
16	73	6	13
17	73	8	18
18	24	36	12
19	76	6	17
20	76	10	4
21	76	13	7
22	78	3	12
23	78	9	13
24	79	3	8
25	79	5	16
26	79	11	15
27	82	3	6
28	82	7	5
29	90	15	9
30	84	3	11
31	84	5	10
32	84	9	3
33	85	1	7
34	87	5	2
35	85	8	4
36	87	7	4
37	86	41	18
38	86	44	14
39	86	46	12
40	85	55	17

i	x_i	y_i	d_i
41	89	43	20
42	89	46	14
43	89	52	16
44	92	42	10
45	92	52	9
46	94	42	11
47	94	44	7
48	94	48	13
49	96	42	5
50	99	46	4
51	99	50	21
52	83	80	13
53	83	83	11
54	85	81	12
55	85	85	14
56	85	89	10
57	87	80	8
58	87	86	16
59	90	77	19
60	90	88	5
61	93	82	17
62	93	84	7
63	93	89	16
64	94	86	14
65	95	80	17
66	99	89	13
67	37	83	17
68	50	80	13
69	35	85	14
70	35	87	16
71	44	86	7
72	46	89	13
73	46	83	9
74	46	87	11
75	46	89	35
76	48	83	5
77	50	85	28
78	50	88	7
79	54	86	3
80	54	90	10

i	x_i	y_i	d_i
81	10	35	7
82	10	40	12
83	18	30	11
84	17	35	10
85	16	38	8
86	14	40	11
87	15	42	21
88	11	42	4
89	18	40	15
90	21	39	16
91	20	40	4
92	18	41	16
93	20	44	7
94	22	44	10
95	16	45	9
96	20	45	11
97	25	45	17
98	30	55	12
99	20	50	11
100	22	51	7
101	18	49	9
102	16	48	11
103	20	55	12
104	18	53	7
105	14	50	8
106	15	51	6
107	16	54	5
108	28	33	12
109	33	38	13
110	30	50	7
111	13	40	7
112	15	36	8
113	18	31	11
114	25	37	13
115	30	46	11
116	25	52	10
117	16	33	7
118	25	35	4
119	5	40	20
120	5	50	13

Gráfico de la instancia CMT-11



Nodos de la instancia CMT-12

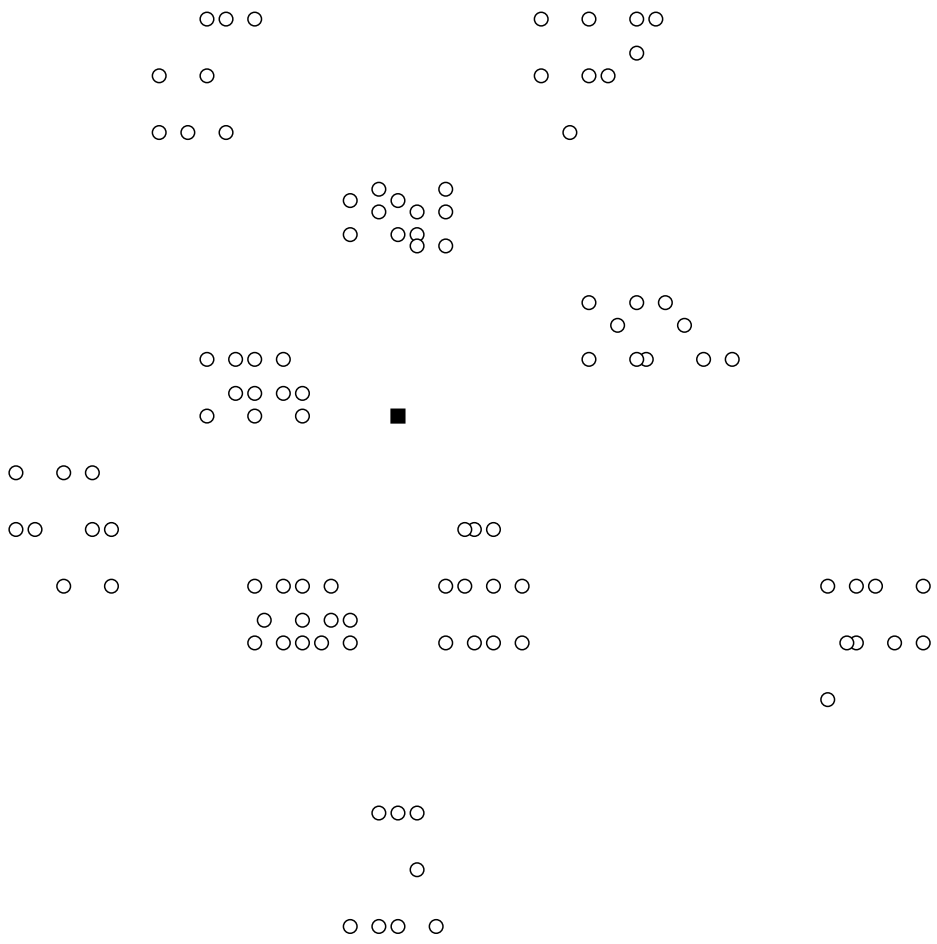
La instancia CMT-12 fue propuesta por Christofides et al. [28], tiene 100 nodos y $Q = 200$.

