Heurísticas para Problemas de Ruteo de Vehículos

Alfredo Olivera
Instituto de Computación, Facultad de Ingeniería,
Universidad de la República, Montevideo, Uruguay.
aolivera@fing.edu.uy

Agosto 2004

Resumen

Desde la primera formulación de los Problemas de Ruteo de Vehículos, una gran cantidad de métodos han sido propuestos para su resolución. Estos métodos incluyen tanto algoritmos exactos como heurísticas. En este trabajo se presenta un relevamiento de algunas de las heurísticas que han sido más significativas.

Palabras clave: Ruteo de Vehículos, Heurísticas, Optimización Combinatoria.

Indice

1 Problemas de Ruteo de Vehículos **1**
- 1.1 Introducción . . . 1
- 1.2 Características de los Problemas . . . 2
  - 1.2.1 Los Clientes . . . 2
  - 1.2.2 Los Depósitos . . . 2
  - 1.2.3 Los Vehículos . . . 3
- 1.3 Formulación Matemática . . . 3
  - 1.3.1 El Problema del Agente Viajero (TSP) . . . 4
  - 1.3.2 El Problema de los m Agentes Viajeros (m-TSP) . . . 5
  - 1.3.3 El Problema con Capacidades (VRP o CVRP) . . . 6
  - 1.3.4 El Problema con Flota Heterogénea (FSMVRP) . . . 7
  - 1.3.5 El Problema con Ventanas de Tiempo (VRPTW) . . . 8
- 1.4 Métodos Exactos . . . 9

2 Heurísticas Clásicas para el VRP **10**
- 2.1 El Algoritmo de Ahorros . . . 10
  - 2.1.1 Algoritmo de Ahorros basado en Matching . . . 12
- 2.2 Heurísticas de Inserción . . . 13
  - 2.2.1 Inserción Secuencial de Mole & Jameson . . . 14
  - 2.2.2 Inserción en Paralelo de Christofides, Mingozzi y Toth . . . 15
- 2.3 Métodos Asignar Primero - Rutear Después . . . 16
  - 2.3.1 Heurística de Barrido o *Sweep* . . . 16
  - 2.3.2 Heurística de Asignación Generalizada de Fisher y Jaikumar 17
  - 2.3.3 Heurística de Localización de Bramel y Simchi-Levi . . . 18
- 2.4 Método Rutear Primero - Asignar Después . . . 20
- 2.5 Algoritmos de Pétalos . . . 21
- 2.6 Procedimientos de Búsqueda Local . . . 21
  - 2.6.1 El operador λ-intercambio . . . 22
  - 2.6.2 El algoritmo de Lin-Kernigham . . . 24
  - 2.6.3 El operador Or-opt . . . 24
  - 2.6.4 Operadores de Van Breedam . . . 25
  - 2.6.5 GENI y GENIUS . . . 25
  - 2.6.6 Transferencias cíclicas . . . 27

3 Extensiones de las heurísticas clásicas **28**
- 3.1 Algoritmo de Ahorros . . . 28
- 3.2 Heurísticas de Inserción Secuencial . . . 30
- 3.3 Heurísticas de Inserción en Paralelo . . . 32

3.4 Algoritmo de Barrido 33
3.5 Rutear Primero - Asignar Después 34

4 Metaheurísticas **35**
4.1 Algoritmos de Hormigas 35
4.1.1 Ant System Híbrido para VRP 38
4.1.2 MACS para VRPTW 39
4.2 Tabu Search 40
4.2.1 Tabu Search para el VRP 41
4.2.2 Tabu Search para el VRPTW 46
4.3 Algoritmos Genéticos 47
4.3.1 Algoritmos Genéticos para el TSP 48
4.3.2 Algoritmos Genéticos para el VRP 48
4.3.3 Algoritmos Genéticos para el VRPTW 49

Capítulo 1

Problemas de Ruteo de Vehículos

1.1 Introducción

El problema de distribuir productos desde ciertos depósitos a sus usuarios finales juega un papel central en la gestión de algunos sistemas logísticos y su adecuada planificación puede significar considerables ahorros. Esos potenciales ahorros justifican en gran medida la utilización de técnicas de Investigación Operativa como facilitadoras de la planificación, dado que se estima que los costos del transporte representan entre el 10% y el 20% del costo final de los bienes [1].

En ese sentido, las últimas cuatro décadas han visto un enorme esfuerzo por resolver estos problemas. En 1959, Dantzig y Ramser [2] realizaron por primera vez una formulación del problema para una aplicación de distribución de combustible. Cinco años más tarde, Clarke y Wright [3] propusieron el primer algoritmo que resultó efectivo para su resolución: el popular Algoritmo de Ahorros. A partir de estos trabajos, el área de *Ruteo de Vehículos* ha crecido de manera explosiva. 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 problemas de manera eficiente.

Estos modelos y algoritmos deben su éxito, en buena parte, a la evolución de los sistemas informáticos. El crecimiento en el poder de cómputo y la baja en sus costos, ha permitido disminuir los tiempos de ejecución de los algoritmos. Por otro lado, el desarrollo de los Sistemas de Información Geográfica resulta fundamental para lograr una adecuada interacción de los modelos y algoritmos con los encargados de realizar la planificación.

Pero el interés que reviste el área no es puramente práctico. Los Problemas de Ruteo de Vehículos son Problemas de Optimización Combinatoria y pertenecen, en su mayoría, a la clase NP-Hard. La motivación académica por resolverlos radica en que no es posible construir algoritmos que en tiempo polinomial resuelvan cualquier instancia del problema (a no ser que $P = NP$).

1.2 Características de los Problemas

A grandes rasgos un Problema de Ruteo de Vehículos consiste en, dado un conjunto de clientes y depósitos dispersos geográficamente y una flota de vehículos, determinar un conjunto de rutas de costo mínimo que comiencen y terminen en los depósitos, para que los vehículos visiten a los clientes. Las características de los clientes, depósitos y vehículos, así como diferentes restricciones operativas sobre las rutas, dan lugar a diferentes variantes del problema.

1.2.1 Los Clientes

Cada cliente tiene cierta demanda que deberá ser satisfecha por algún vehículo. En muchos casos, la demanda es un bien que ocupa lugar en los vehículos y es usual que un mismo vehículo no pueda satisfacer la demanda de todos los clientes en una misma ruta. Un caso equivalente al anterior ocurre cuando los clientes son proveedores y lo que se desea es recoger la mercadería 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 la demanda no es un bien sino un servicio: el 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 y desea ser transportado hacia otro sitio. Aquí la capacidad del vehículo impone una cota sobre la cantidad de clientes que puede alojar simultáneamente.

Es usual que cada cliente deba ser 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 se puede arribar al cliente.

En problemas con varios vehículos diferentes podría existir restricciones de compatibilidad 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).

1.2.2 Los Depósitos

Tanto los vehículos como las mercaderías a distribuir (si las hubiera) suelen estar ubicadas en depósitos. Usualmente se exige que cada ruta comience y finalice en un mismo depósito, aunque este podría no ser el caso en algunas aplicaciones (por ejemplo, podría ser que el viaje debiera finalizar en el domicilio del conductor del vehículo).

En los problemas con múltiples depósitos cada uno de estos tiene diferentes características, por ejemplo, su ubicación y capacidad máxima de producción. Podría ocurrir que cada depósito tenga una flota de vehículos asignada *a priori* o que dicha asignación sea parte de lo que se desea determinar.

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, podría querer evitarse que demasiados vehículos estén operando en un mismo depósito a la vez (es decir, la *congestión* del depósito).

1.2.3 Los Vehículos

La capacidad de un vehículo podría tener varias dimensiones, como por ejemplo peso y volumen. Cuando en un mismo problema existen diferentes mercaderías, los vehículos podrían tener compartimentos, de modo que la capacidad del vehículo dependa de la mercadería de que se trate. En general, cada vehículo tiene asociado un costo fijo en el que se incurre al utilizarlo y un costo variable proporcional a la distancia que recorra.

Los problemas en que los atributos (capacidad, costo, etc.) son los mismos para todos los vehículos se denominan de *flota homogénea*, y, si hay diferencias, de *flota heterogénea*. La cantidad de vehículos disponibles podría ser un dato de entrada o una variable de decisión. El objetivo mas usual suele ser utilizar la menor cantidad de vehículos y minimizar la distancia recorrida ocupa un segundo lugar.

Regulaciones legales podrían imponer restricciones sobre el tiempo máximo que un vehículo puede estar en circulación e incluso prohibir el pasaje de ciertos vehículos por ciertas zonas. En algunos casos se desea que la cantidad de trabajo realizado por los vehículos (usualmente el tiempo de viaje) no sea muy dispar.

En general se asume 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.

1.3 Formulación Matemática

En esta sección se formulan algunos de los problemas clásicos y sus extensiones como problemas de Programación Entera. Dichas formulaciones se dan por completitud y para evitar ambigüedad en la definición, pero no se reportan en este trabajo métodos exactos de resolución.

La red de transporte por la que circulan los vehículos se modela mediante un grafo ponderado $G = (V, E, C)$. Los nodos del grafo representan a los clientes y depósitos. En problemas con un depósito y n clientes, el nodo 0 representa al depósito y los nodos $1, \ldots, n$ a los clientes. En algunos casos (en que se explicitará) se agrega una copia del depósito etiquetada con $n+1$ para simplificar la formulación.

Cada arco $(i, j) \in E$ representa el mejor camino para ir desde el nodo i hacia el nodo j en la red de transporte y tiene asociado un costo c_{ij} y un tiempo de viaje t_{ij}. Según la estructura de los costos y los tiempos y las características de la red, el grafo puede ser simétrico o asimétrico. Puede suponerse que G es completo, pues entre todo par de lugares de una red de transporte razonable, debería existir algún camino. Sin embargo, por una cuestión de flexibilidad, los modelos serán planteados sin realizar dicha hipótesis.

Denotaremos por $\Delta^+(i)$ y $\Delta^-(i)$ al conjunto de nodos adyacentes e incidentes al nodo i, es decir, $\Delta^+(i) = \{j \in V \mid (i, j) \in E\}$ y $\Delta^-(i) = \{j \in V \mid (j, i) \in E\}$.

De manera similar, el conjunto de arcos incidentes hacia el exterior e interior del nodo i se definen como $\delta^+(i) = \{(i,j) \in E\}$ y $\delta^-(i) = \{(j,i) \in E\}$.

1.3.1 El Problema del Agente Viajero (TSP)

En el Problema del Agente Viajero (o TSP por *Travelling Salesman Problem*) se dispone de un solo vehículo que debe visitar a todos los clientes en una sola ruta y a costo mínimo. No suele haber un depósito (y si lo hubiera no se distingue de los clientes), no hay demanda asociada a los clientes y tampoco hay restricciones temporales. El problema puede formularse como:

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1.1}$$

$$\text{s.a.} \sum_{j\in\Delta^+(i)} x_{ij} = 1 \qquad \forall\, i \in V \tag{1.2}$$

$$\sum_{i\in\Delta^-(j)} x_{ij} = 1 \qquad \forall\, j \in V \tag{1.3}$$

$$\sum_{i\in S, j\in\Delta^+(i)\setminus S} x_{ij} \geq 1 \quad \forall\, S \subset V \tag{1.4}$$

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

Esta formulación fue propuesta por Dantzig, Fulkerson y Johnson [4]. Las variables binarias x_{ij} indican si el arco (i,j) es utilizado en la solución. La función objetivo (1.1) establece que el costo total de la solución es la suma de los costos de los arcos utilizados. Las restricciones 1.2 y 1.3 indican que la ruta debe llegar y abandonar cada nodo exactamente una vez. Finalmente, las restricciones 1.4 son llamadas *restricciones de eliminación de sub-tours* e indican que todo subconjunto de nodos S debe ser abandonado al menos una vez. Notar que si no se impusieran estas restricciones la solución podría constar de más de un ciclo, como se muestra en la Figura 1.1. Esta solución viola la restricción 1.4 para $S = \{0,1,2\}$. Existen diferentes tipos de restricciones de eliminación de sub-tours.

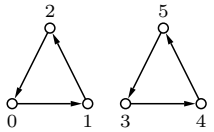


Figura 1.1: Una solución formada por 2 sub-tours.

Asumiendo que $|E| = O(n^2)$, esta formulación tiene una cantidad $O(n^2)$ de variables binarias y $O(2^n)$ restricciones. El problema puede formularse con una cantidad polinomial de restricciones, agregando variables reales u_i para $i = 1, \ldots, n$ y sustituyendo las restricciones 1.4 por

$$u_i - u_j + n x_{ij} \leq n - 1 \quad \forall\, (i,j) \in E, i \neq 0, j \neq 0.$$

Estas desigualdades fueron propuestas por Miller, Tucker y Zemlin [5] y fuerzan a que las variables reales u determinen una cantidad estrictamente creciente a

lo largo de la ruta (es decir, $u_j \geq u_i + 1$ si j es visitado inmediatamente después que i). Bajo la hipótesis de que $|E| = O(n^2)$, en esta nueva formulación hay $O(n^2)$ variables binarias, $O(n)$ variables positivas y $O(n^2)$ restricciones. Sin embargo, esta formulación no resulta apta para la resolución de problemas de tamaño considerable mediante métodos exactos, pues si bien se disminuye la cantidad de restricciones, la cota que se obtiene resolviendo su relajación lineal resulta en general poco ajustada.

La mayor parte de los problemas de ruteo de vehículos son generalizaciones del TSP. En ese sentido, éste puede considerarse el problema de ruteo de vehículos más simple. No obstante, pertenece a la clase de problemas NP-Hard [6] y es uno de los Problemas de Optimización Combinatoria más clásico y difundido.

1.3.2 El Problema de los m Agentes Viajeros (m-TSP)

El Problema de los m Agentes Viajeros o m-TSP es una generalización del TSP en la cual se tiene un depósito y m vehículos. El objetivo es construir exactamente m rutas, una para cada vehículo, de modo que cada cliente sea visitado una vez por uno de los vehículos. Cada ruta debe comenzar y finalizar en el depósito y puede contener a lo sumo p clientes. Una formulación, dada por Miller et al. [5] es la siguiente:

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij} x_{ij} & & \\
\text{s.a.} \quad & \sum_{j\in\Delta^+(0)} x_{0j} = m & & (1.5)\\
& \sum_{j\in\Delta^+(i)} x_{ij} = 1 & \forall\, i \in V \setminus \{0\} & (1.6)\\
& \sum_{i\in\Delta^-(j)} x_{ij} = 1 & \forall\, j \in V \setminus \{0\} & (1.7)\\
& u_i - u_j + p x_{ij} \leq p - 1 & \forall\, (i,j) \in E, i \neq 0, j \neq 0 & (1.8)\\
& x_{ij} \in \{0,1\} & \forall\, (i,j) \in E & \\
& u_i \geq 0 & \forall\, i \in V \setminus \{0\} &
\end{aligned}
$$

El modelo es similar al segundo modelo presentado para el TSP. La restricción 1.5 indica que exactamente m vehículos salen del depósito y las 1.6 y 1.7 aseguran que cada cliente es un nodo intermedio en exactamente una ruta. Finalmente, con 1.8 se eliminan los sub-tours y se impone que en cada ruta no haya más de p clientes.

En el caso que $p = n$ (es decir, cuando la cantidad de clientes por ruta no está acotada) el m-TSP puede formularse como un TSP con m copias del depósito tales que la distancia entre ellas es infinita. Las soluciones a ese TSP no utilizarán arcos que conectan dos copias del depósito y por lo tanto, pueden ser interpretadas como soluciones del m-TSP.

1.3.3 El Problema con Capacidades (VRP o CVRP)

El VRP es una extensión del m-TSP en la cual cada cliente $i \in V \setminus \{0\}$ tiene asociada una demanda d_i [1] y cada vehículo tiene una capacidad C (la flota es homogénea). En este problema la cantidad de rutas no es fijada de antemano como en el TSP y en el m-TSP.

Para un conjunto de clientes S, $d(S) = \sum_{i \in S} d_i$ es su demanda total y $r(S)$ indica la mínima cantidad de vehículos necesarios servirlos a todos. 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. El problema se formula de la siguiente manera [1]:

$$\min \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{1.9}$$

$$\text{s.a.} \sum_{j \in \Delta^+(0)} x_{0j} = m \tag{1.10}$$

$$\sum_{i \in \Delta^-(0)} x_{i0} = m \tag{1.11}$$

$$\sum_{j \in \Delta^+(i)} x_{ij} = 1 \qquad \forall\, i \in V \setminus \{0\} \tag{1.12}$$

$$\sum_{i \in \Delta^-(j)} x_{ij} = 1 \qquad \forall\, j \in V \setminus \{0\} \tag{1.13}$$

$$\sum_{i \in S, j \in \Delta^+(i) \setminus S} x_{ij} \geq r(S) \quad \forall\, S \subset V \setminus \{0\} \tag{1.14}$$

$$m \geq 1$$

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

La función objetivo (1.9) es el costo total de la solución. Las restricciones 1.10 y 1.11 indican que m es la cantidad de vehículos utilizados en la solución y que todos los vehículos que parten del depósito deben regresar. Las restricciones 1.12 y 1.13 aseguran que todo cliente es un nodo intermedio de alguna ruta. Finalmente, la restricción 1.14 actúa como restricción de eliminación de sub-tours y a la vez impone que la demanda total de los clientes visitados por un vehículo no puede superar la capacidad C.

Determinar el valor de $r(S)$ requiere la resolución del siguiente problema:

$$\begin{aligned} r(S) = \min & \sum_{k \in K} y_k \\ \text{s.a.} & \sum_{i \in S} d_i x_{ik} \leq C y_k \quad \forall\, k \in K \\ & \sum_{k \in K} x_{ik} = 1 \qquad \forall\, i \in S \\ & x_{ik} \in \{0, 1\} \qquad \forall\, i \in S, \forall\, k \in K \\ & y_k \in \{0, 1\} \qquad \forall\, k \in K \end{aligned}$$

[1] En algunos casos, por preservar la notación original, se utilizará d_{ij} para indicar la distancia entre los nodos i y j y algún otro símbolo para la demanda de los clientes.

donde K es un conjunto con suficientes vehículos para satisfacer la demanda (por ejemplo, n). Este problema es conocido como Bin Packing Problem (BPP) [7]. Una cota inferior para la cantidad de vehículos está dada por el valor óptimo de la relajación lineal del BPP, que es $\left\lceil \frac{d(s)}{C} \right\rceil$. La formulación es válida incluso cuando se sustituye $r(S)$ por la cota inferior $\left\lceil \frac{d(s)}{C} \right\rceil$.