i	x_i	y_i	d_i
0	40	50	–
1	45	68	10
2	45	70	30
3	42	66	10
4	42	68	10
5	42	65	10
6	40	69	20
7	40	66	20
8	38	68	20
9	38	70	10
10	35	66	10
11	35	69	10
12	25	85	20
13	22	75	30
14	22	85	10
15	20	80	40
16	20	85	40
17	18	75	20
18	15	75	20
19	15	80	10
20	30	50	10
21	30	52	20
22	28	52	20
23	28	55	10
24	25	50	10
25	25	52	40
26	25	55	10
27	23	52	10
28	23	55	20
29	20	50	10
30	20	55	10
31	10	35	20
32	10	40	30
33	8	40	40

i	x_i	y_i	d_i
34	8	45	20
35	5	35	10
36	5	45	10
37	2	40	20
38	0	40	30
39	0	45	20
40	35	30	10
41	35	32	10
42	33	32	20
43	33	35	10
44	32	30	10
45	30	30	10
46	30	32	30
47	30	35	10
48	28	30	10
49	28	35	10
50	26	32	10
51	25	30	10
52	25	35	10
53	44	5	20
54	42	10	40
55	42	15	10
56	40	5	30
57	40	15	40
58	38	5	30
59	38	15	10
60	35	5	20
61	50	30	10
62	50	35	20
63	50	40	50
64	48	30	10
65	48	40	10
66	47	35	10
67	47	40	10

i	x_i	y_i	d_i
68	45	30	10
69	45	35	10
70	95	30	30
71	95	35	20
72	53	30	10
73	92	30	10
74	53	35	50
75	45	65	20
76	90	35	10
77	88	30	10
78	88	35	20
79	87	30	10
80	85	25	10
81	85	35	30
82	75	55	20
83	72	55	10
84	70	58	20
85	68	60	30
86	66	55	10
87	65	55	20
88	65	60	30
89	63	58	10
90	60	55	10
91	60	60	10
92	67	85	20
93	65	85	40
94	65	82	10
95	62	80	30
96	60	80	10
97	60	85	30
98	58	75	20
99	55	80	10
100	55	85	20

Gráfico de la instancia CMT-12



Nodos de la instancia F-11

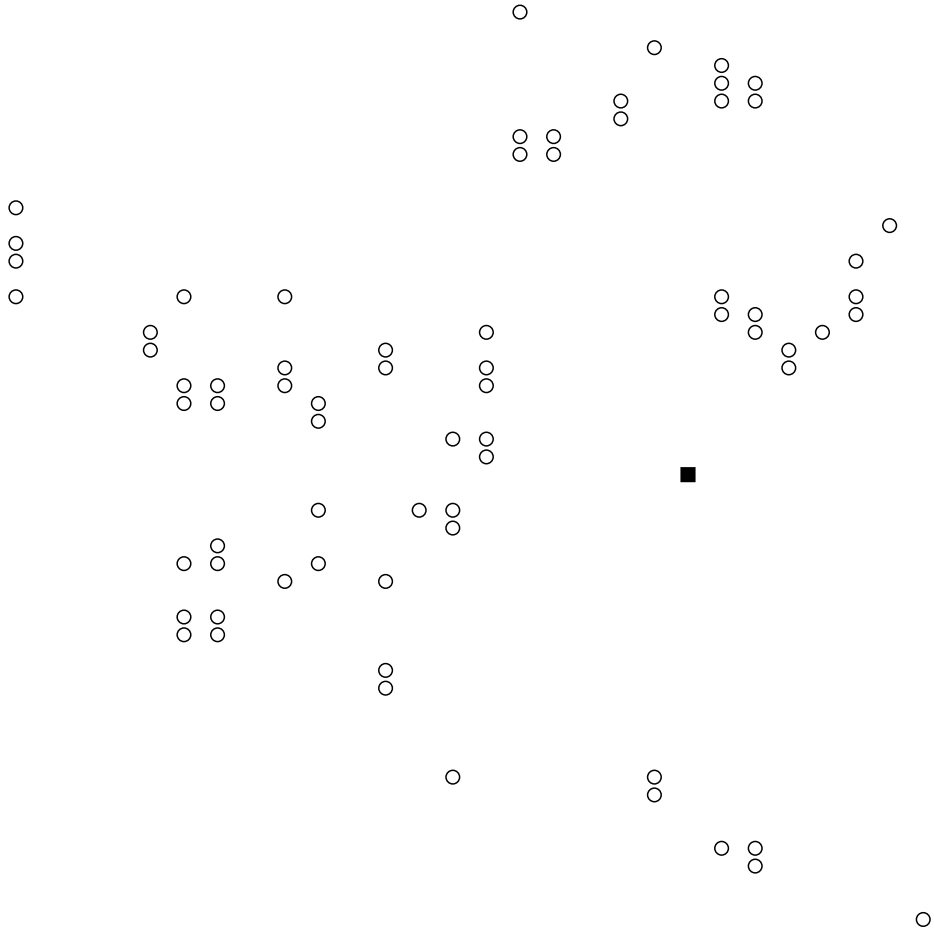
La instancia F-11 fue propuesta por Fisher [49], tiene 71 nodos y $Q = 30000$.

i	x_i	y_i	d_i
0	0	0	–
1	-12	-6	7063
2	-15	-5	51
3	-1	-18	23
4	2	-21	3074
5	-1	-17	349
6	-9	-12	1047
7	2	-22	698
8	1	-21	3001
9	7	-25	31
10	-7	-17	1135
11	-11	-5	21611
12	-14	-9	57
13	-14	-8	51
14	-11	-2	551
15	-14	-5	179
16	-15	-9	6
17	-15	-8	528
18	-9	-6	2832
19	-14	-4	1514
20	3	6	889
21	5	9	2554
22	5	10	1215
23	2	8	1810

i	x_i	y_i	d_i
24	1	9	3050
25	1	10	4
26	2	9	1563
27	6	14	741
28	5	12	1532
29	3	7	709
30	4	8	1022
31	-6	1	883
32	-6	2	1689
33	-8	-2	10235
34	-7	2	29
35	-7	-3	2894
36	-7	-2	450
37	-20	12	411
38	-20	13	207
39	-12	10	496
40	-20	15	1021
41	-6	8	117
42	2	21	46
43	2	22	8
44	1	21	18
45	-1	24	561
46	1	22	1877
47	-2	20	3542

i	x_i	y_i	d_i
48	-2	21	801
49	-4	18	967
50	-4	19	62
51	-5	18	1366
52	-5	26	230
53	1	23	4
54	-6	5	12
55	-6	6	145
56	-9	6	7149
57	-9	7	2250
58	-12	5	383
59	-12	6	134
60	-11	3	1947
61	-11	4	182
62	-14	4	3934
63	-14	5	468
64	-15	4	18
65	-15	5	133
66	-16	7	2340
67	-16	8	754
68	-15	10	1264
69	-20	10	806
70	-5	19	3665
71	-9	-11	2452

Gráfico de la instancia F-11

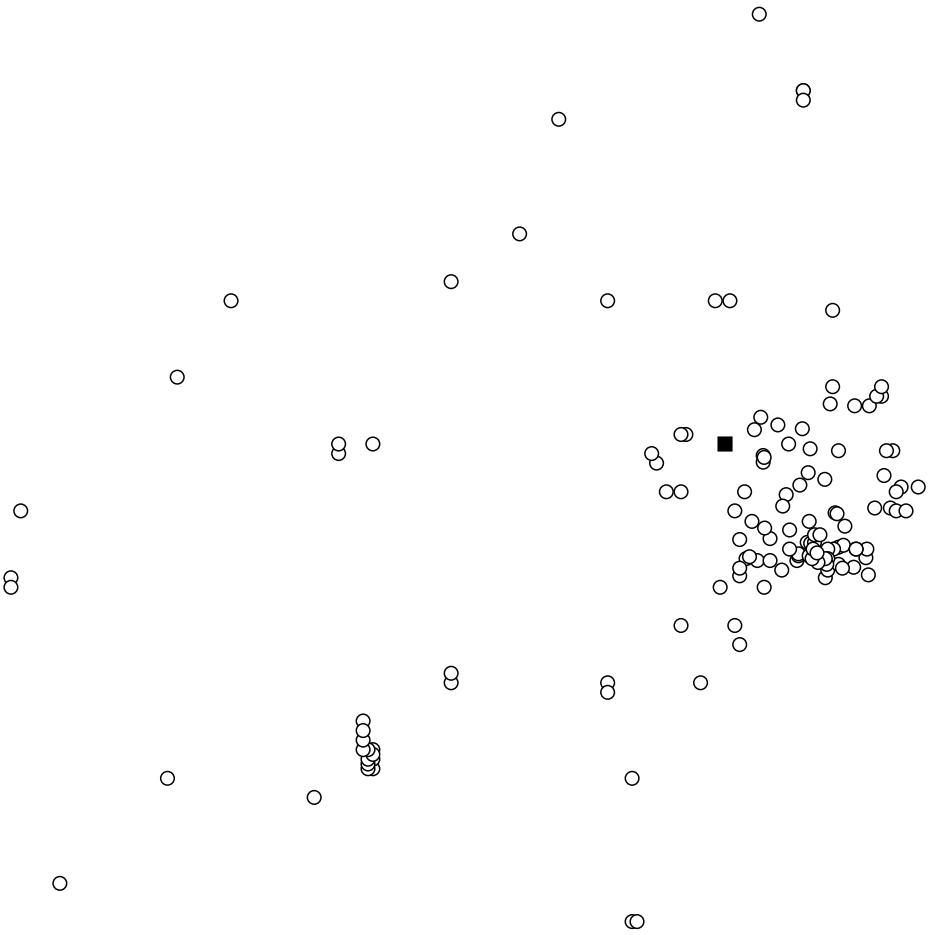


Nodos de la instancia F-12

La instancia F-12 fue propuesta por Fisher [49], tiene 134 nodos y $Q = 2210$.