En algunos casos se agrega a este problema la restricción de que ninguna ruta puede tener largo mayor que cierta cota L. Dicha restricción no puede incorporarse a esta formulación, pues las rutas no están individualizadas.

En esta formulación la cantidad de vehículos m es una variable de decisión que no tiene cota superior, es decir, se asume que la disponibilidad de vehículos es ilimitada. Si se tuviera un flota finita se podría agregar una cota superior para m o directamente fijar su valor. Notar que está implícito que cada vehículo puede recorrer a lo sumo una ruta.

1.3.4 El Problema con Flota Heterogénea (FSMVRP)

En los problemas con flota heterogénea los costos y capacidades de los vehículos varían, existiendo un conjunto $T = \{1, \ldots, |T|\}$ de tipos de vehículo. La capacidad de los vehículos $k \in T$ es q^k y su costo fijo (si lo tuvieran) es f^k. Los costos y tiempos de viaje para cada tipo de vehículo son c_{ij}^k y t_{ij}^k respectivamente. Se asume que los índices de los vehículos están ordenados en forma creciente por capacidad (es decir, $q^{k_1} \leq q^{k_2}$ para $k_1, k_2 \in T, k_1 < k_2$).

En la siguiente formulación de *flujo de vehículos de tres índices* [8] se agrega un índice para discriminar entre los tipos de vehículos.

$$\min \sum_{k \in T} f^k \sum_{j \in \Delta^+(0)} x_{0j}^k + \sum_{k \in T} \sum_{(i,j) \in E} c_{ij}^k x_{ij}^k \tag{1.15}$$

$$\text{s.a.} \sum_{k \in T} \sum_{i \in \Delta^+(j)} x_{ij}^k = 1 \qquad \forall\, j \in V \setminus \{0\} \tag{1.16}$$

$$\sum_{j \in \Delta^+(i)} x_{ij}^k - \sum_{j \in \Delta^-(i)} x_{ji}^k = 0 \qquad \forall\, i \in V, \forall\, k \in T \tag{1.17}$$

$$r_0 = 0 \tag{1.18}$$

$$r_j - r_i \geq (d_j + q^{|T|}) \sum_{k \in T} x_{ij}^k - q^{|T|} \qquad \forall\, i \in V \setminus \{0\}, \forall\, j \in \Delta^+(i) \tag{1.19}$$

$$r_j \leq \sum_{k \in T} \sum_{i \in \Delta^-(j)} q_k x_{ij}^k \qquad \forall\, j \in V \setminus \{0\} \tag{1.20}$$

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

$$r_j \geq 0 \qquad \forall\, j \in V$$

Las variables binarias x_{ij}^k indican si el arco (i,j) es utilizado por el vehículo k y las variables r_i positivas indican la carga acumulada en la ruta correspondiente hasta el nodo i (inclusive). La función objetivo (1.15) mide el costo total de la solución incluyendo costos fijos y variables. Las restricciones 1.16 establecen que todo cliente debe ser visitado por algún vehículo. En 1.17 se indica que si un vehículo de tipo k visita al nodo i, entonces un vehículo del mismo tipo debe abandonarlo. Las restricciones 1.18 y 1.19 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 1.20.

En esta formulación se asume que la cantidad de vehículos de cada tipo es ilimitada. El problema correspondiente se denomina *Fleet Size and Mix Vehicle Routing Problem* (o FSMVRP). No solo se debe decidir las rutas, sino la composición de la flota de vehículos a utilizar. Usualmente, al tratar con problemas de flota heterogénea, se opta por utilizar este modelo aún cuando en algunos casos no refleja la realidad.

Si la cantidad de vehículos disponibles de cada tipo k fuera v_k, conocida de antemano, debería agregarse las restricciones

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

1.3.5 El Problema con Ventanas de Tiempo (VRPTW)

En esta variante del problema, además de capacidades, cada cliente $i \in V \setminus \{0\}$ 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 (i, j) es un arco de la solución y t_i y t_j son las horas de arribo a los clientes i y j, las ventanas de tiempo implican que necesariamente debe cumplirse $t_i \leq l_i$ y $t_j \leq l_j$. Por otro lado, si $t_i < e_i$, entonces el vehículo deberá esperar hasta que el cliente "abra" y necesariamente $t_j = e_i + s_i + t_{ij}$.

Utilizando los nodos 0 y $n+1$ para representar al depósito y el conjunto K para representar a los vehículos (no a los tipos de vehículos como en la sección anterior), el problema se formula para una flota de vehículos posiblemente heterogénea, de la siguiente manera [9]:

$$\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ij}^k x_{ij}^k \tag{1.21}$$

$$\text{s.a.} \sum_{k \in K} \sum_{j \in \Delta^-(i)} x_{ij}^k = 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{1.22}$$

$$\sum_{j \in \Delta^+(0)} x_{0j}^k = 1 \qquad \forall k \in K \tag{1.23}$$

$$\sum_{j \in \Delta^+(i)} x_{ij}^k - \sum_{j \in \Delta^-(i)} x_{ji}^k = 0 \qquad \forall k \in K,\, i \in V \setminus \{0, n+1\} \tag{1.24}$$

$$\sum_{i \in V \setminus \{0, n+1\}} d_i \sum_{j \in \Delta^+(i)} x_{ij}^k \leq q^k \qquad \forall k \in K \tag{1.25}$$

$$y_j^k - y_i^k \geq s_i + t_{ij}^k - M(1 - x_{ij}^k) \qquad \forall i, j \in V \setminus \{0, n+1\},\, k \in K \tag{1.26}$$

$$e_i \leq y_i^k \leq l_i \qquad \forall i \in V \setminus \{0, n+1\},\, k \in K \tag{1.27}$$

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

$$y_i^k \geq 0 \qquad \forall i \in V \setminus \{0, n+1\},\, k \in K$$

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 arribo al 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 (1.21) es el costo total de las rutas.

La restricción 1.22 indica que todos los clientes deben ser visitados. Las restricciones 1.23 y 1.24 determinan que cada vehículo $k \in K$ recorre un camino de 0 a $n+1$. La capacidad de cada vehículo es impuesta en 1.25. Siendo M una constante lo suficientemente grande, la restricción 1.26 asegura que si un vehículo k viaja de i a j, no puede llegar a j antes que $y_i + s_i + t_{ij}^k$, y actúan además como restricciones de elimianción de sub-tours. Finalmente, los límites de las ventanas de tiempo son impuestos en 1.27.

1.4 Métodos Exactos

Dada la complejidad de los problemas, solo las instancias con pocos clientes (hasta 50 aproximadamente) pueden ser resueltas consistentemente por métodos exactos. En este tipo de metodologías, suele resolverse alguna relajación del problema y utilizarse un esquema de ramificación y acotamiento al estilo del método *Branch and Bound* [10]. También se han propuesto algoritmos basados en Programación Dinámica que aceleran los cálculos mediante una relajación del espacio de estados. Por otro lado, hay diversas implementaciones del método de Generación de Columnas, que han resultado especialmente efectivas para problemas con ventanas de tiempo muy ajustados. Por un completo compendio de métodos exactos para Problemas de Ruteo de Vehículos, puede consultarse los trabajos de Laporte y Norbert [11] y de Laporte [12].

Capítulo 2

Heurísticas Clásicas para el VRP

En este capítulo se presentan algunas de las heurísticas clásicas más significativas para el VRP con capacidades y, en algunos casos, la restricción sobre el largo máximo de cada ruta. Estas heurísticas son procedimientos simples que realizan una exploración limitada del espacio de búsqueda y dan soluciones de calidad aceptable en tiempos de cálculo generalmente moderados. Las soluciones obtenidas con esta clase de procedimientos pueden, en general, ser mejoradas utilizando métodos de búsqueda más sofisticados, pero incurriendo en elevados tiempos de ejecución. Muchas de estas heurísticas pueden ser extendidas para manejar restricciones adicionales a las del VRP.

2.1 El Algoritmo de Ahorros

Uno de los algoritmos más difundidos para el VRP es el *Algoritmo de Ahorros* de Clarke y Wright [3]. 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 2.1, el ahorro (en distancia) obtenido por dicha unión es

$$s_{ij} = c_{i0} + c_{0j} - c_{ij} \tag{2.1}$$

pues en la nueva solución los arcos $(i, 0)$ y $(0, j)$ no serán utilizados y se agregará el arco (i, j). En este algoritmo se parte de una solución inicial y se realizan las uniones que den mayores ahorros siempre que no violen las restricciones del

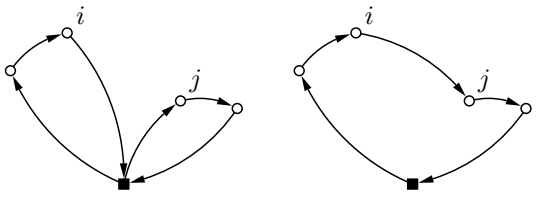


Figura 2.1: Dos rutas antes y después de ser unidas

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.

Algoritmo de Ahorros (Versión paralela)
Paso 1 (inicialización). Para cada cliente i construir la ruta $(0, i, 0)$.
Paso 2 (cálculo de ahorros). Calcular s_{ij} para cada par de clientes i y j.
Paso 3 (mejor unión). Sea $s_{i^*j^*} = \max s_{ij}$, donde el máximo se toma entre los ahorros que no han sido considerados aún. Sean r_{i^*} y r_{j^*} las rutas que contienen a los clientes i^* y j^* respectivamente. Si i^* es el último cliente de r_{i^*} y j^* es el primer cliente de r_{j^*} y la combinación de r_{i^*} y r_{j^*} es factible, combinarlas. Eliminar $s_{i^*j^*}$ de futuras consideraciones. Si quedan ahorros por examinar ir a 3, si no terminar.

Algoritmo de Ahorros (Versión secuencial)
Paso 1 (inicialización). Para cada cliente i construir la ruta $(0, i, 0)$.
Paso 2 (cálculo de ahorros). Calcular s_{ij} para cada par de clientes i y j.
Paso 3 (selección). Si todas las rutas fueron consideradas, terminar. Si no, seleccionar una ruta que aún no haya sido considerada.
Paso 4 (extensión). Sea $(0, i, \ldots, j, 0)$ la ruta actual. Si no existe ningún ahorro conteniendo a i o a j, ir a 3. Sea s_{k^*i} (o s_{jl^*}) el máximo ahorro conteniendo a i (o a j). Si k^* (o l^*) es el último (o primer) cliente de su ruta y la combinación de dicha ruta con la actual es factible, realizar dicha combinación. Eliminar s_{k^*i} (o s_{jl^*}) de futuras consideraciones. Ir a 4.

Dado que en la definición de s_{ij} solamente interviene la ubicación de los clientes i y j, todos los ahorros pueden calcularse una sola vez al comienzo de la ejecución del algoritmo. En la versión secuencial, podría calcularse los ahorros a medida que son necesarios.

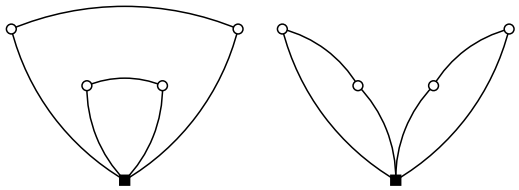
Si el máximo ahorro es negativo, la combinación de las rutas aumentará la distancia recorrida pero disminuirá la cantidad de rutas de la solución (y por lo tanto la cantidad de vehículos utilizados). Dependiendo de las particularidades de cada problema, debe decidirse si realizar o no ese tipo de combinaciones.

Se ha observado que utilizando la definición original de ahorro suele generarse algunas rutas circulares (ver Figura 2.2) lo cual puede ser negativo. Para solucionar este problema algunos autores [13, 14, 15] proponen redefinir el ahorro como

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij} \tag{2.2}$$

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

En general ocurre que al comienzo de la ejecución del algoritmo dos alternativas pueden parecer equivalentes y, sin embargo, la elección tiene un gran impacto en la solución final. Las soluciones obtenidas con el Algoritmo de Ahorros pueden, en general, ser mejoradas mediante operadores de búsqueda local como el algoritmo 3-opt [16].



(a) Rutas Circulares (b) Rutas Radiales

Figura 2.2: Un ejemplo de rutas circulares y radiales.

2.1.1 Algoritmo de Ahorros basado en Matching

Cuando en el Algoritmo de Ahorros se decide unir dos rutas r_i y r_j, se está descartando otras uniones posibles en las que participan r_i o r_j (porque i y j dejan de ser extremos en la nueva ruta). La unión que da el máximo ahorro podría, en algunos casos, hacer que las uniones que permanecen factibles no sean buenas. Elegir siempre el máximo ahorro es una estrategia demasiado voraz. En el Algoritmo de Ahorros Basado en Matching se decide la union a realizar considerando como afecta ésta a las posibles uniones en iteraciones siguientes. Para esto, se considera un grafo que tiene a todas las rutas como nodos y un arco entre dos nodos p y q cuyo peso es el ahorro obtenido si las rutas correspondientes se combinan (siempre que la combinación sea factible). Un Matching de peso máximo[1] sobre dicho grafo da un conjunto de combinaciones a realizar que es globalmente bueno.

Algoritmo de Ahorros Basado en Matching
Paso 1 (inicialización). Para cada cliente i construir la ruta $(0, i, 0)$.
Paso 2 (cálculo de los ahorros). Actualizar s_{pq} para cada par de rutas p y q que pueda ser combinado manteniendo la factibilidad. Si ningún par de rutas puede ser combinado, terminar.
Paso 3 (matching). Resolver un problema de matching de peso máximo sobre un grafo cuyos nodos son las rutas de la solución actual y en el cual hay un arco entre las rutas p y q con peso s_{pq} si su combinación es factible.
Paso 4 (uniones). Dado el matching de peso máximo, combinar todo par de rutas p y q tal que el arco (p, q) pertenezca al matching. Ir a 2.

Hallar un Matching de Peso Máximo en un grafo puede resolverse en tiempo polinomial en la cantidad de nodos del grafo [17]. Pero, por un lado, el grafo tiene tantos nodos como rutas haya en la solución (inicialmente hay n^2 rutas) y, por otro lado, se halla un nuevo matching en cada iteración. Entonces, para problemas grandes puede ser conveniente hallar el matching en forma aproximada mediante alguna heurística.

En la Figura 2.3 se muestra una posible solución parcial de 5 rutas, el grafo de asociado y la nueva solución inducida por el Matching de Peso Máximo que es $\{(1, 3), (4, 5)\}$. El ahorro total es de 17. Supongamos que el vehículo no tiene capacidad para satisfacer la demanda de ninguna combinación de tres de las rutas originales. Si se hubiera ejecutado el Algoritmo de Ahorros clásico,

[1] Un Matching en un grafo es un conjunto de arcos que no tienen extremos en común. El Peso de un Matching es la suma de los pesos de sus arcos.

se hubiera combinado primero las rutas 3 y 4 y luego las 2 y 5, obteniendo un ahorro total de 15.

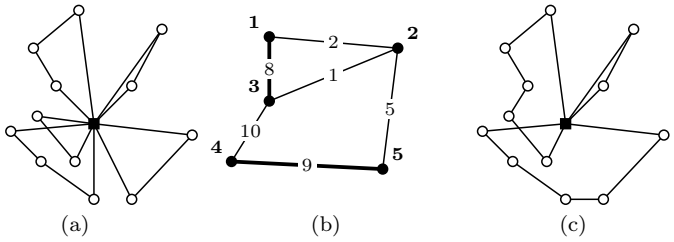


Figura 2.3: Una iteración de un Algoritmo de Ahorros basado en Matching

En la propuesta de Desrochers y Verhoog [18] y de Altinkemer y Gavish [19], al combinar dos rutas no solamente se considera la posibilidad de insertar una al final de la otra como en el Algoritmo de Ahorros original, sino todas las posibilidades de armar una nueva ruta con los clientes de ambas. Entonces, si S_p y S_q son los conjuntos de clientes de las rutas p y q, el ahorro por combinarlas se define como $s_{pq} = t(S_p) + t(S_q) - t(S_p \cup S_q)$, donde $t(S)$ indica el costo de una solución óptima para el TSP sobre el conjunto de clientes S. Con esta definición no solo se hace costoso el cálculo de los ahorros, sino que además éstos deben ser re-calculados en cada nueva iteración. Cuando la cantidad de clientes en cada ruta es grande conviene realizar el cálculo de s_{pq} en forma aproximada.

En las uniones dadas por el matching de peso máximo una misma ruta no participa en más de una unión (porque los arcos correspondientes forman un matching en el grafo). Sin embargo, podría ser ventajoso que la ruta resultante de una unión sea inmediatamente unida con otra. Resulta conveniente, entonces, no realizar todas las uniones dadas por el matching de peso máximo, sino solamente algunas. Una opción es combinar sólo las rutas que correspondan al arco de mayor peso del matching [18]. Otra alternativa para realizar las uniones de manera más gradual es conectar los nodos del grafo sobre el que se calcula el matching con nodos ficticios con pesos tales que se priorice la inclusión de algunos de estos últimos en el matching [19]. Así, no todos los arcos del matching se corresponderán con uniones de rutas.

En la propuesta de Wark y Holt [20] el ahorro puede definirse de la forma usual (2.1 o 2.2) o puede modificarse para privilegiar las uniones que estén lejos de ser infactibles (por ejemplo, si la ruta resultante está lejos de colmar la capacidad del vehículo o lejos de la máxima distancia permitida). Cuando no hay más combinaciones de rutas que sea factibles, algunas rutas son divididas probabilísticamente, lo que permite proseguir la búsqueda de soluciones.

2.2 Heurísticas de Inserción

Las heurísticas de inserción son métodos constructivos en los cuales se crea una solución mediante sucesivas inserciones de clientes en las rutas. En cada iteración se tiene una solución parcial cuyas rutas sólo visitan un subconjunto de los clientes y se selecciona un cliente no visitado para insertar en dicha solución.

En las heurísticas de inserción secuencial sólo se considera insertar clientes en

la última ruta creada. 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 [21, 22]. Las heurísticas de inserción en paralelo surgen para remediar esta deficiencia, permitiendo insertar un cliente en cualquiera de las rutas de la solución. Esta distinción es similar a la hecha para las dos versiones del Algoritmo de Ahorros (ver Sección 2.1).

Cualquier heurística de inserción para el TSP puede ser utilizada para el VRP siempre que se verifique la factibilidad antes de realizar las inserciones. Por un compendio de heurísticas de inserción para el TSP puede consultarse el trabajo de Bodin et al. [23]. En esta sección nos ocuparemos de aquellas diseñadas explícitamente para el VRP.