i	x_i	y_i	d_i
0	-6	15	–
1	3.2	5.1	30
2	24.6	8.3	226
3	23.3	1.3	37
4	27.8	8.3	24
5	29	8	36
6	31	8	1
7	33.5	10.5	31
8	30	10.5	24
9	29	10	30
10	26.5	11.7	24
11	28.3	14.3	24
12	27	14.3	32
13	23.5	19	24
14	26	20	24
15	25	20	19
16	20.5	19	24
17	-20	13	18
18	-21	14	36
19	-30	30	115
20	-5	30	24
21	1.3	17.8	24
22	1.8	13.8	61
23	1.8	13.1	71
24	2	13.6	36
25	4.8	17	18
26	7	15	30
27	9.8	16.6	31
28	11.4	14.5	36
29	14.4	11.3	18
30	11	12	1004
31	9.3	10.7	18
32	0.6	2.8	34
33	-30	-10	504
34	2	0	18
35	14.5	1	39
36	15	1.8	24
37	17.2	2.4	37
38	17.2	4.2	24
39	18.2	4.4	99
40	20.3	2.1	24
41	22.8	3.1	24
42	23	4	36
43	20.8	4	30
44	20.8	4	25
45	18.5	6.4	24
46	-14	16	122
47	-0.5	6.9	196
48	3.2	2.8	229
49	5.6	1.8	83
50	8.7	2.8	18
51	9	3.3	24
52	9	3.5	306
53	11.2	3.3	18
54	10.8	4.7	20
55	11.5	4.6	18
56	12.3	4.7	24
57	12.3	5.5	22
58	11.2	6.9	24
59	6.5	9.7	18
60	5.8	8.5	18
61	7.2	6	24
62	7.2	4	24
63	-4	-4	30
64	-3	1.2	24
65	-40	49	40
66	-15	10	166
67	-11	-10	254
68	-25	-20	187
69	-25	-35	94
70	-24	-35	17
71	-18	10	285
72	-2	10	24
73	-4	8	24
74	-3	5	205
75	2.1	6.2	23
76	-1.7	3	28
77	-3	2	51
78	-7	0	49
79	-3	-6	19
80	-30	-11	262
81	-62	-10	120
82	-8	30	266
83	1	60	704
84	10	52	38
85	10	52	18
86	10	51	30
87	16	29	25
88	26	21	12
89	16	21	18
90	15.5	19.2	25
91	0	16.5	35
92	17.2	14.3	18
93	16.5	7.8	12
94	16.9	7.7	20
95	18	2	1126
96	16.2	4	9
97	15	4	36
98	15	3	12
99	14.8	2.4	31
100	14.5	3	96
101	13	2.6	27
102	11.8	3	54
103	12	4	137
104	12.8	3.6	12
105	13.4	5.5	58
106	-150	8	206
107	-152	1	178
108	-152	0	486
109	-142	-31	36
110	-78	-19	261
111	-78	-18	135
112	-78	-17	135
113	-80	-14	373
114	-118	22	535
115	-107	30	42
116	-85	14	9
117	-78	15	110
118	-15	16	36
119	-62	32	18
120	-120	-20	726
121	-90	-22	187
122	-79	-19	23
123	-79	-18.5	134
124	-79	-18	47
125	-78	-17.5	51
126	-79	-17	43
127	-80	-17	79
128	-80	-16	112
129	-80	-15	91
130	-48	37	232
131	-85	15	483
132	-62	-9	828
133	-15	-4	11
134	-1	3.2	12

Gráfico de la instancia F-12



Apéndice D

Nuevas soluciones encontradas

En este apéndice se detallan las soluciones obtenidas para las instancias en que todos los trabajos anteriores obtuvieron soluciones no factibles. Se reporta la solución factible de menor costo hallada para cada instancia.

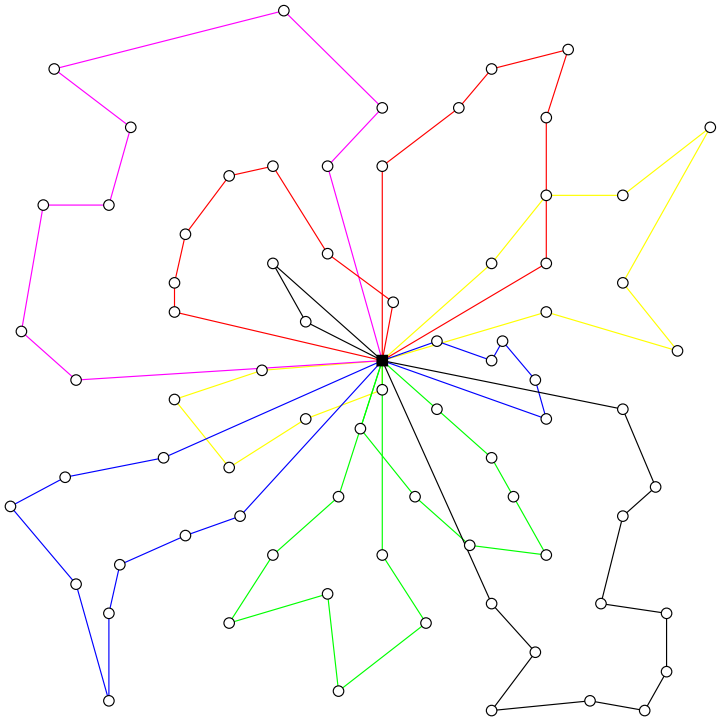
Para cada una de las soluciones reportadas se indica la cantidad de nodos, el horizonte de tiempo y el valor de la mejor solución conocida para instancia base. Se provee información acerca de la versión del algoritmo que la encontró (AMPE o AMPD), el costo total y el *GAP*. Se muestra la secuencia de nodos de cada ruta y la asignación de las rutas a los vehículos disponibles. Además, se provee una tabla con información acerca de la utilización de cada vehículo. Finalmente, se muestra la solución en forma gráfica.

Instancia CMT-2 | 6 | T_1

En esta instancia, $n = 75$, $m = 6$, $T = 146$ y $z_{\text{VRP}}^{\text{best}} = 835.26$. La solución fue obtenida por el algoritmo AMPD. Se tiene $c(s) = 857.58$ y $GAP(s) = 2.67\,\%$.

Ruta	Veh.	$d(r)$	$t(r)$	Secuencia de nodos
1	0	136	115.82	$(0, 47, 36, 69, 71, 60, 70, 20, 37, 15, 57, 13, 0)$
2	0	53	28.91	$(0, 17, 40, 0)$
3	1	112	38.29	$(0, 67, 34, 46, 52, 27, 0)$
4	1	139	106.59	$(0, 73, 1, 43, 42, 64, 41, 56, 23, 63, 0)$
5	2	138	81.81	$(0, 35, 11, 66, 65, 38, 58, 0)$
6	2	135	63.27	$(0, 26, 12, 39, 9, 32, 44, 3, 0)$
7	3	138	89.34	$(0, 74, 21, 61, 28, 22, 62, 2, 0)$
8	3	137	55.35	$(0, 68, 30, 48, 5, 29, 45, 4, 0)$
9	4	140	135.48	$(0, 72, 10, 31, 55, 25, 50, 18, 24, 49, 0)$
10	5	97	47.41	$(0, 75, 6, 33, 16, 51, 0)$
11	5	139	95.33	$(0, 8, 54, 19, 59, 14, 53, 7, 0)$

k	$\lvert R_k \rvert$	t_k	t_k/T
0	2	144.73	99.1 %
1	2	144.88	99.2 %
2	2	145.08	99.4 %
3	2	144.69	99.1 %
4	1	135.48	92.8 %
5	2	142.74	97.8 %

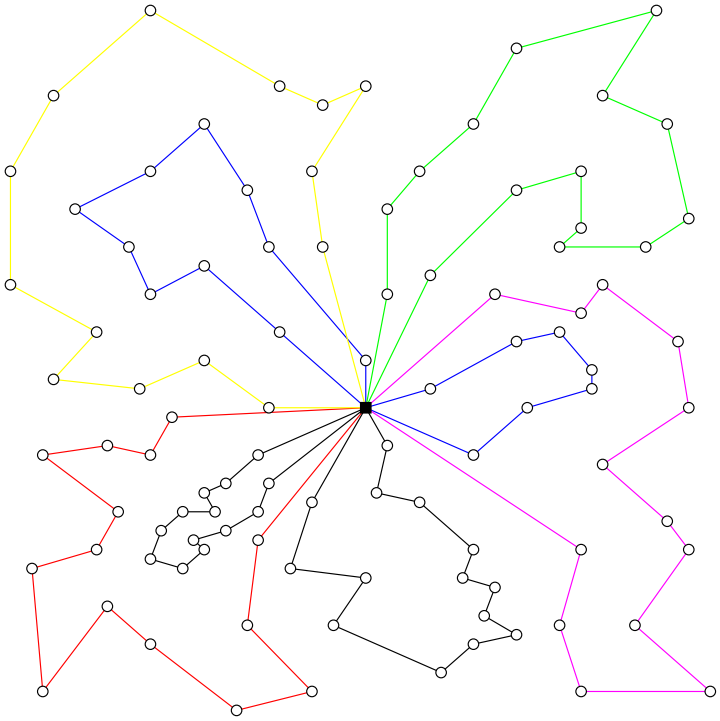


Instancia CMT-3 | 6 | T_1

En esta instancia, $n = 100$, $m = 6$, $T = 145$ y $z_{\text{VRP}}^{\text{best}} = 826.14$. La solución fue obtenida por el algoritmo AMPE. Se tiene $c(s) = 836.21$ y $GAP(s) = 1.22\,\%$.

Ruta	Veh.	$d(r)$	$t(r)$	Secuencia de nodos
1	0	198	84.77	$(0, 53, 58, 40, 21, 73, 72, 74, 75, 22, 41, 57, 2, 87, 13, 0)$
2	0	199	58.26	$(0, 94, 95, 92, 98, 37, 100, 91, 85, 93, 59, 99, 96, 6, 0)$
3	1	166	88.35	$(0, 52, 7, 82, 48, 47, 19, 11, 62, 88, 27, 0)$
4	1	121	51.46	$(0, 28, 76, 77, 68, 80, 12, 26, 0)$
5	2	191	135.62	$(0, 60, 5, 84, 17, 61, 16, 86, 38, 44, 14, 43, 15, 42, 97, 0)$
6	3	199	138.41	$(0, 1, 51, 9, 81, 33, 78, 34, 35, 71, 65, 66, 20, 30, 70, 69, 0)$
7	4	197	139.06	$(0, 50, 3, 79, 29, 24, 54, 55, 25, 39, 67, 23, 56, 4, 0)$
8	5	187	140.28	$(0, 89, 18, 83, 45, 8, 46, 36, 49, 64, 63, 90, 32, 10, 31, 0)$

k	$\lvert R_k \rvert$	t_k	t_k/T
0	2	143.03	98.6 %
1	2	139.81	96.4 %
2	1	135.62	93.5 %
3	1	138.41	95.5 %
4	1	139.06	95.9 %
5	1	140.28	96.7 %