2.2.1 Inserción Secuencial de Mole & Jameson

En esta heurística [21] se utilizan dos medidas para decidir el próximo cliente a insertar en la solución parcial. Por un lado, para cada cliente no visitado se calcula la mejor posición para ubicarlo en la ruta actual teniendo en cuenta solamente las distancias y sin reordenar los nodos que ya están en la ruta. Se tiene una ruta $(v_0, v_1, \ldots, v_t, v_{t+1})$ donde $v_0 = v_{t+1} = 0$. Si w es un cliente no visitado, el costo de insertar w entre v_i y v_{i+1} $(0 \leq i \leq t)$ se define como

$$c_1(v_i, w) = \begin{cases} c_{v_i,w} + c_{w,v_{i+1}} - \lambda c_{v_i,v_{i+1}} & \text{si } (v_0, \ldots, v_i, w, v_{i+1}, \ldots, v_{t+1}) \\ & \text{es factible} \\ \infty & \text{si no} \end{cases}$$

La mejor posición para insertar el cliente w en la ruta actual está dada por

$$i(w) = \arg \min_{i=0,\ldots,t} c_1(v_i, w).$$

Si se utilizara solamente la medida c_1 para decidir el próximo cliente a insertar, es probable que los clientes lejanos al depósito no sean tenidos en cuenta sino hasta las iteraciones finales del algoritmo, es decir, cuando sean las únicas alternativas factibles. Por lo tanto, es necesario utilizar un incentivo adicional para la inserción de clientes lejanos al depósito. Se define $c_2(v_i, w) = \mu c_{0w} - c_1(v_i, w)$ para cada cliente w. En cada iteración se busca el cliente que maximiza la medida c_2 (llamada medida de *urgencia*) y se lo inserta en la posición dada por el mínimo valor de c_1.

Además de las medidas anteriores, debe considerarse la factibilidad de las inserciones. Cuando ninguna inserción es factible y si aún quedan clientes sin visitar, se selecciona un cliente para comenzar una nueva ruta. El algoritmo es el siguiente.

Algoritmo de Mole & Jameson
Paso 1 (creación de una ruta). Si todos los clientes pertenecen a alguna ruta, terminar. Si no, seleccionar un cliente no visitado w y crear la ruta $r = (0, w, 0)$.
Paso 2 (inserción). Sea $r = (v_0, v_1, \ldots, v_t, v_{t+1})$ donde $v_0 = v_{t+1} = 0$. Para cada cliente no visitado w, calcular $i(w) = \arg\min_{i=0,\ldots,t} c_1(v_i, w)$. Si no hay inserciones factibles, ir al paso 1. Calcular $w^* = \arg\max_w c_2(v_{i(w)}, w)$. Insertar w^* luego de $v_{i(w^*)}$ en r.
Paso 3 (optimización). Aplicar el algoritmo 3-opt [16] sobre r. Ir al paso 2.

Para seleccionar el cliente que iniciará una ruta pueden utilizarse diferentes alternativas, por ejemplo, el más lejano al depósito. El algoritmo utiliza dos parámetros λ y μ para modificar sus criterios de selección de clientes. Al hacer crecer el parámetro λ se favorece la inserción de clientes entre nodos lejanos; y al aumentar el valor de μ, se privilegia la inserción de clientes lejanos al depósito.

En general, los últimos clientes no visitados son lejanos entre sí y por ende, las últimas rutas construidas son de mala calidad. Para corregir esta deficiencia, se propone utilizar un procedimiento de intercambio de clientes entre las rutas una vez que el algoritmo finaliza su ejecución. Primero se busca reasignar clientes de modo de disminuir el costo de la solución. Luego, cuando todos los cambios aumentan el costo, se prosigue realizando intercambios buscando una solución que utilice menos vehículos.

2.2.2 Inserción en Paralelo de Christofides, Mingozzi y Toth

El algoritmo propuesto por Christofides, Mingozzi y Toth [24] opera en dos fases. En la primera fase se determina la cantidad de rutas a utilizar, junto con un cliente para inicializar cada una de las rutas. En la segunda fase se crean dichas rutas y se inserta el resto de los clientes en ellas.

En la primera fase del algoritmo se aplica un algoritmo de inserción secuencial para obtener rutas compactas. No se presta especial atención a la ubicación de los clientes dentro de cada ruta, pues de esta fase solo se conservan los clientes iniciales de cada ruta y la cantidad de rutas de la solución final.

Para inicializar la k-ésima ruta se selecciona un cliente v_k dentro de los no visitados. Se define el costo de insertar el cliente w en la ruta que contiene a v_k como $\delta_{w,v_k} = c_{0w} + \lambda c_{w,v_k}$ (si el cliente no puede ser insertado, la función toma el valor ∞) y se asignan clientes a la ruta comenzando por los menores valores de δ hasta que no haya inserciones factibles, en cuyo caso se crea una nueva ruta o se termina el algoritmo.

Algoritmo de Christofides, Mingozzi y Toth. Fase 1
Paso 1 (nueva ruta). Hacer $k := 1$.
Paso 2 (cliente inicial). Seleccionar un cliente no visitado v_k para insertar en la ruta. Para cada cliente no visitado w, calcular δ_{w,v_k}.
Paso 3 (inserciones). Calcular $w^* = \arg\min_w \delta_{w,v_k}$ sobre los clientes no visitados w. Insertar w^* en la ruta y aplicar el algoritmo 3-opt. Si quedan clientes no visitados que puedan insertarse en la ruta, ir a 3.
Paso 4 (siguiente ruta). Si todos los clientes pertenecen a alguna ruta, terminar. Si no, hacer $k := k + 1$ e ir a 2.

En la segunda fase del algoritmo se crean k rutas y se las inicializa con los clientes seleccionados en el paso 2 de la fase 1. Cada cliente no visitado se asocia con la ruta en la que el costo de insertarlo es minimizado. Luego se selecciona una ruta cualquiera en la que se insertan los clientes que tiene asociados. Para decidir el orden en que se insertan los clientes asociados a una ruta se calcula, para cada cliente, la diferencia entre el costo de realizar la inserción en esa ruta y en la segunda mejor opción para él. Cuanto mayor es esa diferencia, mayor es la urgencia por insertar dicho cliente en esta ruta.

Algoritmo de Christofides, Mingozzi y Toth. Fase 2
Paso 5 (inicialización). Crear k rutas $r_t = (0, v_t, 0)$ para $t = 1, \ldots, k$, siendo k la cantidad de rutas obtenidas en la fase 1. Sea $J = \{r_1, \ldots, r_k\}$.
Paso 6 (asociación). Para cada cliente w que no haya sido visitado calcular $t_w = \arg\min_{t|r_t \in J} \delta_{w,v_t}$.
Paso 7 (urgencias). Seleccionar $r_t \in J$ y hacer $J := J \setminus \{r_t\}$. Para cada cliente w tal que $t_w = t$, calcular $t'_w = \arg\min_{t|r_t \in J} \delta_{w,v_t}$ y $\tau_w = t'_w - t_w$.
Paso 8 (inserción). Calcular $w^* = \arg\max_{w|t_w=t} \tau_w$. Insertar w^* en la ruta r_t y aplicar el algoritmo 3-opt. Si quedan clientes asociados a r_t que pueden ser insertados, ir a 8.
Paso 9 (finalización). Si $J \neq \phi$, ir a 6. Si todos los clientes han sido visitados, terminar. Si no, aplicar el algoritmo nuevamente (incluyendo la fase 1) sobre los clientes no visitados.

2.3 Métodos Asignar Primero - Rutear Después

Los métodos asignar primero y rutear después (*cluster first - route second*) proceden en dos fases. Primero se busca generar grupos de clientes, 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 es un TSP que, dependiendo de la cantidad de clientes en el cluster, se puede resolver el forma exacta o aproximada.

2.3.1 Heurística de Barrido o *Sweep*

En la heurística de barrido [25, 26, 27], 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. Cada cluster es luego ruteado resolviendo un TSP de forma exacta o aproximada.

Este algoritmo puede aplicarse en problemas planos, es decir, en los que cada nodo se corresponde con un punto en el plano y las distancias entre ellos se definen como la distancia euclídea. Se supone que cada cliente i está dado por sus coordenadas polares (ρ_i, θ_i) en un sistema que tiene al depósito como origen.

Heurística de barrido
Paso 1 (inicialización). Ordenar los clientes según θ de manera creciente. Si dos clientes tienen igual valor de θ, colocar primero el de menor valor de ρ. Seleccionar un cliente w para comenzar y hacer $k := 1$ y $\mathcal{C}_k := \{w\}$
Paso 2 (selección). Si todos los clientes pertenecen a algún cluster, ir a 3. Si no, seleccionar el siguiente cliente w_i. Si w_i puede ser agregado $\mathcal{C}_k$ sin violar las restricciones de capacidad, hacer $\mathcal{C}_k := \mathcal{C}_k \cup \{w_i\}$. Si no, hacer $k := k + 1$ y crear un nuevo cluster $\mathcal{C}_k := \{w_i\}$. Ir a 2.
Paso 3 (optimización). Para cada cluster $\mathcal{C}_k$ para $t = 1, \ldots, k$, resolver un TSP con sus clientes.

Por la forma en que se generan los clusters, las rutas obtenidas no se superponen, lo que puede ser bueno en algunos casos. Un posible resultado de la aplicación de este algoritmo se muestra en la Figura 2.4 donde las líneas punteadas indican los límites de los clusters.

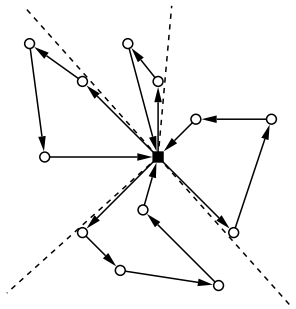


Figura 2.4: Una solución obtenida mediante el Algoritmo de Barrido

El procedimiento se repite n veces, comenzando en cada ejecución por un cliente diferente. Se propone además un procedimiento para eliminar clientes de cada ruta finalizada e insertar clientes que aún no fueron visitados, en el caso que esto disminuya el costo de la ruta. Los clientes eliminados de la ruta serán *barridos* por alguna ruta posterior. Utilizando esta variante las rutas pueden solaparse.

2.3.2 Heurística de Asignación Generalizada de Fisher y Jaikumar

Fisher y Jaikumar [28] proponen generar los clusters resolviendo un Problema de Asignación Generalizada (GAP) sobre los clientes. Primero se fijan K clientes semilla s_k con $k = 1, \ldots, K$ sobre la base de los cuales se construirán los clusters. En una segunda fase, se decide qué clientes asignar a cada uno de los clusters de modo de no violar la capacidad del vehículo, resolviendo un GAP que se define a continuación:

$$\min \sum_{k=1}^{K} \sum_{i \in V \setminus \{0\}} d_{ik} x_{ik} \tag{2.3}$$

$$\text{s.a.} \sum_{k=1}^{K} x_{ik} = 1 \qquad \forall\, i \in V \setminus \{0\} \tag{2.4}$$

$$\sum_{i \in V \setminus \{0\}} q_{ik} x_{ik} \leq Q \quad \forall\, k = 1, \ldots, K \tag{2.5}$$

$$x_{ik} \in \{0, 1\} \qquad \forall\, i \in V \setminus \{0\}, \forall\, k = 1, \ldots, K$$

Las variables x_{ik} indican si el cliente i es asignado al cluster k. El objetivo es minimizar el costo total de la asignación, como se indica en 2.3. La restricción 2.4 indica que cada cliente es asignado a exactamente un cluster. En la desigualdad 2.5 se impone que la demanda de los clientes en un mismo cluster no puede superar la capacidad del vehículo.

El costo de insertar un cliente i en el cluster k se define como el costo de la mejor inserción del cliente i ruta $(0, s_k, 0)$, es decir, $d_{ik} = \min\{c_{0i} + c_{i,s_k} + c_{s_k,0}, c_{0,s_k} + c_{s_k,i} + c_{i0}\} - (c_{0,s_k} + c_{s_k,0})$. En el caso de que los costos sean simétricos, $d_{ik} = c_{0i} + c_{i,s_k} - c_{0,s_k}$.

Heurística de Fisher y Jaikumar
Paso 1 (inicialización). Formar K clusters e inicializar cada uno con un cliente s_k $(k = 1, \ldots, K)$.
Paso 2 (asignación). Resolver el Problema de Asignación Generalizada (GAP) para decidir a qué cluster es asignado cada cliente.
Paso 3 (ruteo). Para cada cluster, resolver un TSP con sus clientes.

2.3.3 Heurística de Localización de Bramel y Simchi-Levi

El planteo de Bramel y Simchi-Levi [29] es similar al de Fisher y Jaikumar. En ambos existe un conjunto de clientes semilla y a cada uno se le asignan algunos clientes. Sin embargo, en esta propuesta, los clientes semilla son determinados por el algoritmo resolviendo un Problema de Localización de Concetradores con Capacidades (CCLP).

El CCLP se describe a continuación. Se dispone de m posibles ubicaciones para concentradores de capacidad Q_j $(j = 1, \ldots, m)$ y n terminales, cada uno de los cuales utiliza w_i $(i = 1, \ldots, n)$ de la capacidad del concentrador al que se conecta. El costo por ubicar un concentrador en la ubicación j es f_j y el costo de conectar el terminal i al concentrador j es $\hat{c}_{ij}$. El CCLP consiste en decidir cuáles concentradores colocar y qué terminales conectar a cada concentrador de modo que cada terminal se conecte con exactamente un concentrador, se satisfagan las restricciones de capacidad y se minimicen los costos. Una formulación como un Problema de Progamación 0-1 es:

$$\min \sum_{j=1}^{m} f_j y_j + \sum_{i=1}^{n} \sum_{j=1}^{m} \hat{c}_{ij} x_{ij} \tag{2.6}$$

$$\text{s.a.} \sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{2.7}$$

$$\sum_{i=1}^{n} w_i x_{ij} \leq y_j Q_j \qquad \forall j = 1, \ldots, m \tag{2.8}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i = 1, \ldots, n, \forall j = 1, \ldots, m$$

$$y_j \in \{0, 1\} \qquad \forall j = 1, \ldots, m$$

Las variables binarias y_j indican si se instala un concentrador en el sitio j y las x_{ij} indican si el terminal i se conecta al concentrador j. La función objetivo (2.6) es la suma de los costos fijos por instalación de concentradores y los costos variables de conexión de terminales. La restricción 2.7 asegura que todo terminal se conecte a exactamente un concentrador. Finalmente, la restricción 2.8 asegura, por un lado, que si se instala un concentrador en el sitio j la demanda de los terminales que se le conectan no supera su capacidad y, por otro lado, que si se decide no colocar un concentrador en j tampoco se conecten terminales a ese sitio.

El CCLP brinda un marco general para resolver Problemas de Ruteo de Vehículos, definiendo a las posibles semillas como sitios para ubicar concentradores. Dados m subconjuntos de clientes, $T_1, \ldots, T_m$, el costo de utilizar el subconjunto T_j en la solución es $t(T_j)$ (el costo de un TSP óptimo sobre los clientes T_j), siempre que la demanda de los clientes de T_j no supere la capacidad del vehículo. Además, para cualquier cliente i, el costo de agregarlo a T_j es $t(T_j \cup \{i\}) - t(T_j)$. El problema de decidir qué semillas utilizar y qué clientes conectar con qué semillas es un CCLP donde los posibles concentradores son las posibles semillas, los terminales son los clientes, los costos fijos son los costos de rutear cada semilla y los costos de conexión son los costos de inserción de clientes en las semillas.

Heurística de Localización
Paso 1 (inicialización). Seleccionar m semillas $T_1, \ldots, T_m$. Para cada semilla j, calcular $f_j = t(T_j)$ y $Q_j = Q - \sum_{k \in T_j} q_j$. Para cada cliente i y cada semilla j, calcular $\hat{c}_{ij} = t(T_j \cup \{i\}) - t(T_j)$. Definir $w_i = q_i$.
Paso 2 (localización). Resolver un CCLP con las semillas como potenciales sitios concentradores y los clientes como terminales, utilizando los costos y capacidades calculadas en el paso 1.
Paso 3 (ruteo). La solución del CCLP define un cluster para cada concentrador seleccionado. Resolver un TSP para cada uno de dichos clusters.

Este procedimiento es general, pero el cálculo de los valores f_j y $\hat{c}_{ij}$ puede consumir mucho tiempo. Se propone utilizar n conjuntos semillas donde $T_j = \{j\}$ y, por lo tanto, $f_j = 2c_{0j}$. Además, se dan dos propuestas para los costos de inserción:

Seed Tour Heuristic (STH): el costo se define como el costo de insertar i en la ruta $(0, j, 0)$, como en la Heurística de Fisher y Jaikumar. Asumiendo simetría en los costos, $\hat{c}_{ij} = c_{i0} + c_{ij} - c_{j0}$.

Star Connection Heuristic (SCH): el costo se define como el costo de ir de j a i y volver a j. En este caso tenemos $\hat{c}_{ij} = c_{ij} + c_{ji}$. Si los clientes fueran insertados de acuerdo a esta política, las rutas formadas tendrían forma de estrella (ver Figura 2.5) y de ahí el nombre de esta heurística. Se puede probar que, bajo ciertas hipótesis, utilizando esta estimación la heurística es asintóticamente óptima.

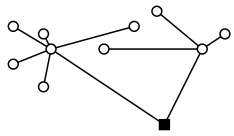


Figura 2.5: Rutas en forma de estrella utilizadas en SCH

2.4 Método Rutear Primero - Asignar Después

En los métodos rutear primero - asignar después [30] también se procede en dos fases. Primero se calcula una ruta que visita a todos los clientes resolviendo un TSP. En general esta ruta no respeta las restricciones del problema y se particiona en varias rutas, cada una de las cuales sí es factible.

Dada $r = (0, v_1, \ldots, v_n, 0)$, la solución del TSP obtenida en la primera fase, se determina la mejor partición de r que respete la capacidad del vehículo. Este problema se puede formular como el de hallar un camino mínimo en un grafo dirigido y acíclico. Para ello, se construye un grafo $G = (X, V, W)$ donde $X = \{0, v_1, \ldots, v_n\}$. Los arcos del G conectan todo par de clientes v_i y v_j con $i < j$ y tales que la demanda total de los clientes $v_{i+1}, \ldots, v_j$ no supera la capacidad del vehículo: $V = \{(v_i, v_j) \mid i < j, \sum_{k=i+1}^{j} d_{v_k} \leq Q\}$. Cada arco (v_i, v_j) se pondera con el costo de la ruta $(0, v_{i+1}, \ldots, v_j, 0)$, es decir

$$w(v_i, v_j) = c_{0,v_{i+1}} + c_{v_j,0} + \sum_{k=i+1}^{j-1} c_{v_k,v_{k+1}} \tag{2.9}$$

Un arco (v_i, v_j) representa la ruta $(0, v_{i+1}, \ldots, v_j, 0)$. Cada camino de 0 a v_n en G representa una posible partición de la ruta r en rutas que respetan las restricciones de demanda. Por lo tanto, el camino de costo mínimo entre 0 y v_n representa la partición de costo mínimo de la ruta original en rutas que respetan la restricción de capacidad. Como el grafo es acíclico (sólo hay arcos (v_i, v_j) con $i < j$), puede utilizarse el Algoritmo de Dijkstra para hallar dicho camino.