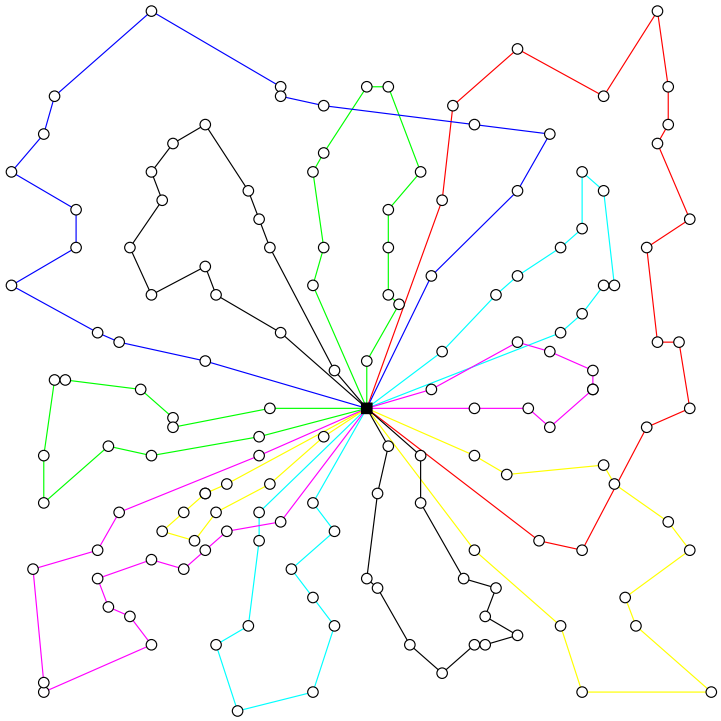
Instancia CMT-4 | 7 | T_1

En esta instancia, $n = 150$, $m = 7$, $T = 154$ y $z_{\text{VRP}}^{\text{best}} = 1028.42$. La solución fue obtenida por el algoritmo AMPD. Se tiene $c(s) = 1068.49$ y $GAP(s) = 3.90\,\%$.

Ruta	Veh.	$d(r)$	$t(r)$	Secuencia de nodos
1	0	195	70.34	$(0, 53, 58, 2, 115, 145, 41, 22, 133, 75, 74, 72, 73, 40, 105, 0)$
2	0	195	82.81	$(0, 146, 88, 148, 62, 11, 107, 19, 123, 48, 82, 7, 106, 52, 0)$
3	1	200	150.74	$(0, 18, 114, 8, 46, 124, 47, 36, 143, 49, 64, 63, \ldots$ $\ldots, 126, 90, 20, 103, 51, 1, 0)$
4	2	200	152.49	$(0, 110, 4, 134, 24, 29, 121, 78, 34, 135, 35, \ldots$ $\ldots, 136, 65, 71, 66, 128, 122, 0)$
5	3	197	77.84	$(0, 89, 118, 60, 83, 125, 45, 17, 113, 84, 5, 147, 0)$
6	3	182	75.02	$(0, 27, 132, 69, 101, 70, 30, 131, 32, 108, 10, 31, 127, 0)$
7	4	141	48.88	$(0, 28, 76, 116, 68, 80, 150, 109, 12, 138, 0)$
8	4	198	103.25	$(0, 117, 92, 37, 100, 91, 141, 44, 119, 14, 38, 140, 86, 16, 61, 6, 0)$
9	5	186	47.83	$(0, 112, 94, 59, 98, 85, 93, 99, 104, 96, 0)$
10	5	196	105.58	$(0, 21, 56, 23, 67, 39, 139, 25, 55, 130, 54, 149, 26, 0)$
11	6	146	80.02	$(0, 13, 137, 87, 144, 57, 15, 43, 142, 42, 97, 95, 0)$
12	6	199	73.70	$(0, 111, 50, 102, 33, 81, 9, 120, 129, 79, 3, 77, 0)$

k	$\lvert R_k \rvert$	t_k	t_k/T
0	2	153.15	99.4 %
1	1	150.74	97.9 %
2	1	152.49	99.0 %
3	2	152.86	99.3 %

k	$\lvert R_k \rvert$	t_k	t_k/T
4	2	152.13	98.8 %
5	2	153.41	99.6 %
6	2	153.72	99.8 %