En la Figura 2.6 se presenta un posible ordenamiento de los clientes de un problema y su grafo asociado. Si el camino más corto de 0 a 4 en el grafo fuera $(0, 2, 4)$, la solución dada por el algoritmo serían las rutas $(0, 1, 2, 0)$ y $(0, 3, 5, 4, 0)$, como se indica.

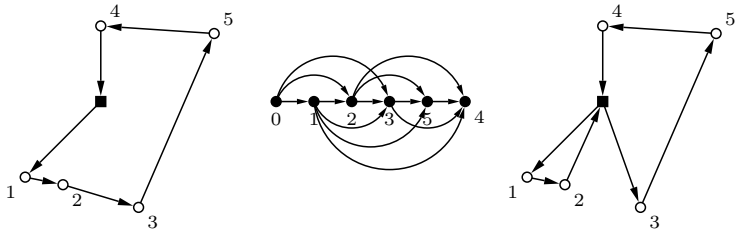


Figura 2.6: Aplicación del algoritmo rutear primero - asignar después

Aunque la ruta inicial sea la solución óptima del TSP y la partición se realice de manera óptima, las rutas obtenidas no necesariamente son una solución óptima para el problema. Por lo tanto, alcanza con que la ruta inicial se calcule en forma heurísitca, por ejemplo, mediante la aplicación de 2-opt sobre una ruta aleatoria como en el trabajo original. El algoritmo puede ejecutarse repetidas veces, partiendo de diferentes rutas iniciales.

Según la definición original, $w(v_i, v_j)$ es el costo de la ruta que comienza en v_{i+1}, sigue el orden de la ruta original y termina en v_j. Esta definición puede modificarse permitiendo variar el orden de los clientes $v_{i+1}, \ldots, v_j$ de modo de

obtener una ruta mejor. En el artículo original se aplica el algoritmo 2-opt sobre la ruta $(0, v_{i+1}, \ldots, v_j, 0)$.

2.5 Algoritmos de Pétalos

Supongamos que se dispone de un conjunto de rutas R, de modo que cada ruta $r \in R$ es factible, pero cada cliente es visitado por varias de las rutas. El problema de seleccionar un subconjunto de R de costo mínimo que visite exactamente una vez a cada cliente puede formularse como un Set Partitioning Problem (SPP):

$$\begin{aligned} &\min \sum_{k \in R} c_k x_k \\ &\text{s.a.} \sum_{k \in R} a_{ik} x_k = 1 && \forall i \in V \setminus \{0\} \\ &\quad x_k \in \{0, 1\} && \forall k \in S \end{aligned}$$

donde a_{ik} vale 1 si el cliente i es visitado por la ruta r_k y 0 si no y donde c_k es el costo de la ruta r_k. La variable x_k indica si la ruta r_k es seleccionada o no en la solución final. Esta formulación se debe a Balinski y Quandt [31].

En el caso extremo de que R contenga todas las posibles rutas factibles, solucionar el SPP es equivalente a resolver el problema en forma exacta. Como la cantidad de rutas factibles es, en el caso general, exponencial en la cantidad de clientes, se suele generar solamente un subconjunto de formado por "buenas" rutas.

Cada columna del SPP representa una ruta de R. Cuando en toda columna los ceros aparecen de forma consecutiva, el problema verifica la propiedad de Columnas Circulares y el SPP correspondiente puede ser resuelto en tiempo polinomial [32]. Trasladada al problema la propiedad establece que, para determinado ordenamiento de los clientes del problema, el conjunto de clientes visitado por cada ruta forma un intervalo (que en algunos casos tiene forma de pétalo). Cuando las rutas se generan con el Algoritmo de Barrido verifican la propiedad de Columnas Circulares. Diversas técnicas han sido propuestas para generar "buenos" conjuntos de rutas que verifiquen la propiedad llamadas 1-pétalos [33, 34] y 2-pétalos [35].

2.6 Procedimientos de Búsqueda Local

Una vez que se tiene una solución para el problema, se puede intentar mejorarla mediante algún procedimiento de búsqueda local. Para cada solución s se define un conjunto de soluciones vecinas $N(s)$. Un procedimiento de Búsqueda Local parte de una solución s, la reemplaza por una solución $s^* \in N(s)$ de menor costo y repite el procedimiento hasta que la solución no pueda ser mejorada. Al terminar, se obtiene una solución localmente óptima respecto a la definición de la vecindad. Para obtener s^* puede buscarse la mejor solución de $N(s)$ (estrategia *best improvement*) o simplemente tomar la primera solución de $N(s)$ que mejore el costo (estrategia *first improvement*).

Usualmente se define $N(s)$ como las soluciones que pueden obtenerse aplicando a s alguna regla o procedimiento sencillo llamado *movida*. Las movidas

para el VRP pueden clasificarse en movidas de una ruta y movidas multi-ruta. En las movidas de una ruta los clientes que se visitan en una ruta no varían luego de la aplicación del operador, lo que varía es el orden en que se realizan las visitas. En las movidas multi-ruta, además de cambios en el orden de las visitas suele modificarse el conjunto de clientes visitados en cada ruta.

2.6.1 El operador λ-intercambio

Uno de los operadores de búsqueda local para una ruta más conocidos es el λ-intercambio definido por Lin [16]. Un λ-intercambio consiste en eliminar λ arcos de la solución ($\lambda > 1$) y reconectar los λ segmentos restantes. Una solución se dice λ-óptima si no puede ser mejorada utilizando λ-intercambios. Se llama λ-opt a un algoritmo de búsqueda local que utiliza λ-intercambios hasta alcanzar una solución λ-óptima.

En una ruta que visita n clientes, hay $\binom{n+1}{\lambda}$ maneras posibles de eliminar λ arcos y, dada una elección de arcos a eliminar, hay $2^{\lambda-1}(\lambda-1)!$ maneras de reconectar la ruta (incluyendo la reconexión que vuelve a generar la misma ruta). Por lo tanto, la cantidad de λ-intercambios posibles es $2^{\lambda-1}(\lambda-1)!\binom{n+1}{\lambda}$. Chequear si una solución es λ-óptima puede hacerse en tiempo $O(n^\lambda)$ en el peor caso.

Usualmente se implementan 2-intercambios y 3-intercambios. Posibles movidas de este tipo se muestran en las Figuras 2.7 y 2.8. En general, estas movidas invierten el orden de algunas visitas. Asumiendo que las movidas no afectan la factibilidad y que los costos son simétricos, puede buscarse movidas que mejoren el costo de una ruta sin necesidad de explorar todas las posibilidades [36]. Si no se cumplieran dichos supuestos, buscar λ-intercambios tendría un costo computacional más elevado debido a la necesidad de incorporar chequeos de factibilidad y re-calcular algunos costos.

Renaud, Boctor y Laporte [37] propusieron una versión reducida de 4-opt llamada 4-opt*. Se parte de dos secuencias disjuntas de nodos $(v_0, \ldots, v_{u+1})$ y (v_k, v_{k+1}, v_{k+2}) respetando el orden de la ruta, donde $u \leq w$ y w es un parámetro del algoritmo. Si $\min\{c_{v_1,v_{k+1}}, c_{v_u,v_{k+1}}\} < \max\{c_{v_0,v_1}, c_{v_u,v_{u+1}}, c_{v_k,v_{k+1}}, c_{v_{k+1},v_{k+2}}\}$, se eliminan los arcos (v_0, v_1), (v_u, v_{u+1}), (v_k, v_{k+1}) y (v_{k+1}, v_{k+2}), se agrega el de menor costo entre (v_1, v_{k+1}) y (v_u, v_{k+1}) y se reconecta la ruta agregando tres arcos más. De esta manera se asegura que al menos uno de los arcos agregados tiene menor costo que alguno de los arcos eliminados. Como siempre se agrega el arco de menor costo entre (v_1, v_{k+1}) y (v_u, v_{k+1}), los segmentos que se debe reconectar son tres y no cuatro. Por lo tanto la cantidad de alternativas para la reconexión es 8 y no 48 como en el 4-opt original. En la Figura 2.9 se muestra todas las posibles maneras de reconectar una ruta, en el caso que $c_{v_1,v_{k+1}} \leq c_{v_u,v_{k+1}}$; las líneas punteadas representan caminos y las líneas llenas representan arcos.

En el mismo trabajo, se propone buscar movidas que mejoren la solución partiendo de $u = 1$ e incrementándolo en 1 cada vez que no se encuentren mejoras, hasta que $u = w$. Cuando eso ocurre, se vuelve a comenzar con $u = 1$ hasta realizar una ronda completa de búsquedas infructuosas, en cuyo caso se dice que la solución es 4-óptima*. Chequear si una solución es 4-óptima* requiere un tiempo $O(wn^2)$, por lo tanto utilizando valores de w que sean pequeños en comparación con n se obtiene un algoritmo con tiempo de ejecución $O(n^2)$.

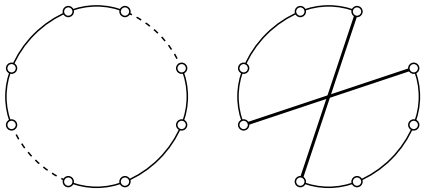


Figura 2.7: El único 2-intercambio posible para los arcos marcados

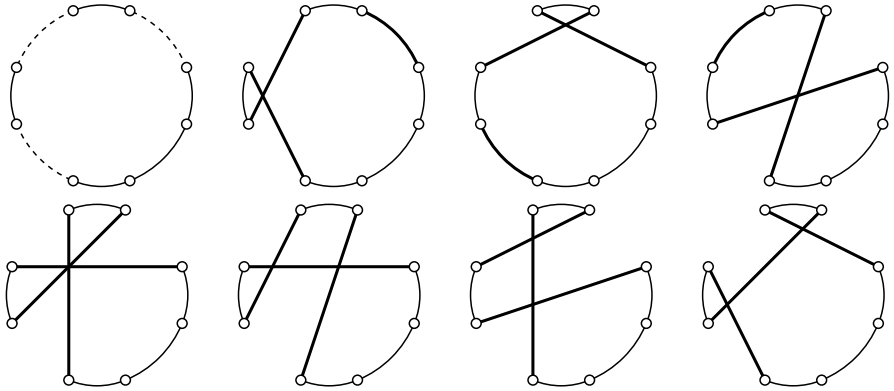


Figura 2.8: Todos los 3-intercambios posibles para los arcos marcados

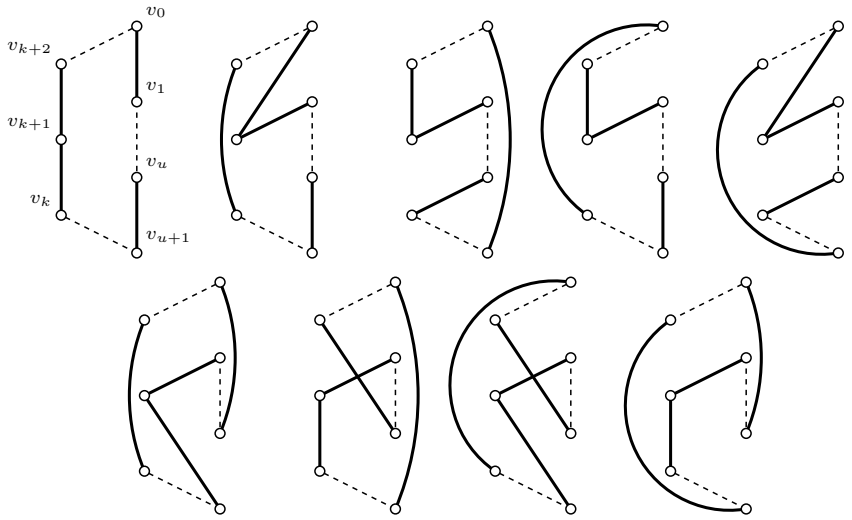


Figura 2.9: Todos los 4-intercambios* posibles para los arcos marcados si $c_{v_1,v_{k+1}} \leq c_{v_u,v_{k+1}}$.

2.6.2 El algoritmo de Lin-Kernigham

Un problema que surge al aplicar λ-intercambios es que debe fijarse el valor de λ de antemano. El algoritmo propuesto por Lin y Kernigham [38] utiliza la idea de intercambiar un conjunto de arcos por otro, pero determinando dichos conjuntos (y su cardinalidad) dinámicamente durante la ejecución del algoritmo. Dada una ruta, la idea es determinar dos conjuntos de arcos $\{x_1, \ldots, x_k\}$ e $\{y_1, \ldots, y_k\}$ tales que su intercambio disminuya el costo de la solución. Los arcos x deben ser parte de la ruta, ambos conjuntos deben ser disjuntos y, además, eliminar los arcos x y agregar los arcos y debe formar una ruta cerrada.

Se comienza seleccionando un arco $x_1 = (v_1, v_2)$. Luego, se busca $y_2 = (v_2, v_3)$ de modo que $c_{x_1} - c_{y_1} = c_{v_1,v_2} - c_{v_2,v_3} > 0$. Al comenzar la iteración i de este proceso, se han elegido $x_1, \ldots, x_{i-1}$ y $y_1, \ldots, y_{i-1}$, donde $x_h = (v_{2h-1}, v_{2h})$ y $y_h = (v_{2h}, v_{vh+1})$.

Se busca ahora $x_i = (v_{2i-1}, v_{2i})$, de modo que si se uniera v_{2i} con v_1 se obtendría una ruta cerrada (esto asegura que el proceso pueda finalizarse obteniendo una solución factible) . Como v_{2i-1} fue determinado al seleccionar y_{i-1} y x_i debe ser un arco de la ruta, existen dos opciones para v_{2i} (el anterior a v_{2i-1} en la ruta y el siguiente a él), pero solo una de estas cerraría la ruta a ser unida con v_1. De modo que x_i queda determinado por y_{i-1}.

Para seleccionar $y_i = (v_{2i}, v_{2i+1})$ debe cumplirse $y_i \notin \{x_1, \ldots, x_i\}$ para que los conjuntos sean disjuntos, $\sum_{j=1}^{i} c_{x_j} - c_{y_j} > 0$ para que el intercambio no empeore la solución y, además, debe poder elegirse el siguiente x_{i+1}. Si no es posible encontrar y_i que cumpla todo lo anterior, se busca k tal que $\sum_{j=1}^{k} c_{x_j} - c_{y_j}$ sea máximo y se realizan intercambian los arcos $\{x_1, \ldots, x_k\}$ e $\{y_1, \ldots, y_k\}$.

En la Figura 2.10 se muestra una posible aplicación del método para encontrar un intercambio. El procedimiento se repite hasta terminar en un óptimo local, cuando no es posible encontrar intercambios que mejoren la solución.

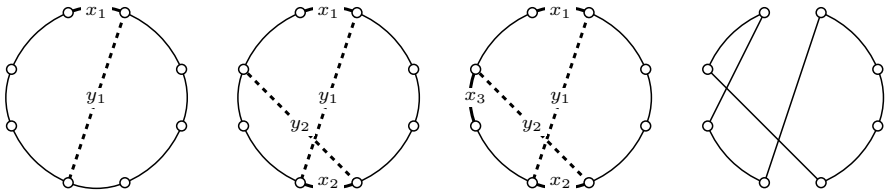


Figura 2.10: Una posible iteración del algoritmo de Lin-Kernigham.

2.6.3 El operador Or-opt

Una versión reducida del algoritmo 3-opt es el algoritmo Or-opt [39], que consiste en 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 2.11 se muestra una ruta y todas las posibles maneras de reubicar los 3 primeros clientes a la manera de Or-opt. Si una ruta visita n clientes existen $O(n^2)$ de estas movidas.

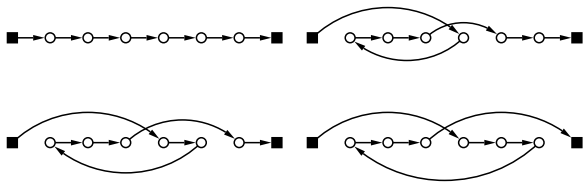


Figura 2.11: Movidas para reubicar los 3 primeros clientes de una ruta.

2.6.4 Operadores de Van Breedam

Van Breedam [40] propuso dos operadores para intercambiar clientes entre un par de rutas. En el operador *String Relocation*, una secuencia de m nodos es transferida de una ruta a la otra manteniendo el orden en la ruta original. En operador *String Exchange* una ruta envía una secuencia de m clientes a la otra y esta última envía otra secuencia de n clientes a la primera. Simbólicamente se denota con $(m, 0)$ a cada String Relocation y con (m, n) a cada String Exchange. En las Figuras 2.12(a)-(b) se muestra una movida SR $(2, 0)$ y en 2.12(c)-(d) se ilustra una movida SE $(2, 1)$.

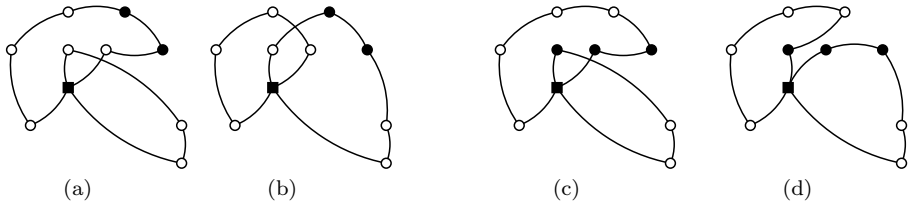


Figura 2.12: Movidas String Relocation y String Exchange.

2.6.5 GENI y GENIUS

Las inserciones generalizadas [41] (GENI, por *GENeralized Insertions*) surgen dentro de un método de solución del TSP y tienen como principal característica que la inserción de un cliente en una ruta no necesariamente ocurre entre dos nodos adyacentes. Si v_k es un nodo de la ruta, denotamos v_{k+1} a su sucesor y v_{k-1} a su predecesor. Se desea insertar un nodo v entre los nodos v_i y v_j (no necesariamente consecutivos).

En una inserción de Tipo I, se considera un nodo v_k en el camino de v_j a v_i ($v_k \neq v_i$ y $v_k \neq v_j$). La inserción consiste en eliminar los arcos (v_i, v_{i+1}), (v_j, v_{j+1}) y (v_k, v_{k+1}) y agregar los arcos (v_i, v), (v, v_j), (v_{i+1}, v_k) y (v_{j+1}, v_{k+1}). Es necesario además, invertir el sentido de los caminos $(v_{i+1}, \ldots, v_j)$ y $(v_{j+1}, \ldots, v_k)$. Una inserción de este tipo se muestra en la Figura 2.13, donde las líneas punteadas representan caminos y las líneas llenas representan arcos.

Para una inserción de Tipo II, debe seleccionarse v_k en el camino de v_j a v_i de modo que $v_k \neq v_j$ y $v_k \neq v_{j+1}$ y además v_l en el camino de v_i a v_j tal que $v_l \neq v_i$ y $v_l \neq v_{i+1}$. La inserción consiste, como se muestra en la Figura 2.14, en eliminar los arcos (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}) y (v_{k-1}, v_k), y agregar los arcos

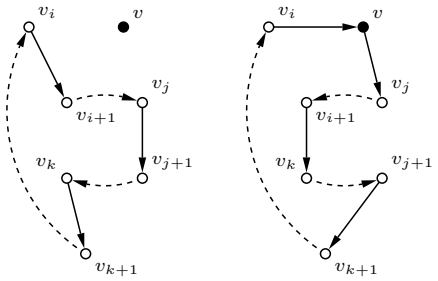


Figura 2.13: Inserción GENI de Tipo I.

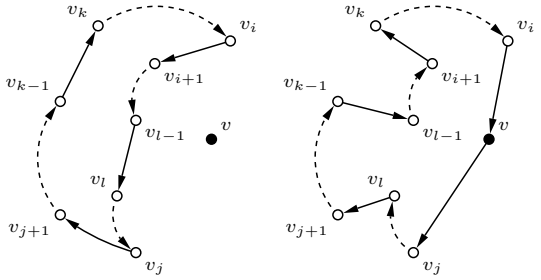


Figura 2.14: Inserción GENI de Tipo II.

(v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}) y (v_{i+1}, v_k). Además, debe revertirse el orden de los caminos $(v_{i+1}, \ldots, v_{l-1})$ y $(v_l, \ldots, v_j)$.

Dado que para evaluar las inserciones debe seleccionarse v_i, v_j, v_k y v_l, para lo cual hay $O(n^4)$ combinaciones, se propone disminuir las posibilidades mediante la definición de una p-vecindad. La p-vecindad del nodo v, $N_p(v)$, se define como el conjunto de los p nodos de la ruta más cercanos a v según los costos del problema (si la ruta tiene menos de p nodos, entonces todos pertenecen a $N_p(v)$). Fijado un valor para p, la búsqueda se restringe exigiendo que $v_i, v_j \in N_p(v)$, $v_k \in N_p(v_{i+1})$ y $v_l \in N_p(v_{j+1})$.

El algoritmo GENI para construir una ruta, inicializa la misma con 3 nodos al azar. Luego selecciona un nodo v que no esté en la ruta y realiza la mejor de todas las inserciones de Tipo I y Tipo II restringidas a las p-vecindades. El proceso se repite hasta que todos los nodos hayan sido visitados.

Dada una ruta, se definen dos procedimientos de inserción y remoción de clientes: el procedimiento *Stringing* consiste en realizar una inserción de GENI de Tipo I o Tipo II con p-vecindades y el *Unstringing* es el inverso de una inserción GENI de Tipo I o Tipo II. El algoritmo de post-optimización GENIUS comienza considerando el primer cliente de la ruta: se lo elimina mediante *Unstringing* y se lo vuelve a insertar utilizando *Stringing*. Esto podría incrementar el costo de la solución. Si la solución es mejorada, 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.

2.6.6 Transferencias cíclicas

Las transferencias cíclicas, introducidas por Thompson y Psaraftis [42], son movidas multi-ruta que intentan eliminar clientes de una ruta y reubicarlos en otra de manera cíclica. Si $R = \{r_1, \ldots, r_m\}$ es el conjunto de rutas de la solución y $\rho : R \to R$ es una permutación cíclica de algún subconjunto $R_b \subseteq R$ (con $|R_b| = b$), la transferencia de exactamente k clientes de cada ruta $r \in R_b$ a la ruta $\rho(r)$ es una k-transferencia b-cíclica. Cuando $R_b = R$ se habla de k-transferencias cíclicas.

Dado un valor de k, buscar una k-transferencia cíclica que mejore la solución actual, puede formularse como el problema de hallar un ciclo de costo negativo en un grafo dirigido $G = (V, A, W)$. Los nodos de G corresponden a conjuntos de k clientes que están en una misma ruta de la solución: $V = \{s \mid s \subset r_i, |s| = k, r_i \in R\}$. El conjunto de arcos $A = \{(s_i, s_j) \mid s_i \subset R_i, s_j \subset R_j, R_i \neq R_j, R_j \cup s_i \setminus s_j$ es factible$\}$ indica las transferencias factibles. El arco (s_i, s_j) representa la transferencia que elimina los clientes s_i de R_i, los agrega a R_j y elimina s_j de R_j. El costo de un arco (s_i, s_j) es el incremento del costo en la ruta R_j si se realiza la transferencia, es decir, $w(s_i, s_j) = t(R_j \cup s_i \setminus s_j)$ donde $t(S)$ es el costo de un TSP óptimo sobre el conjunto de clientes S.

Todo ciclo en G tal que sus nodos corresponden a conjuntos de clientes en rutas diferentes, representa una k-transferencia cíclica. Dicha transferencia mejora la solución si el peso del ciclo correspondiente es negativo. Encontrar un ciclo de costo negativo con las características deseadas es NP-Hard [42]. Incluso G no puede ser construido en tiempo polinomial: $|A|$ es $O(m^2 n^{2k})$ y calcular $w(s_i, s_j)$ implica resolver un TSP. En el artículo se da un método para aproximar $w(s_i, s_j)$ en tiempo polinomial y una heurísitca para buscar ciclos de costo negativo que pasan por subconjuntos de rutas diferentes.

Capítulo 3

Extensiones de las heurísticas clásicas

Las heurísticas presentadas en el Capítulo 2 surgen con el objetivo de resolver el VRP. En esta sección se muestran algunos trabajos que extienden dichas heurísticas para contemplar algunas restricciones adicionales como ventanas de tiempo y flota heterogénea.

3.1 Algoritmo de Ahorros

Solomon (VRPTW)

Solomon [43] propuso una extensión del Algoritmo de Ahorros (ver Sección 2.1) para contemplar problemas con ventanas de tiempo. Determinar si la unión de dos rutas r_i y r_j es factible no resulta directo en presencia de ventanas de tiempo, pues el arribo a los clientes de r_j puede atrasarse y esto podría ocasionar que se violaran algunas ventanas de tiempo.

Dada una ruta factible $(v_0, \ldots, v_{m+1})$ con $v_0 = v_{m+1} = 0$, llamamos b_i y w_i al tiempo de arribo al nodo v_i $(0 \leq i \leq m+1)$. El mínimo tiempo de espera en el nodo v_i $(i \neq 0)$ es $w_i = \max\{0, e_i - (b_{i-1} + s_{i-1} + t_{i-1,i})\}$. Si el tiempo de arribo al nodo v_i se incrementa en PF_i, el tiempo de arribo a los nodos posteriores v_j (con $i < j \leq m+1$) se incrementará en $\mathrm{PF}_j = \max\{0, \mathrm{PF}_{j-1} - w_j\}$. El valor PF recibe el nombre de *push forward* y mide cuánto se desplaza el tiempo de arribo respecto a su valor original. Dado que el Push Forward en un nodo sólo depende de sus predecesores, es posible determinar si un incremento en el tiempo de arribo a un cliente mantiene la factibilidad de la ruta, mediante una recorrida por los clientes posteriores. Para cada uno de ellos se calcula el nuevo tiempo de arribo (que depende solamente del cliente anterior) y se verifica si está dentro de la ventana de tiempo. Puede observarse que los valores del *push forward* son no crecientes a lo largo de la ruta. La búsqueda puede terminarse antes si se encuentra algún nodo con *push forward* igual a 0, pues para todos los nodos posteriores a él también será igual a 0, y como la ruta original era factible la nueva también lo será. Este procedimiento es agregado al Algoritmo de Ahorros original para determinar, de manera eficiente, si la unión de dos rutas es factible.

Utilizando las definiciones originales del ahorro (2.1 o 2.2) no se considera la dimensión temporal del problema. Esto puede llevar a unir rutas muy cercanas en cuanto a distancias pero con ventanas de tiempo muy diferentes, provocando grandes tiempos de espera. Si el máximo ahorro se da en s_{ij} pero e_j es mucho mayor que $b_i + t_{ij}$, en la nueva ruta el tiempo de espera en j será muy elevado. Dado que no se considera la posibilidad de insertar clientes entre i y j, otro vehículo deberá visitar clientes que podrían haber sido visitados en ese tiempo ocioso, resultando en soluciones que utilizan una gran cantidad de vehículos. Para resolver este problema, en lugar de modificar la definición de s_{ij}, se fija un valor máximo W para los tiempos de espera y no se realizan las uniones que ocasionen tiempos de espera mayores que W.

Golden, Assad, Levy y Gheysens (FSMVRP)

En esta propuesta se modifica la definición del ahorro para incorporar los costos fijos de los diferentes vehículos [8]. En cada iteración se tiene un conjunto de rutas, cada una de las cuales cubre cierta demanda z y está asignada al vehículo capaz de cargar dicha demanda que tenga el menor costo fijo. Notamos con $F(z)$ al costo fijo del vehículo más barato cuya capacidad sea no menor que z y con $P(z)$ a la capacidad de dicho vehículo.

Si dos rutas (una con el cliente i al final y otra con el cliente j al comienzo) con demandas respectivas z_i y z_j son combinadas, el ahorro en términos de costos fijos es $F(z_i) + F(z_j) - F(z_i + z_j)$. Se define, entonces, el *Ahorro Combinado* (CS, por sus siglas en inglés) como

$$\overline{s}_{ij} = s_{ij} + F(z_i) + F(z_j) - F(z_i + z_j) \tag{3.1}$$

donde s_{ij} es el ahorro original (2.1 o 2.2). Es decir, se suma el ahorro por distancias y el ahorro por costos fijos. Este enfoque es una extensión directa del algoritmo original y toma en cuenta sólamente el ahorro de una iteración a la siguiente. Se proponen dos medidas adicionales que toman en cuenta el costo de oportunidad de utilizar un vehículo más grande al unir dos rutas.

Luego de unir las rutas, la capacidad libre del vehículo utilizado para recorrer la nueva ruta será $P(z_i + z_j) - z_i - z_j$. En el *Ahorro de Oportunidad Optimista* (OOS por sus siglas en inglés), se supone (de manera optimista) que dicha capacidad estará colmada al final del algoritmo. Por lo tanto, al unir las rutas se ahorra, además, el costo fijo del vehículo más barato que pueda transportar dicha carga. El OOS es entonces:

$$s^*_{ij} = \overline{s}_{ij} + F\left(P\left(z_i + z_j\right) - z_i - z_j\right) \tag{3.2}$$

Los autores reportan que el método basado en CS desfavorece la combinación de rutas, mientras que el OOS las combina excesivamente. Una tercer medida llamada *Ahorro de Oportunidad Realista* (ROS por sus siglas en inglés), surge como solución intermedia. Esta medida incluye el término del ahorro de oportunidad sólo cuando la nueva ruta requiere un vehículo de mayor capacidad que los que utilizan las dos rutas por separado, es decir, cuando $w = P(z_i + z_j) - \max\{z_i, z_j\} > 0$. Además, considera como ahorro de oportunidad, al costo del vehículo más caro que no puede satisfacer la demanda de la nueva ruta. Siendo $F'(z)$ el costo fijo del vehículo más caro que no puede cargar una demanda de z, se define

$$s'_{ij} = \overline{s}_{ij} + \delta(w)F'\left(P\left(z_i + z_j\right) - z_i - z_j\right) \tag{3.3}$$

donde $\delta(0) = 0$ y $\delta(w) = 1$ si $w > 0$. Al igual que en el algoritmo original, puede agregarse un parámetro de forma.

Una desventaja de cualquiera de estas propuestas es que incorporan datos de la demanda de las rutas en la definición del ahorro y por lo tanto algunos ahorros deben ser recalculados en cada iteración del algoritmo.

3.2 Heurísticas de Inserción Secuencial

Solomon (VRPTW)

Solomon [43] propone varias heurísticas de inserción secuencial para resolver el VRPTW. Todas se basan en extensiones de las funciones de costo para incorporar tanto las distancias como el tiempo. Se define b_j y w_j al igual que en la Sección 3.1.

Por un lado, se define una heurística de inserción más cercana. Si se tiene una ruta $(0, \ldots, v_i, 0)$, se define el costo de insertar el cliente v_j a continuación de v_i en la ruta como

$$\hat{c}_{ij} = \delta_1 c_{ij} + \delta_2 T_{ij} + \delta_3 V_{ij} \tag{3.4}$$

$$T_{ij} = b_j - (b_i + s_i) \tag{3.5}$$

$$V_{ij} = l_j - (b_i + s_i + t_{ij}) \tag{3.6}$$

donde los parámetros δ_1, δ_2 y δ_3 son no negativos y suman 1. El valor de T_{ij} indica la diferencia entre la hora de comienzo del servicio en j y la del fin del servicio en i, midiendo la cercanía de los clientes en términos temporales. Por otro lado, V_{ij} mide la urgencia de realizar la inserción como la diferencia entre la hora de arribo a j (sin incluir la espera) y la última hora a la que se podría arribar a dicho cliente. Valores de V_{ij} cercanos a 0 indican que dentro de "pocas" iteraciones, el cliente j no podrá ser insertado en esta ruta.

El algoritmo selecciona el cliente inicial de una ruta como el más cercano al depósito (según la medida $\hat{c}$) dentro de los clientes no visitados. En cada paso se selecciona al cliente no visitado que sea más cercano al último cliente de la ruta (considerando solamente las inserciones que factibles) y se lo agrega al final. Cuando no hay más clientes para insertar en la ruta actual, se crea una nueva o se termina si todos los clientes han sido visitados.

Por otro lado, se definen 3 heurísticas más elaboradas que responden a los mismos criterios genéricos. Se tiene la ruta $(v_0, \ldots, v_{t+1})$ con $v_0 = v_{t+1} = 0$. Al igual que en la heurística de Mole & Jameson (ver Sección 2.2.1), se utilizan dos medidas para decidir el próximo cliente a insertar en una ruta: $c_1(v_i, w)$ considera la ventaja de agregar el cliente no visitado w entre los clientes consecutivos v_i y v_{i+1} y $c_2(v_i, w)$ indica la urgencia de realizar dicha inserción. El algoritmo es esencialmente el mismo (salvo porque no se aplica ninguna post-optimización a las rutas): se crea una nueva ruta, se selecciona un cliente a insertar según c_2 y se inserta según c_1, repitiendo el proceso hasta que no haya inserciones factibles.

Se proponen tres alternativas para seleccionar el primer cliente de la ruta: el más lejano al depósito, el cliente cuya ventana de tiempo termine antes y el que minimice la suma del tiempo y distancia al depósito. Para las medidas c_1 y c_2 también se realizan tres propuestas.

1. En esta propuesta se considera la ventaja de visitar al cliente dentro de la ruta parcial y no en una ruta específica para él. En este caso tenemos:

$$c_1(v_i, w) = \alpha_1 c_{11}(v_i, w) + \alpha_2 c_{12}(v_i, w) \tag{3.7}$$
$$c_{11}(v_i, w) = d_{iw} + d_{w,i+1} - \mu d_{i,i+1} \tag{3.8}$$
$$c_{12}(v_i, w) = b^w_{i+1} - b_{i+1} \tag{3.9}$$
$$c_2(v_i, w) = \lambda d_{0w} - c_1(v_i, w) \tag{3.10}$$

   donde b^w_{i+1} indica el tiempo de arribo al nodo $i+1$ cuando w es insertado en la ruta y d_{ij} es la distancia entre los nodos i y j. Los parámetros verifican $\alpha_1, \alpha_2, \mu, \lambda \geq 0$ y $\alpha_1 + \alpha_2 = 1$.

   Para calcular la mejor inserción (3.7) se tiene en cuenta dos factores. Por un lado (3.8), se mide el ahorro en la distancia si se insertara w entre v_i y v_{i+1}. Además (3.9), se tiene en cuenta el retardo que provoca insertar el cliente en la ruta. Finalmente, para seleccionar el próximo cliente a insertar se considera (3.10) tanto el valor c_1 como la distancia del cliente al depósito, procurando privilegiar a los clientes para los que sería demasiado crear una ruta individual.

2. En este caso se intenta seleccionar el cliente que, si fuera insertado en la ruta, minimiza una medida de la distancia y el tiempo total de la misma. Aquí, c_1 es igual que en la parte anterior y

$$c_2(v_i, w) = \beta_1 R_d(w) + \beta_2 R_t(w)$$

   donde $R_d(w)$ y $R_t(w)$ son el tiempo y la distancia total de la ruta si w es insertado entre v_i y v_{i+1}. En este caso, el cliente seleccionado es el que minimiza c_2, contrariamente a los demás casos (en que se busca maximizar la medida c_2). Los parámetros verifican $\beta_1 \geq 0$, $\beta_2 > 0$ y $\beta_1 + \beta_2 = 1$.

3. En la última propuesta se define

$$c_1(v_i, w) = \alpha_1 c_{11}(v_i, w) + \alpha_2 c_{12}(v_i, w) + \alpha_3 c_{13}(v_i, w) \tag{3.11}$$
$$c_{13}(v_i, w) = l_w - b_w \tag{3.12}$$
$$c_2(v_i, w) = c_1(v_i, w) \tag{3.13}$$

   siendo c_{11} y c_{12} los de la propuesta 1. Se agrega un nuevo término a c_1 (3.11), que mide que tan cerca del final de su ventana de tiempo se arribaría al nodo w si éste fuera insertado en la ruta (3.12). Finalmente, se utiliza c_1 para seleccionar el próximo cliente a insertar (3.13).

En el trabajo se experimenta con varias combinaciones de los criterios de inicialización y de selección propuestos.

Dullaert, Janssens, Sorensen y Vernimmen (FSMVRPTW)

Dullaert, Janssens, Sorensen y Vernimmen [44] extienden las ideas de los algoritmos de Inserción Secuencial de Solomon para resolver el FSMVRP, que es una variante del FSMVRP en la que se incluyen ventanas de tiempo en los clientes.