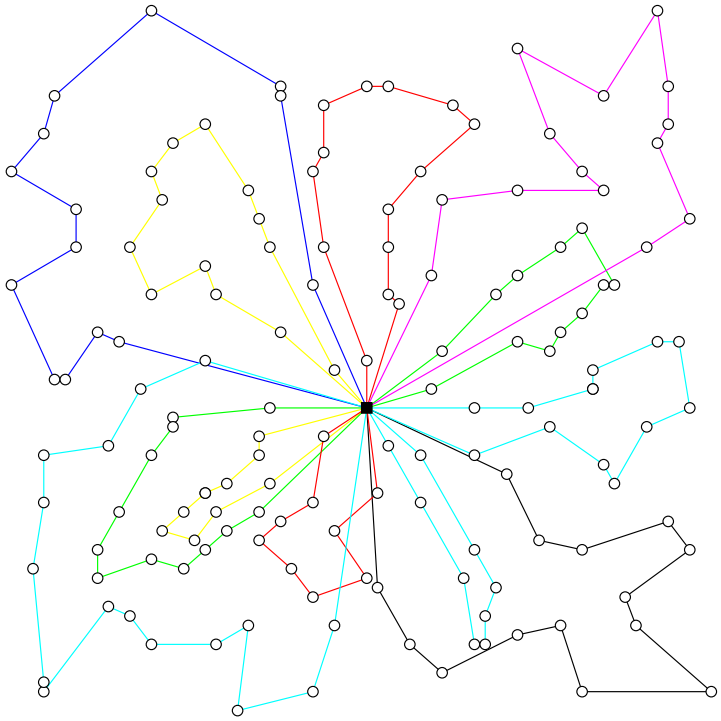
Instancia CMT-4 | 8 | T_1

En esta instancia, $n = 150$, $m = 8$, $T = 135$ y $z_{\text{VRP}}^{\text{best}} = 1028.42$. La solución fue obtenida por el algoritmo AMPE. Se tiene $c(s) = 1056.58$ y $GAP(s) = 2.74\,\%$.

Ruta	Veh.	$d(r)$	$t(r)$	Secuencia de nodos
1	0	198	120.97	$(0, 115, 145, 41, 75, 56, 23, 67, 39, 139, 25, 55, 4, 110, 149, 0)$
2	1	199	133.72	$(0, 127, 126, 63, 64, 49, 143, 36, 47, 124, 46, 45, 125, 8, 114, 0)$
3	2	143	52.28	$(0, 112, 13, 117, 97, 87, 144, 2, 137, 58, 0)$
4	2	193	82.46	$(0, 27, 31, 10, 108, 90, 32, 131, 128, 20, 30, 70, 101, 69, 132, 0)$
5	3	192	69.52	$(0, 89, 60, 118, 5, 61, 16, 141, 91, 100, 37, 92, 95, 0)$
6	3	199	62.99	$(0, 28, 76, 116, 77, 3, 79, 129, 81, 33, 102, 50, 111, 0)$
7	4	196	134.32	$(0, 1, 122, 51, 120, 9, 103, 66, 71, 65, 136, 35, 135, 34, 78, 0)$
8	5	195	82.81	$(0, 146, 88, 148, 62, 11, 107, 19, 123, 48, 82, 7, 106, 52, 0)$
9	5	185	49.22	$(0, 94, 59, 98, 85, 93, 104, 99, 96, 6, 147, 0)$
10	6	200	133.50	$(0, 18, 83, 84, 17, 113, 86, 140, 38, 44, 119, 14, 142, \ldots$ $\ldots, 42, 43, 15, 57, 0)$
11	7	197	77.72	$(0, 26, 109, 54, 130, 134, 24, 29, 121, 68, 150, 80, 12, 138, 0)$
12	7	138	57.08	$(0, 105, 21, 72, 74, 133, 22, 73, 40, 53, 0)$

k	$\lvert R_k \rvert$	t_k	t_k/T
0	1	120.97	89.6 %
1	1	133.72	99.1 %
2	2	134.74	99.8 %
3	2	132.51	98.2 %

k	$\lvert R_k \rvert$	t_k	t_k/T
4	1	134.32	99.5 %
5	2	132.03	97.8 %
6	1	133.50	98.9 %
7	2	134.80	99.9 %

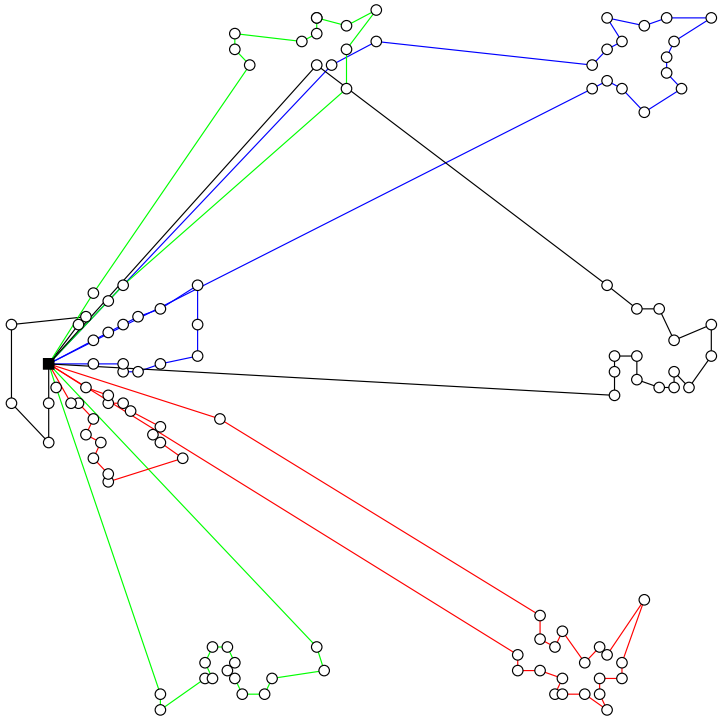


Instancia CMT-11 | 4 | T_1

En esta instancia, $n = 120$, $m = 4$, $T = 274$ y $z_{\text{VRP}}^{\text{best}} = 1042.11$. La solución fue obtenida por el algoritmo AMPE. Se tiene $c(s) = 1078.64$ y $GAP(s) = 3.51\,\%$.