El algoritmo comienza creando una nueva ruta para el vehículo de menor capacidad. Dicha ruta puede inicializarse con el cliente más lejano al depósito o con el cliente cuya ventana de tiempo finalice antes. Para cada cliente no visitado se define c_1 utilizando 3.11, 3.8 y 3.9 junto con un término c_{13} que incorpora las costos fijos de los vehículos. Para dicho valor, se adaptan las ideas de Ahorros Combinados (CS), Ahorros de Oportunidad Optimista (OOS) y Ahorros de Oportunidad (ROS) Realista de Golden et al. (ver Sección 3.1). Originalmente estas medidas están definidas para el caso en que se unen dos rutas diferentes y no para inserciones de un cliente en una ruta.

Se denota como Q y $\overline{Q}$ la carga y la capacidad del vehículo, respectivamente. Dada una posible inserción, Q^{new} y $\overline{Q^{\text{new}}}$ representan la nueva carga y la potencialmente nueva capacidad del vehículo, respectivamente. Al Ahorro Combinado Adaptado (ACS) es el ahorro inmediato en costos fijos, es decir $F(Q^{\text{new}}) - F(Q)$. Al Ahorro de Oportunidad Optimista Adaptado (AOOS) suma a ACS el costo fijo del menor vehículo capaz de transportar la capacidad libre del vehículo, $F(\overline{Q^{\text{new}}} - Q^{\text{new}})$. Finalmente, el Ahorro de Oportunidad Realista Adaptado (AROS), agrega a ACS el costo fijo del vehículo más caro que no puede transportar la capacidad libre: $F'(\overline{Q^{\text{new}}} - Q^{\text{new}})$.

Para decidir el próximo cliente a insertar en la ruta se utiliza la medida c_2, definida como:

$$c_2(v_i, w) = \lambda(d_{0w} + t_{0w}) + s_w + F(q_w) - c_1(v_i, w)$$

siendo q_w la demanda del cliente w. Esta medida cuantifica el beneficio de incluir a w en la ruta en lugar de utilizar una ruta nueva para el solo.

3.3 Heurísticas de Inserción en Paralelo

Potvin y Rousseau (VRPTW)

La heurística para el VRPTW propuesta por Potvin y Rousseau [22] es una versión Paralela de la heurística de Inserción Secuencial de Solomon (ver Sección 3.2).

La cantidad de rutas de la solución se decide al comienzo y no se altera durante la ejecución del algoritmo. Para decidir cuántas rutas crear, se ejecuta una de las Heurísticas de Inserción Secuencial dada por Solomon y se crea una nueva solución con la cantidad de rutas de la solución obtenida. Además, de cada una de las rutas dadas por el algoritmo de Solomon, se selecciona el cliente más lejano al depósito como cliente inicial de una ruta en la nueva solución. De esta manera, el algoritmo de Solomon se utiliza para decidir cuántas rutas iniciales crear y también para inicializar dichas rutas. Este enfoque es similar al de Christofides et al. (ver Sección 2.2.2).

En cada iteración se debe decidir qué cliente agregar a la solución y dónde ubicarlo (eso implica decidir la ruta y la posición dentro de la ruta). Al igual que en los algoritmos de Mole & Jameson (ver Sección 2.2.1) y de Solomon, se utilizan dos medidas para decidir qué cliente insertar y dónde insertarlo. Se define la medida $c_{1r}(v_i, w)$ es similar a la c_1 de los algoritmos de Solomon:

$$\begin{aligned} c_{1r}(v_i, w) &= \alpha_1 c_{11r}(v_i, w) + \alpha_2 c_{12r}(v_i, w) \\ c_{11r}(v_i, w) &= d_{ik} + d_{kj} - \mu d_{ij} \\ c_{12r}(v_i, w) &= b_{j_k} - b_j \end{aligned}$$

es decir, la suma ponderada del incremento en distancia y el retardo provocados por la inserción del cliente w entre los clientes consecutivos v_i y v_{i+1} en la ruta r. Notar que hay un valor de la medida para cada ruta. Luego, para cada cliente w se calcula la mejor alternativa $c^*_{1r^*}(v^*_i, w) = \min c_{1r}(v_i, w)$, donde el mínimo se toma de todas las posibles rutas r y todos los pares de nodos consecutivos en dicha ruta.

Finalmente, para decidir cuál de todas las inserciones realizar, se calcula, para cada cliente

$$c_2(w) = \sum_{r \neq r^*} (c_{1r}(v_i, w) - c^*_{1r^*}(v^*_i, w))$$

Esta expresión mide la urgencia de insertar al cliente w en la solución como la diferencia del costo de la mejor alternativa con todas las demás. Es una generalización de la medida utilizada en la Heurística de Inserción en Paralelo de Christofides et al. (ver Sección 2.2.2). Valores elevados de $c_2(w)$ indican una gran diferencia entre el costo la mejor opción y el costo de insertarlo en otras rutas. Estos clientes deben considerarse primero, pues la cantidad de alternativas interesantes para ellos es reducida. Por lo tanto, se calcula $w^* = \arg\max_w c_2(w)$ y ese cliente es insertado en la ruta r^*, en la posición dada por c_{1r^*}.

La medida c_1 toma un valor muy grande L si la inserción del cliente en la ruta no es factible. Si para el cliente w hay N_w rutas en las que puede ser insertado (con $0 \leq N_w \leq N$), el valor de $c_2(w)$ será cercano a $(N - N_w)L$, pues $N - N_w$ términos de la suma serán cercanos a L. Entonces, siempre que para dos clientes w y w' se cumpla que $N_w < N_{w'}$, el cliente w será preferible al w'. Es decir, con esta medida tienen prioridad los clientes con menos alternativas siempre que L sea lo suficientemente grande. Para dos clientes con igual cantidad de alternativas, la medida discrimina por los costos de inserción.

Los únicos parámetros del algoritmo son α_1 y α_2 en la definición de c_{11r}. Además de variar los parámetros, se ejecuta el algoritmo varias veces buscando soluciones con menos rutas. Para eso, el valor de N se decrementa en 1 y el procedimiento se repite hasta que no se encuentren soluciones factibles. Cuando el valor de N se decrementa en 1, debe decidirse cuál de los clientes iniciales se elimina. El criterio utilizado es el de eliminar uno de los clientes más cercanos al resto de los clientes de la solución inicial.

3.4 Algoritmo de Barrido

Solomon (VRPTW)

Solomon también propuso una extensión del Algoritmo de Barrido para el VRPTW [43]. Las restricciones temporales no se consideran al construir los sectores, sino al diseñar las rutas una vez que éstos han sido construidos.

En cada paso se crea un sector del mismo modo que en el algoritmo original, teniendo en cuenta solamente la restricción de capacidad. Sobre los clientes de ese sector se aplica un algoritmo de Inserción Secuencial de Solomon (ver Sección 3.2) para se construir una sola ruta. Dado que la ruta debe respetar las ventanas de tiempo, algunos clientes del sector pueden quedar fuera de ella. Los clientes visitados por la ruta construida se eliminan del problema y los otros se mantienen y serán considerados en futuras iteraciones.

Para continuar con el proceso debe decidirse el cliente a partir del cual se formará el siguiente sector. Asumiendo que el giro de la recta se realiza en sentido anti-horario, se divide al último sector en 2 y se selecciona el cliente que forme el menor ángulo con la recta que bisecta el sector original. Con esto se pretende asegurar que los clientes que quedaron sin visitar estén en sectores diferentes en próximas iteraciones.

3.5 Rutear Primero - Asignar Después

Golden, Assad, Levy y Gheysens (FSMVRP)

Golden et al. [8] dan una extensión directa del método Rutear Primero - asignar después (ver Sección 2.4) para FSMVRP. En la versión original del algoritmo, cada arco (r_i, r_j) representa la ruta $(0, r_{i+1}, \ldots, r_j, 0)$ y se pondera con la distancia recorrida por dicha ruta (ver Ecuación 2.9). En la versión para FSMVRP se considera, además de la distancia, el costo del menor vehículo capaz que recorrer la ruta:

$$w(r_i, r_j) = c_{0,r_{i+1}} + \sum_{k=i+1}^{j-1} c_{r_k,r_{k+1}} + c_{r_j,0} + F\left(\sum_{k=i+1}^{j} d_{r_k}\right)$$

Luego, al igual que en el algoritmo original, se halla el camino de costo mínimo entre 0 y r_n para obtener las rutas de la solución. Los mismos autores proponen un procedimiento similar a éste, que permite considerar más particiones de la ruta original incorporando varias copias del depósito en el grafo auxiliar.

Capítulo 4

Metaheurísticas

Para obtener mejores soluciones que las heurísticas presentadas en el Capítulo 2, es necesario recurrir a técnicas que realicen una mejor exploración del espacio de soluciones. Las Metaheurísticas son procedimientos genéricos de exploración del espacio de soluciones para problemas de optimización y búsqueda. Proporcionan una linea de diseño que, adaptada en cada contexto, permite generar algoritmos de solución. En general, las metaheurísticas obtienen mejores resultados que las heurísticas clásicas, pero incurriendo en mayores tiempos de ejecución (que de todos modos, son inferiores a los de los algoritmos exactos).

En este capítulo se reportan los resultados más significativos en la aplicación de Algoritmos de Hormigas, Búsqueda Tabú y Algoritmos Genéticos para resolver el VRP y el VRPTW. Los métodos seleccionados son representantes de tres paradigmas diferentes. Los Algoritmos de Hormigas son procedimientos basados en agentes que utilizan métodos constructivos aleatorizados y cooperan entre sí compartiendo información. Los algoritmos de Búsqueda Tabú son métodos de búsqueda local que aceptan empeorar las soluciones para escapar de los óptimos locales. Los Algoritmos Genéticos se basan en mantener un conjunto de soluciones lo suficientemente diverso como para cubrir gran parte del espacio de soluciones.

4.1 Algoritmos de Hormigas

Los Sistemas de Hormigas o *Ant Systems* 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 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 Algoritmos de Hormigas se simula el comportamiento de una colonia de estos animales. Cada hormiga construye una solución combinando un cri-

terio ávido que le indica que tan bueno *parece ser* tomar cierta decisión, y la información histórica (bajo la forma de feromona) que le indica que tan bueno *fue* tomar dicha decisión. El primer problema al que se aplicó esa metodología fue el TSP [45]. Las decisión que toma una hormiga en cada paso es elegir la próxima ciudad a visitar. El criterio ávido, llamado *visibilidad*, está dado por una medida $\eta_{ij} = 1/c_{ij}$, que indica qué tan bueno es moverse al nodo j estando en el nodo i. Además, en la iteración t del algoritmo, cada arco (i, j) del problema tiene asociada una cantidad de feromona $\tau_{ij}(t)$.

En el modelo *ANT-Cycle* [45], M hormigas se distribuyen entre los nodos (en general se coloca una hormiga en cada nodo, siendo $M = n$). En la iteración t, cada hormiga construye una solución seleccionando nodos de acuerdo a una regla probabilística que combina η_{ij} y $\tau_{ij}(t)$. Si está ubicada en el nodo i, la hormiga se mueve al nodo j con probabilidad

$$p_{ij}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in \Omega} [\tau_{ih}(t)]^\alpha [\eta_{ih}]^\beta} & \text{si } j \in \Omega \\ 0 & \text{si } j \notin \Omega \end{cases} \tag{4.1}$$

siendo Ω el conjunto de nodos aún no visitados. 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 una solución se actualiza la feromona en cada arco (i, j) según la siguiente regla

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \Delta\tau_{ij} \tag{4.2}$$

donde $\Delta\tau_{ij} = \sum_{k=1}^{M} \Delta\tau_{ij}^k$ y $\Delta\tau_{ij}^k$ indica la feromona depositada por la k-ésima hormiga en dicho arco. Si la solución encontrada por la k-ésima hormiga tiene longitud L_k y utiliza el arco (i, j), entonces $\Delta\tau_{ij}^k = Q/L_k$ (Q es un parámetro del algoritmo); en otro caso $\Delta\tau_{ij}^k = 0$.

El parámetro $\rho \in [0, 1]$, llamado *coeficiente de evaporación*, establece el porcentaje de feromona que permanece de una iteración a otra y surge con el objetivo principal de evitar la convergencia prematura del algoritmo.

Algoritmo Ant System para TSP
Paso 1 (inicialización). Colocar una hormiga en cada nodo. Hacer $t := 0$ y $\tau_{ij}(t) := \tau_0 \, \forall (i, j) \in E$
Paso 2 (construcción). Para cada hormiga, construir una solución de acuerdo a la regla probabilística 4.1.
Paso 3 (actualización). Actualizar la feromona en cada arco según 4.2.
Paso 4 (terminación). Hacer $t := t + 1$. Si $t < T^{\max}$ colocar una hormiga en cada nodo e ir a 2. Si no, terminar.

Notar que las probabilidades p_{ij} no dependen de cuál sea la ruta parcial y recorrida por la hormiga, por lo tanto, solo es necesario calcularlas al comienzo de cada iteración. Cada hormiga tarda un tiempo $O(n^2)$ en construir una solución y al utilizar n hormigas, el tiempo de ejecución de una iteración del algoritmo es $O(n^3)$. Si se trabaja con *conjuntos de candidatos* puede reducirse este tiempo a $O(n^2)$. Para cada nodo i se define un conjunto de nodos candidatos $C(i)$ y la probabilidad de la fórmula 4.1 se calcula sobre $\Omega \cap C(i)$. Si la

cardinalidad de los conjuntos de candidatos es una constante γ, una iteración del algoritmo tarda un tiempo $O(\gamma n^2)$.

El algoritmo presentado es un esquema básico sobre el que se han desarrollado diversas variantes. A continuación se presentan algunas de estas variantes para resolver el TSP.

Ant System con Selección Elitista

Una de las mejoras propuestas para el algoritmo Ant System es la introducción de una estrategia elitista para la actualización de la feromona [46]. Esta idea consiste en dar más énfasis a la mejor solución encontrada en cada iteración. Si L^* es el largo de la mejor solución encontrada en la iteración t, la fórmula de actualización de feromona en el arco (i, j) es ahora

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} + \Delta\tau^*_{ij} \tag{4.3}$$

donde $\Delta\tau^*_{ij} = \sigma\frac{Q}{L^*}$ si el arco (i, j) es parte de la mejor solución y $\Delta\tau^*_{ij} = 0$ en otro caso. La idea es actualizar la feromona como si σ hormigas (llamadas *hormigas elitisitas*) hubieran encontrado la mejor solución.

Ant System con Selección Elitista y Ranking

La idea de la Selección Elitista, que da más énfasis a la mejor solución encontrada en una iteración, puede generalizarse dando lugar a la *Selección por Ranking* [47]. Antes de actualizar la feromona, se ordena a las hormigas según el largo de la solución obtenida ($L_1 \leq \ldots \leq L_M$). Solamente las $\sigma - 1$ primeras hormigas colocan feromona en los arcos, donde σ es la cantidad de hormigas elitistas. Además, la feromona aportada por la hormiga en la posición μ es ponderada por $\sigma - \mu$. De este modo, la regla de actualización de feromona es la dada en 4.3, pero $\Delta\tau_{ij} = \sum_{\mu=1}^{\sigma-1}\Delta\tau^{\mu}_{ij}$ y $\Delta\tau^{\mu}_{ij} = (\sigma-\mu)\frac{Q}{L_\mu}$ si la μ-ésima hormiga utiliza el arco (i, j) y $\Delta\tau^{\mu}_{ij} = 0$ si no.

Ant-Q

En Ant-Q [48] la próxima ciudad a visitar por una hormiga que está en el nodo i es $j = \arg\max_{j\in\Omega}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$ con probabilidad p y la dada por 4.1 con probabilidad $1 - p$. Es decir, con cierta probabilidad se elige la mejor ciudad (según la combinación de feromona y la visibilidad) y si no, se aplica la regla probabilística del Ant System dada en 4.1.

La feromona sólo se actualiza en los arcos recorridos por alguna hormiga (en el Ant Systems, la evaporación se aplica a todos los arcos). La actualización se realiza cuando todas las hormigas han construido su solución y consta de una fase de actualización local y otra de actualización global. En la actualización local de feromona, cada hormiga modifica la feromona de los arcos (i, j) que ha recorrido, según

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\max_{h\in\Omega}\tau_{jh}(t)$$

En la actualización global de feromona, la hormiga que obtuvo la mejor solución (de costo L_{best}), actualiza la feromona en los arcos (i, j) de dicha solución según

$$\tau_{ij}(t) := (1-\rho)\tau_{ij}(t) + \rho W/L_{\text{best}}$$

Ant Colony System (ACS)

El Ant Colony System [49] es un refinamiento del Algoritmo Ant-Q. La regla de actualización local se modifica por

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\tau_0$$

donde τ_0 es un parámetro del algoritmo y en la de actualización global se utiliza siempre $W = 1$. En la variante *Iteration Best* la hormiga que realiza la actualización global es la que obtuvo la mejor solución en la iteración, mientras que en *Global Best* la realizar la que obtuvo la mejor solución considerando toda la ejecución del algoritmo.

MAX-MIN Ant System (MMAS)

El MAX-MIN Ant System [50] es un refinamiento del Ant System que busca una mayor explotación de las mejores soluciones encontradas así como un mecanismo para evitar la convergencia prematura.

Solamente una hormiga se utiliza para actualizar la feromona, pudiendo elegirse la mejor de la iteración o la mejor de toda la ejecución, al igual que en ACS. La regla para actualizar las feromonas es

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}^{\text{best}}$$

donde $\Delta\tau_{ij}^{\text{best}} = 1/L_{\text{best}}$, y L_{best} es el costo de la mejor solución.

Si para todos los nodos i, existe un nodo j que cumple que τ_{ij} es demasiado grande en comparación con τ_{ih} para todo $h \neq j$, es muy probable que el algoritmo converja prematuramente pues probablemente siempre se seleccione el (i, j) para abandonar el nodo i. En MMAS, se establece una cota superior y una inferior para la feromona en cada arco. De este modo, $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max}\ \forall(i,j) \in E$. Si en alguna actualización la feromona queda fuera de ese intervalo, se le asigna el valor $\tau_{\min}$ o $\tau_{\max}$ según corresponda. En el artículo se da un método para determinar analíticamente los valores de dichos límites.

Al iniciar el algoritmo se coloca la máxima cantidad posible de feromona en cada arco. De este modo, por medio de la evaporación, los arcos que no participan en buenas soluciones decrementan su feromona en un factor ρ. En este caso, ρ puede ser interpretado como una *tasa de aprendizaje*.

Finalmente, se propone que algunas hormigas mejoren la solución obtenida mediante algún operador de búsqueda local. Una opción es que a la mejor solución obtenida en cada iteración se le aplique el algoritmo 2-opt. La otra opción es, en lugar de utilizar n hormigas, reducir la cantidad y permitir que todas apliquen dicho operador en cada iteración. Para las pruebas realizadas, la segunda opción dió los mejores resultados.