Ruta	Veh.	$d(r)$	$t(r)$	Secuencia de nodos
1	0	200	224.31	$(0, 73, 40, 43, 45, 48, 51, 50, 49, 47, 46, 44, 41, 42, \ldots$ $\ldots, 39, 38, 37, 0)$
2	0	66	44.94	$(0, 82, 81, 119, 120, 106, 105, 0)$
3	1	200	220.93	$(0, 52, 54, 57, 59, 65, 61, 62, 64, 66, 63, 60, 56, 58, \ldots$ $\ldots, 55, 53, 79, 76, 0)$
4	1	132	52.67	$(0, 95, 96, 93, 94, 97, 115, 110, 98, 116, 100, 99, \ldots$ $\ldots, 101, 102, 0)$
5	2	186	54.95	$(0, 87, 92, 89, 91, 90, 114, 18, 118, 108, 83, 113, \ldots$ $\ldots, 117, 84, 112, 85, 86, 111, 0)$
6	2	197	207.94	$(0, 17, 16, 19, 25, 22, 24, 27, 33, 30, 31, 34, 36, 29, \ldots$ $\ldots, 35, 32, 28, 26, 23, 20, 21, 109, 0)$
7	3	195	137.93	$(0, 107, 67, 69, 70, 71, 74, 75, 72, 78, 80, 77, 68, 103, 104, 0)$
8	3	199	134.96	$(0, 88, 2, 1, 3, 4, 5, 6, 7, 9, 10, 11, 15, 14, 13, 12, 8, 0)$

k	$\lvert R_k \rvert$	t_k	t_k/T
0	2	269.25	98.3 %
1	2	273.60	99.9 %
2	2	262.89	95.9 %
3	2	272.89	99.6 %

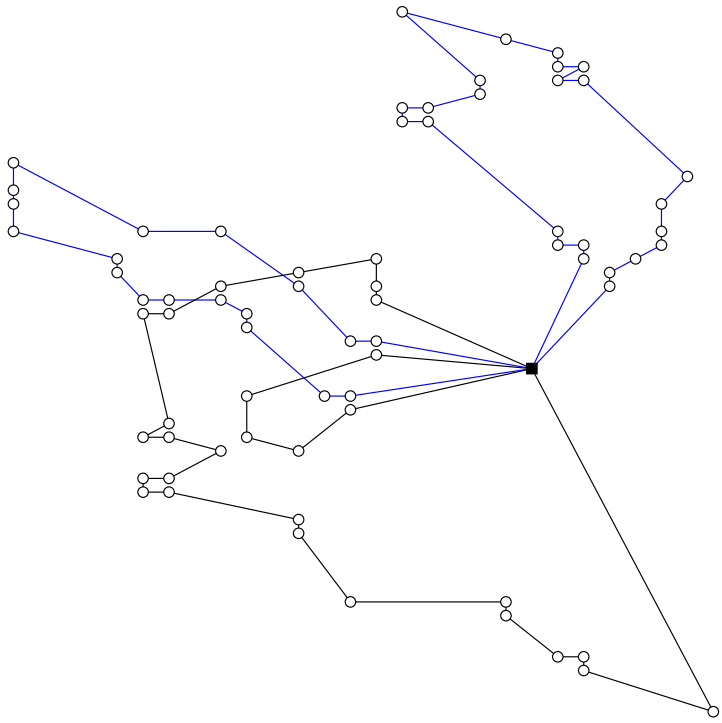


Instancia F-11 | 2 | T_1

En esta instancia, $n = 71$, $m = 2$, $T = 127$ y $z_{\text{VRP}}^{\text{best}} = 241.97$. La solución fue obtenida por ambos algoritmos. Se tiene $c(s) = 250.85$ y $GAP(s) = 3.67\,\%$.

Ruta	Veh.	$d(r)$	$t(r)$	Secuencia de nodos
1	0	27869	95.67	$(0, 9, 7, 4, 8, 3, 5, 10, 6, 71, 12, 16, 17, 13, 1, 15, \ldots$ $\ldots, 2, 19, 64, 62, 59, 57, 41, 55, 54, 0)$
2	0	28771	28.37	$(0, 35, 18, 11, 14, 31, 0)$
3	1	29964	59.10	$(0, 36, 33, 60, 61, 58, 63, 65, 66, 67, 69, 37, 38, \ldots$ $\ldots, 40, 68, 39, 56, 34, 32, 0)$
4	1	28236	67.70	$(0, 23, 26, 24, 25, 49, 51, 70, 50, 47, 48, 52, 45, \ldots$ $\ldots, 53, 46, 43, 44, 42, 27, 28, 22, 21, 30, 29, 20, 0)$

k	$\lvert R_k \rvert$	t_k	t_k/T
0	2	124.04	97.7 %
1	2	126.80	99.8 %



Indice de tablas

Indice de figuras