4.1.1 Ant System Híbrido para VRP

En la primera propuesta de Bullnheimer, Hartl y Strauss [51] la construcción de la solución se realiza según la regla probabilística dada por la Ecuación 4.1, pero sólo se considera la inclusión de un nodo si no viola las restricciones del problema. Por lo tanto, el conjunto Ω contiene a los nodos no visitados que

pueden ser insertados en la ruta actual manteniendo la factibilidad. Cuando $\Omega = \phi$ la hormiga vuelve al depósito y comienza de nuevo la construcción sobre los clientes aún no visitados. El algoritmo consiste en un Ant System con Selección Elitista en el que cada hormiga aplica el algoritmo 2-opt al finalizar la construcción de su solución.

En el mismo artículo se propone modificar la regla de selección del próximo nodo, incorporando más información acerca del problema. La nueva regla es

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

donde $\mu_{ij} = c_{i0} + c_{0j} - c_{ij}$ es la medida de Ahorro de Clarke & Wright (ver Sección 2.1) y $\kappa_{ij} = (Q_i + q_j)/Q$ es una medida de la utilización del vehículo si j se agregara a la ruta (siendo Q_i la demanda acumulada en la ruta parcial construida). Como μ y κ dependen de la ruta parcial, las probabilidades deben calcularse en cada paso de la construcción de cada solución, lo que implica altos costos en términos computacionales.

Los mismos autores refinaron el algoritmo obteniendo mejores resultados [52]. Por un lado, se utiliza la regla probabilística original dada por la Ecuación 4.1 y se incorpora a la visibilidad una medida del ahorro: $\eta_{ij} = c_{i0} + c_{0j} - gc_{ij} + f|c_{i0} - c_{0j}|$. Con esta definición, las probabilidades no necesitan calcularse en cada paso, lo que disminuye la complejidad respecto a la versión anterior del algoritmo. Se definen conjuntos de candidatos, de modo que para cada cliente sólo se consideran sus $\lfloor n/4 \rfloor$ vecinos más cercanos. En el caso que ninguno de éstos pueda ser visitado, se regresa al depósito.

Además, se utiliza una Selección Elitista con Ranking y cada hormiga aplica el algoritmo 2-opt al finalizar la construcción de su solución, al igual que en la versión anterior del algoritmo.

4.1.2 MACS para VRPTW

MACS (Multiple Ant Colony System) es un algoritmo basado en el Ant System para resolver el VRPTW [53]. Se trabaja con dos colonias de hormigas, una de ellas (ACS-VEI) tiene como objetivo minimizar la cantidad de rutas utilizadas y la otra (ACS-TIME) minimizar la longitud total de la solución. Las dos colonias evolucionan en paralelo y cuando alguna encuentra una solución que utiliza menos vehículos que la mejor solución encontrada, ambas son reiniciadas.

Las dos colonias trabajan con el depósito duplicado tantas veces como vehículos haya disponibles. La distancia entre copias del depósito es cero. De esta manera una solución es representada por una sola ruta, como en el TSP. Cuando una hormiga llega a alguna copia del depósito, el tiempo y la carga se colocan en cero.

Al ser iniciada, la colonia ACS-TIME recibe como dato de entrada, la menor cantidad de rutas para la que se encontró una solución (v) y trabaja con v copias del depósito intentando minimizar la distancia recorrida. Las hormigas de esta colonia se mueven siguiendo la estrategia ACS en la cual la visibilidad se define como

$$\eta_{ij} = 1/\max\{(b_i + t_{ij}, e_j) - b_i\}\,(l_j - b_i)$$

donde b_i es la hora de salida del nodo i. Al realizar las inserciones sólo se tiene en cuenta los clientes no visitados que pueden ser agregados en la ruta de manera

factible. Debido a esas restricciones de factibilidad y a que la cantidad de copias del depósito es finita, puede ocurrir que ciertos clientes no sean visitados. En dicho caso se aplica la mejor inserción para cada cliente no visitado, en orden decreciente de demanda. Finalmente, se utiliza el procedimiento CROSS [54] para realizar una búsqueda local.

La colonia ACS-VEI también recibe v como dato de entrada, pero trabaja con $v-1$ copias del depósito e intenta maximizar la cantidad de clientes visitados. El objetivo es determinar si existe alguna solución factible que utilice solamente $v-1$ vehículos. La colonia sigue la estrategia ACS donde

$$\eta_{ij} = 1/\max\left\{1, \left(\max\left\{b_i + t_{ij}, e_j\right\} - b_i\right)\left(e_j - b_i\right) - IN(j)\right\}$$

siendo $IN(j)$ es la cantidad de soluciones que no visitaron j desde la última vez que esta colonia mejoró una solución. También se aplica un procedimiento de inserción para los clientes no visitados pero no se realiza una búsqueda local, pues el objetivo es visitar la mayor cantidad de clientes y no obtener una buena solución en cuanto a los costos. Cuando esta colonia encuentra una solución factible que visita a todos los clientes se re-inician ambas colonias: ACS-TIME buscará soluciones con $v-1$ vehículos y ACS-VEI determinará si es posible solucionar el problema con $v-2$ vehículos.

4.2 Tabu Search

La metaheurística de Búsqueda Tabú o *Tabu Search* fue propuesta por Glover [55] y tiene como principio básico realizar una búsqueda local aceptando soluciones que aumentan el costo. En la iteración t el algoritmo se mueve de la solución s_t a la s_{t+1}, que es la mejor dentro de un subconjunto de sus soluciones vecinas $N(s_t)$. Se llama *movida* a la operación que se aplica a s_t para obtener s_{t+1}. Notar que s_{t+1} no necesariamente es de menor costo que s_t y, por lo tanto, debe utilizarse algún mecanismo para que en la iteración siguiente no se vuelva a s_t. Una opción sería almacenar todas las soluciones por las que se va pasando, pero a un costo de almacenamiento excesivo. En lugar de eso se utiliza una *memoria de corto plazo* que registra algunos atributos de las soluciones ya visitadas y se evita, durante cierta cantidad de iteraciones θ, considerar soluciones que posean dichos atributos. Las soluciones prohibidas se denominan *soluciones tabú* y las movidas que llevan hacia soluciones tabú se llaman *movidas tabú*. Suele utilizarse un criterio, llamado *criterio de aspiración*, para aceptar soluciones aún cuando sean tabú, por ejemplo, si mejoran el costo de la mejor solución encontrada hasta el momento. Se llama *soluciones admisibles* a aquellas que no son soluciones tabu y a las que pasan el criterio de aspiración (aún si son tabu). La búsqueda se realiza sobre las soluciones admisibles de la vecindad.

Como mecanismo de *diversificación*, es decir, para asegurar una exploración de diferentes regiones del espacio de soluciones, puede utilizarse una *memoria de largo plazo* que penalice las movidas realizadas con mucha frecuencia. Finalmente, también puede utilizarse algún mecanismo de *intensificación*, como realizar búsqeudas más exhaustivas periódicamente o utilizar vecindades más grandes en algunas de las mejores soluciones encontradas.

4.2.1 Tabu Search para el VRP

El Algoritmo de Osman

En el Algoritmo de Osman [56] las vecinas de una solución se obtienen mediante intercambios de clientes entre pares de rutas. Si r_p y r_q son dos rutas diferentes de una solución, se define un λ-intercambio[1] como el pasaje de a lo sumo λ clientes de r_p a r_q y a lo sumo λ clientes de r_q a r_p. La cantidad de clientes que cada ruta envía a la otra no necesariamente debe ser la misma, pero ambas deben ser no mayores que λ. La vecindad de una solución s, $N_\lambda(s)$, consiste en todas las soluciones que se pueden obtener aplicando esta operación a cualquier par de rutas diferentes de s. En este trabajo se utiliza $\lambda = 1$.

El costo de una movida se evalúa de manera heurística. Al eliminar un cliente v_i de una ruta, ésta se reconstruye uniendo el anterior y el siguiente a v_i. Al insertar un cliente en una ruta, se coloca entre dos clientes consecutivos de forma de incrementar el costo lo menos posible. Luego de realizar una movida, se aplica el algoritmo 2-opt [16] sobre cada una de las rutas implicadas.

Si se realiza una movida en que las rutas r_p y r_q intercambian los clientes v_i y v_j respectivamente, entonces la movida que vuelve v_i a r_p y v_j a r_q es una movida tabú. Del mismo modo, si se realiza la movida en que la ruta r_p envía el cliente v_i a la ruta r_q y ésta última no le envía ningún cliente, la movida inversa, en la que r_q devuelve v_i a r_p y r_p no le envía ningún cliente a r_q es una movida tabú. El tiempo que una movida es considerada tabú es fijo y se determina según las características de cada problema. Se utiliza el criterio de aspiración clásico: si una solución es mejor que la mejor solución hallada hasta el momento, se acepta aunque esto implique realizar una movida tabu.

Para realizar la búsqueda en la vecindad de una solución s se utilizan dos criterios. En el criterio *Best Admissible* (BA) se busca la mejor solución admisible en $N_\lambda(s)$ y en el *First Best Admisible* (FBA) se selecciona la primer solución admisible $N_\lambda(s)$ que mejore el costo de s (si ninguna mejora el costo se selecciona la que menos lo empeora, pero habiendo necesitado examinar todos los intercambios posibles).

El Algoritmo Taburoute

Gendreau, Hertz y Laporte [57] propusieron un algoritmo de búsqueda tabú que acepta soluciones no factibles durante la búsqueda. Se consideran conjuntos de rutas que visitan exactamente una vez a cada cliente, pero se permite violar las restricciones de capacidad y de largo máximo de cada ruta. Si $Q(s)$ mide el exceso total de capacidad de las rutas y $L(s)$ mide el exceso total de largo de las rutas en la solución s, se utiliza como función objetivo a

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

Siendo $c(s)$ el costo original en el problema. Notar que si una solución $\hat{s}$ es factible, entonces $c'(\hat{s}) = c(\hat{s})$. En cambio, para las soluciones no factibles se incorporan dos términos que penalizan las violaciones a las restricciones, presionando las soluciones a pertenecer a la región factible. Cada 10 iteraciones, se

[1] en inglés se utiliza λ-interchange para esta operación, que se aplica sobre dos rutas, y λ-exchange para la operación definida por Lin [16], que se aplica sobre una sola ruta. Sin embargo, tanto "exchange" como "interchange" se traducen como "intercambio" en español. En este documento nos referimos a éstos como los λ-intercambios de Osman.

ajusta los parámetros α y β. Si todas las soluciones de las últimas 10 iteraciones fueron factibles respecto a la capacidad, α se divide entre 2 y si todas fueron no factibles, se multiplica por 2. Lo mismo se realiza para β pero considerando la restricción de largo máximo de la ruta.

Dada una solución se evalúa, para cada cliente v, eliminarlo de su ruta e insertarlo en otra ruta que contenga alguno de sus p vecinos más cercanos (p es un parámetro del algoritmo). Se utiliza las inserciones GENI (ver Sección 2.6.5). Si el cliente v se elimina de la ruta r, entonces volver a insertarlo en r es una movida tabu durante θ iteraciones donde θ se sortea uniformemente en $[5, 10]$.

Se utilizan dos mecanismos de intensificación. Por un lado, si la mejor solución encontrada no es mejorada durante cierta cantidad de iteraciones, se amplía el tamaño de la vecindad incrementando el valor de p. Además, periódicamente se aplica el operador US (ver Sección 2.6.5) para realizar una búsqueda local sobre la solución obtenida luego de la movida. Como método de diversificación se penaliza la concentración de movidas sobre los mismos clientes. Al considerar el costo de una movida en la que participa el cliente v, se suma un término proporcional a la cantidad de veces que dicho cliente fue movido durante la ejecución del algoritmo.

El procedimiento de búsqueda se ejecuta varias veces partiendo de diferentes soluciones y durante pocas iteraciones; a la mejor solución obtenida en estos *falsos comienzos* se le aplica el algoritmo durante más iteraciones.

El Algoritmo de Taillard

En el algoritmo de Taillard [58], se realiza una partición del conjunto de clientes y cada partición se resuelve como un VRP independiente de los demás mediante Tabu Search. Para problemas uniformes (en los que el depósito está apróximadamente en el centro y los clientes están regularmente distribuidos) y con coordenadas euclídeas se propone una partición basada en consideraciones geométricas. Para el resto de los problemas, se da un método de partición que utiliza árboles de cubrimiento formados por los caminos mínimos del depósito a cada cliente. Cada algunas iteraciones se modifica la partición utilizando información sobre la mejor solución obtenida.

Se utiliza la misma definición de vecindad que en el algoritmo de Osman [56] con $\lambda = 1$. El costo de las movidas se evalúa en forma heurística: al insertar un cliente en una ruta se hace entre dos clientes consecutivos de modo de minimizar el aumento en el costo; y al eliminar un cliente de una ruta, se unen el anterior y el siguiente a él. La inversa de una movida es considerada tabu por una cantidad θ de iteraciones, donde θ se sortea uniformemente en $[0.4n, 0.6n]$. La movida de un mismo cliente repetidas veces es penalizada, a menos que cumpla con el criterio de aspiración, que es el usual de mejorar la mejor solución encontrada hasta el momento.

Periódicamente, cada ruta es optimizada resolviendo un TSP sobre sus clientes de forma exacta mediante el algoritmo de Volgenant y Jonker [59]. Como en los casos estudiados la cantidad de clientes por ruta es baja (siempre menor que 40), esto no representa un problema siempre que no se realice con demasiada frecuencia.

Memorias Adaptativas de Rochat y Taillard

Si bien la propuesta de Rochat y Taillard [60] puede ser utilizada junto con cualquier algoritmo de búsqueda local, tiene su origen en el marco de Tabu Search. La idea sobre la que se basa es que si cierta característica aparece consistentemente en las buenas soluciones, entonces muy probablemente las soluciones que posean dicha caracerística serán mejores que las que no la posean. En los problemas de ruteo de vehículos, es razonable considerar a las rutas como dicha característica: si cierta ruta aparecen repetidamente en las buenas soluciones, probablemente sea bueno considerarla.

El algoritmo comienza generando un conjunto de soluciones diferentes mediante varias ejecuciones de alguna heurística no determinística de búsqueda local. Luego de esta etapa, se espera tener (aunque de manera "oculta") toda la información necesaria para armar buenas soluciones. Dicho de otro modo, es de esperar que varias rutas muy similares a las necesarias para construir una buena solución hayan sido generadas, aunque quizás estén en soluciones diferentes. Al final de esta fase se tiene un conjunto R formado por las rutas de todas las soluciones obtenidas; si una ruta aparece en más de una solución, se tiene múltiples copias de dicha ruta en R. Cada ruta se etiqueta con el valor de la solución a la que pertenece.

En una segunda fase se busca extraer rutas de R para armar una nueva solución. Para esto, se crea R' ordenando R de manera creciente en las etiquetas y considerando solamente las primeras L rutas (considerando las rutas repetidas como rutas diferentes). Luego se selecciona una ruta de R', donde la probabilidad de elegir la ruta en la posición i es $\frac{2i}{|R'||R'+1|}$. Dicha ruta se agrega a la solución y se eliminan de R' todas las que contengan clientes en común con ella. El procedimiento se repite hasta que R' se vacíe. Al finalizar podría quedar algunos clientes sin visitar, para los que se resolverá un VRP con la heurística de búsqueda local, agregando las rutas obtenidas a la solución. Sobre esta solución se realiza una nueva búsqueda local y las rutas obtenidas se agregan al conjunto original R. Esta segunda fase de construcción y búsqueda local se puede repetir tantas veces como se quiera.

En el trabajo original se utiliza el Algoritmo de Taillard [58] como método de búsqueda local. Al finalizar el algoritmo, como método de post-optimización, se resuelve un Set Partitioning Problem con las rutas de R.

El Algoritmo de Xu y Kelly

En el algoritmo propuesto por Xu y Kelly [61] se utiliza una red de flujo para buscar intercambios de clientes entre las rutas. Dada una solución con m rutas, se construye una red con 4 niveles de arcos, como la que se muestra en la Figura 4.1. En el primer nivel, se conecta el nodo fuente con un nodo por cada ruta. El flujo en estos arcos indica la cantidad de clientes eliminados de cada ruta. La oferta del nodo s está acotada superiormente por U. En el segundo nivel hay un arco entre cada ruta y los clientes que ésta visita, cuya capacidad máxima es 1 y el flujo asignado indica si el cliente es eliminado de la ruta o no. Los arcos del tercer nivel unen a cada cliente con cada ruta. Su capacidad máxima es también 1 y el flujo indica si el cliente es insertado en la ruta o no. Finalmente, se conecta a cada ruta con un nodo terminal y el flujo a través de esos arcos indica la cantidad de clientes insertados en cada ruta.

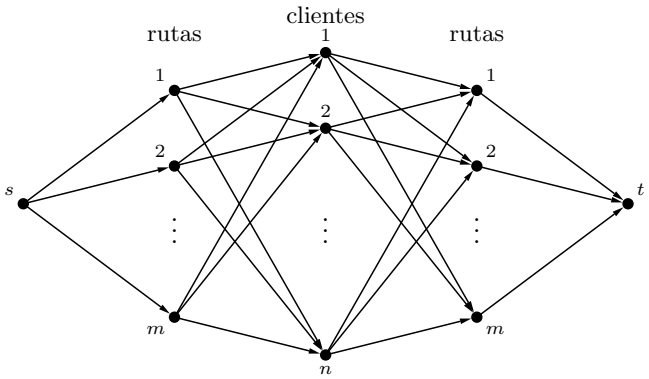


Figura 4.1: Red de flujo utilizada en el Algoritmo de Xu y Kelly.

Un flujo sobre esta red da una manera de reasignar a lo sumo U clientes entre las rutas, sin tener en cuenta la capacidad de los vehículos. Los vehículos sobrecargados se penalizan al definir el costo de las movidas como en Taburoute. A cada arco de los niveles 2 y 3 se asocia un costo compuesto de dos términos. Por un lado, se mide de manera aproximada el aumento en la función objetivo del problema si se realizara la eliminación o inserción asociada al arco. Además, se agrega un término para favorecer la inserción de clientes en las rutas que están muy por debajo de la capacidad del vehículo y para penalizar las inserciones en rutas muy sobrecargadas.

La vecindad utilizada en el algoritmo de Tabu Search alterna entre las movidas definidas por el flujo de costo mínimo en la red y los λ-intercambios de Osman. Se penaliza la realización repetida de movidas sobre el mismo cliente sumando al costo de las movidas, un término que mide cuántas veces se realizaron dichas movidas. Cuando se realiza una movida, la movida inversa es declarada tabu por una cantidad fija de iteraciones. Periódicamente se aplica el algoritmo 3-opt o el 2-opt sobre la ruta.

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.

Ejection Chains de Rego y Roucariol

Las Cadenas de Expulsiones (o *ejection chains*) fueron propuestas por Glover [62] como una técnica general para definir vecindades en un espacio de soluciones. En su versión más general, una movida consiste en seleccionar algunos elementos que cambiarán su estado o valor y, como resultado, otros elementos serán expulsados de su estado actual. En el caso de los Problemas de Ruteo de Vehículos se trata de cambiar las posiciones de algunos nodos en las rutas [63].

Dado un nodo v_i, denotamos como v_{i-1} al nodo anterior y v_{i+1} al posterior en la ruta de v_i. Una cadena de expulsiones de ℓ niveles consiste en triplas $\left(v_{i-1}^k, v_i^k, v_{i+1}^k\right)$ para $k = 0, \ldots, \ell$, en las que cada v_i^{k-1} reemplazará a v_i^k luego de la movida. Los nodos v_i^k se denominan *nodos centrales*. Una movida se interpreta como que el nodo central de un nivel expulsa al nodo central del nivel siguiente de su ubicación; este último debe ubicarse en otro lugar y expulsará al

siguiente nodo central, ocurriendo una cadena de expulsiones. El último nodo en ser expulsado, v_i^ℓ, debe ubicarse de modo tal que se obtenga rutas cerradas y se proponen dos alternativas. En las movidas de Tipo I se lo ubica entre v_{i-1}^0 y v_{i+1}^0. En las movidas de Tipo II, se crea el arco $\left(v_{i-1}^0, v_{i+1}^0\right)$ y v_i^ℓ se coloca entre dos nodos cualesquiera v_p y v_q que no hayan sido nodos centrales de ninguna tripla. Las Figuras 4.2 y Figuras 4.3 muestran un ejemplo de movidas de cada tipo.

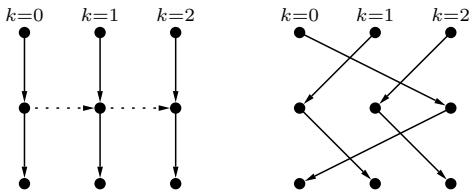


Figura 4.2: Una Cadena de Expulsiones de Tipo I para 3 niveles.

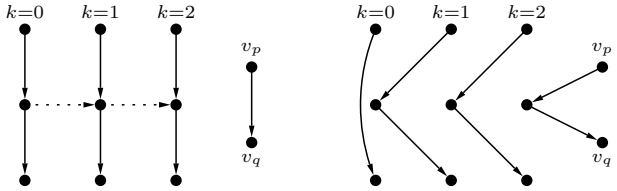


Figura 4.3: Una Cadena de Expulsiones de Tipo II para 3 niveles.

Debe notarse que no es necesario que todas las triplas estén en diferentes rutas. De todos modos, no cualquier conjunto de triplas es válido. En el artículo original se da un conjunto de condiciones sobre la cantidad de veces que cada nodo puede aparecer en las triplas, llamadas *restricciones de legitimidad*, así como un procedimiento para buscar la mejor cadena de expulsiones para cierta solución. Dicho procedimiento se utiliza embebido en un algoritmo de Tabu Search paralelo. Se declara como tabu a las cadenas de expulsiones que tengan en común las inversas de la primera y última expulsión de la movida realizada, por una cantidad de iteraciones sorteada uniformemente en un intervalo. Se utiliza el criterio de aspiración usual.

El paralelismo se utiliza para, partiendo de una misma solución, realizar ejecuciones independientes del algoritmo. Cada cierto tiempo, los procesos vuelven a comenzar su búsqueda partiendo de la mejor solución encontrada (considerando todos los procesos).

El Algoritmo "Flor"

En este algoritmo se utiliza una estructura llamada Flor para generar una vecindad [64]. Una flor es un conjunto de rutas y un camino que parte del depósito,

tal que todos los clientes del problema son visitados exactamente una vez. Se utilizan Cadenas de Expulsiones [62] para transformar una Flor en otra mediante ciertas transformaciones y, además, se dan reglas para crear una Flor a partir de una solución y viceversa.

Granular Tabu Search

La idea principal del algoritmo Granular Tabu Search de Toth y Vigo [65] consiste en disminuir la cantidad de arcos considerados, eliminando los que conectan nodos muy lejanos. Estos arcos tienen muy pocas chances de pertenecer a una buena solución. 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 E es el conjunto de arcos originales e I es un conjunto de arcos "importantes", como por ejemplo, los incidentes al depósito. En el mencionado trabajo se utiliza $\nu = \beta\bar{c}$ siendo $\bar{c}$ es el costo medio de un arco en una solución obtenida por alguna heurística y $\beta \in [1,2]$ se actualiza periódicamente durante la ejecución del algoritmo. 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.

DETABA

En el Algoritmo DETABA, propuesto por Barbarosoğlu y Ôzgür [66], se utilizan dos tipos de movidas. Las movidas TANE seleccionan dos rutas r_1 y r_2 al azar y de cada una eligen un conjunto aleatorio de clientes s_1 y s_2. El conjunto s_1 se elimina de r_1 y se inserta en r_2 y la operación análoga se realiza con s_2. Los conjuntos s_1 y s_2 deben elegirse de modo que al realizar la operación anterior no se viole las restricciones del problema. Luego de eso se aplica 2-opt a r_1 y r_2. En las movidas TANEC el procedimiento es similar al anterior, pero se procura que los clientes de s_1 y s_2 sean lejanos al centroide de sus rutas respectivas y cercanos al centroide de la ruta a la que no pertenecen. Para seleccionar la movida a realizar, se genera una cantidad de movidas TANE y TANEC y se implementa la mejor que no sea tabu. Si en una movida el cliente v_1 pasa de r_1 a r_2 y v_2 pasa de r_2 a r_1, entonces se declara tabu el volver a colocar v_1 en r_1 y v_2 en r_2. La cantidad de iteraciones que un movida es tabu se sortea uniformemente en un intervalo.

Se utilizan falsos comienzos como en Taburoute [57], es decir, se ejecuta la búsqueda durante unas pocas iteraciones para generar un conjunto de soluciones iniciales. Luego se vuelve a ejecutar la búsqueda durante más iteraciones, partiendo de la mejor de esas soluciones iniciales.

4.2.2 Tabu Search para el VRPTW

Unified Tabu Search

Cordeau, Laporte y Mercier, propusieron un método de Búsqueda Tabú para el VRPTW al que llamaron Unified Tabu Search [67]. Se supone que el tamaño de la flota es m, conocido de antemano. Se permite considerar soluciones no factibles y las violaciones a la capacidad, largo máximo de las rutas y ventanas de

tiempo, se penaliza como en Taburoute. La actualización de las ponderaciones en los términos de dicha penalización se actualizan en cada iteración.

Las movidas utilizadas simplemente seleccionan un cliente, lo eliminan de su ruta y lo insertan en otra entre dos nodos consecutivos. Al eliminar un cliente de su ruta, ésta se reconecta uniendo el anterior y el sucesor del cliente eliminado. Si el cliente i se elimina de la ruta r, volver a insertarlo i en r es una movida tabú por θ iteraciones.

Como método de diversificación, si colocar el cliente i en la ruta r empeora la función objetivo, se penaliza dicha movida con un término ρ_{ir} que mide cuántas veces el cliente i fue insertado en r durante toda la búsqueda. Al finalizar la búsqueda, se aplica una versión de GENI para problemas con ventanas de tiempo [68] sobre la mejor solución encontrada, como método de post-optimización.

4.3 Algoritmos Genéticos

Los Algoritmos Genéticos (AG), introducidos por Holland [69], utilizan las 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). El algoritmo opera sobre una población P de soluciones codificadas, llamadas individuos. Para cada individuo $i \in P$ se define una *función de fitness* $f(i)$ de modo que cuánto mayor es el fitness de un individuo, mejor es la solución que éste representa[2]. En cada iteración se aplican operadores evolutivos que combinan y modifican a los individuos de la población, creando una nueva. En el esquema usual 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. De modo general, puede decirse que este operador genera una población intermedia (o *mating pool*) cuya cantidad de individuos depende de las características del operador de cruzamiento utilizado. Los operadores de selección son, en general, probabilísticos y suelen privilegiar a los individuos con mayor fitness en la población. En el operador de Selección Proporcional, la probabilidad de seleccionar al individuo i es proporcional a $\frac{f(i)}{\bar{f}}$, siendo $\bar{f} = \frac{1}{|P|}\sum_{i \in P} f(i)$ el fitness promedio de la población. En el operador de Selección por Torneo se toma un conjunto de individuos al azar y se selecciona el de mayor fitness, repitiendo el proceso tantas veces como sea necesario para generar la población intermedia.

Una vez que se generó la población intermedia, se aplica repetidas veces el operador de *cruzamiento* para combinar individuos de dicha población y generar una nueva. Usualmente los operadores de cruzamiento toman dos individuos p_1 y p_2 llamados padres, y generan dos individuos h_1 y h_2 llamados hijos, mediante la aplicación de una regla probabilística. Si utiliza una codificación basada en vectores, los operadores de cruzamiento clásicos son los de n puntos, que consisten en "cortar" cada padre en n posiciones, de manera aleatoria, e intercambiar los segmentos intermedios. Por ejemplo, si $p_1 = 1001001101$ y $p_2 = 1100101110$, un posible resultado de un cruzamiento de dos puntos (que se

[2]En algunos problemas de minimización, se define un fitness proporcional a la función objetivo, de modo que en estos casos se desearía minimizar el fitness. El objetivo quedará claro en cada contexto.

da cuando los padres se cortan luego de las posiciones 3 y 6) es $h_1 = 1000101101$ y $h_2 = 1101001110$. El operador de cruzamiento de un punto se denomina SPX (Single Point Crossover) y el de dos puntos DPX (Double Point Crossover). Otro operador clásico es el UX (Uniform Crossover): para cada posición, con probabilidad 0.5 el hijo h_1 toma el valor de dicha posición de p_1 y h_2 de p_2 y con probabilidad 0.5 ocurre lo contrario.

Finalmente, se aplica a algunos individuos de la nueva población, un operador probabilístico de *mutación* que consiste en realizarle alguna modificación. En codificaciones con vectores binarios suele invertirse algunos bits o permutar los valores de dos posiciones.

Los operadores de cruzamiento y mutación operan sobre los individuos, es decir, sobre la codificación de las soluciones. A cada una de las poblaciones sucesivas se le llama generación. El modelo presentado no es el único modelo de evolución. Por ejemplo, en los Algoritmos Genéticos de Estado Estacionario, los individuos se generan de a uno y reemplazan (si corresponde) por el peor individuo de la población.

4.3.1 Algoritmos Genéticos para el TSP

El esquema de codificación natural para el TSP consiste en representar a una solución como una secuencia de nodos, es decir, una permutación de $(1, \ldots, n)$. Los operadores de cruzamiento y mutación presentados en la sección anterior pueden generar n-úplas que no sean permutaciones. Por lo tanto, debe definirse operadores de cruzamiento y mutación para que mantengan la codificación.

El operador Partially Matched Crossover (PMX), consiste en aplicar el operador DPX y luego utilizar la información el segmento intermedio para reemplazar los valores repetidos. Por ejemplo, si $p_1 = 42|136|57$ y $p_2 = 65|273|14$ (donde | indica un punto de corte), al aplicar DPX se obtendría $42|273|57$ y $65|136|14$, que no son permutaciones. Los elementos de los segmentos intercambiados (es decir, 136 y 273) que están en iguales posiciones, definen los reemplazos a realizar para los elementos repetidos en cada hijo. En el ejemplo, los reemplazos son $2 \leftrightarrow 1$, $7 \leftrightarrow 3$ y $3 \leftrightarrow 6$, lo que resulta en $h_1 = 4126357$ y $h_2 = 7513624$.

El uso de Algoritmos Genéticos para resolver el TSP es extenso. Por una reseña sobre la aplicación de esta técnica al TSP se puede consultar el trabajo de Potvin [70].

4.3.2 Algoritmos Genéticos para el VRP

GVR

En Genetic Vehicle Representation (GVR)[71] se trabaja directamente sobre las soluciones. Para cruzar dos soluciones p_1 y p_2, se toma una sub-ruta $r = (v_1, \ldots, v_k)$ de p_1 (al tratarse de una sub-ruta, no necesariamente se cumple que $v_1 = 0$ y $v_k = 0$) y se determina el cliente w_j más cercano a v_1 que no está en r. Si la ruta a la que pertenece w en la solución p_2 es $r' = (0, w_1, \ldots, w_j, w_{j+1}, \ldots, 0)$ entonces ésta se reemplaza por $(0, w_1, \ldots, w_j, v_1, \ldots, v_k, w_{j+1}, \ldots, 0)$. Es decir, se inserta r en r' a continuación de w. Si esta ruta no fuera factible, se particiona en tantas ruta factibles como sea necesario. Con esto se genera un hijo, el otro hijo es una copia de p_1.

Se utilizan cuatro operadores de mutación: intercambiar la posición de dos clientes en una ruta, invertir el orden de una ruta, re-insertar un cliente en una ruta diferente a la que pertenece y seleccionar una sub-ruta e insertarla en otro lugar de la solución. Los últimos dos operadores pueden generar rutas nuevas o eliminar rutas, mientras que los dos primeros mantienen la cantidad de rutas de la solución.

El Algoritmo de Baker y Ayechew

El algoritmo propuesto por Baker y Ayechew [72] se basa en la técnica de Asignar Primero - Rutear Después (ver Sección 2.3). Cada individuo codifica la asignación de clientes a vehículos mediante un vector que en la posición i indica el número del vehículo asignado. El índice asignado a cada cliente es tal que si la distancia entre dos clientes es pequeña, entonces sus índices serán cercanos. Esto se logra utilizando una modificación de la Heurística de Barrido (ver Sección 2.3.1) para asignar índices a los clientes.

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. Se permite violar las restricciones del problema, penalizando dichas violaciones en la función de fitness. Se utiliza un Algoritmo de Estado Estacionario, Selección por Torneo, DPX como operador de cruzamiento y un intercambio probabilístico entre dos posiciones del vector como operador de mutación.

4.3.3 Algoritmos Genéticos para el VRPTW

GIDEON

El Algoritmo GIDEON de Thangiah [73] utiliza la técnica de Asignar Primero - Rutear Después (ver Sección 2.3). Los clusters se generan mediante la ubicación de K "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 AG se utiliza para determinar la mejor ubicación de los puntos semilla. La ubicación de un punto semilla se codifica, utilizando 5 bits. Un vector de $5K$ bits representa la ubicación de los K puntos semilla.

Para calcular el fitness de un individuo se genera una ruta para cada uno de sus clusters utilizando un algoritmo simple de inserción para el TSP. Se permite que las rutas sean no factibles. El fitness de un individuo es el costo de las rutas obtenidas para cada cluster, más términos que penalizan las violaciones a las restricciones de capacidad, largo máximo de cada ruta y ventanas de tiempo. El operador de cruzamiento utilizado es el DPX y como operador de mutación se modifican algunos bits aleatoriamente.

Al final de la ejecución del algoritmo, se realiza una búsqueda local sobre la mejor solución encontrada, utilizando los λ-intercambios definidos por Osman (ver Sección 4.2.1). Esta nueva solución es utilizada para modificar las coordenadas polares de cada cliente, de modo que los clientes consecutivos en una ruta queden consecutivos si se ordenaran por su ángulo en las nuevas coordenadas polares. Con esas nuevas coordenadas se vuelve a ejecutar el Algoritmo Genético y se sigue el proceso por algunas iteraciones.

GenSAT

En el Algoritmo GenSAT [74] simplemente se aplica un post-procesamiento a la solución obtenida mediante la ejecución de GIDEON [73]. Dicho post-procesamiento incluye la ejecución de métodos de descenso simples (First Best y Global Best) y además Simulated Annealing y Tabu Search utilizando vecindades definidas mediante 2-intercambios.

El Algoritmo de Blanton y Wainwright

En el algoritmo propuesto por Blanton y Wainwright [75] se asume que se tiene una flota fija de K vehículos. Cada individuo es una permutación del conjunto de los clientes. Para obtener el conjunto de rutas que la permutación representa, se inicializan K rutas utilizando los primeros K clientes de la permutación y el resto de los clientes son insertados secuencialmente en la 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.

Se proponen dos operadores de cruzamiento llamados MX1 y MX2, que generan un solo hijo. Ambos consideran un ordenamiento global de los clientes dado por su ventana de tiempo, que expresa un orden deseable para las visitas. Si los clientes en la posición 1 de cada padre son v_1 y v'_1, asigna a la posición 1 del hijo el cliente (v_1 o v'_1) que esté antes en el orden global. Luego se intercambian las posiciones de modo que en la posición 1 de ambos padres quede en cliente elegido y se procede de la misma manera con todas las posiciones hasta procesar todos los clientes. El operador MX2 es similar, pero en lugar de intercambiar las posiciones, se elimina de ambos padres el cliente elegido.

GENEROUS

Potvin y Bengio propusieron un Algoritmo Genético llamado GENEtic ROUting System (GENEROUS) [76] en el cual los operadores evolutivos se aplican directamente sobre las soluciones factibles y no sobre una codificación de estas. Para calcular el fitness de un individuo se utiliza un esquema de Ranking Lineal [77]. Se fija un fitness máximo $f_{\max}$ y un fitness mínimo $f_{\min}$ y se ordenan las soluciones colocando la mejor al comienzo (posición 1) y la peor al final (posición $|P|$). El fitness de la solución en la posición i es

$$f_{\max} - \frac{(f_{\max} - f_{\min})\,(i-1)}{|P|-1},$$

es decir, a la mejor solución se le asigna $f_{\max}$, a la peor $f_{\min}$ y al resto se le asigna valores espaciados uniformemente en ese intervalo. Se utiliza Selección Proporcional.

Se proponen dos operadores para combinar soluciones, el Sequence-Based Crossover (SBX) y en Route-Based Crossover (RBX). En el SBX, se seleccionan dos rutas r_1 y r_2, una ruta de cada padre, y se elimina un arco de cada una. Luego se arma una nueva ruta uniendo el segmento de r_1 anterior al corte con el de r_2 posterior al corte y la ruta obtenida reemplaza a r_1. La solución obtenida puede contener clientes duplicados y una de las copias debe ser eliminada. Puede ocurrir, además, que algunos clientes no sean visitados; a éstos se los inserta secuencialmente en la ruta que incremente menos el costo total de

la solución. Si ocurriera algún cliente no puede ser insertado en ninguna de las rutas manteniendo la factibilidad, la solución es descartada (pues reconstruirla implicaría utilizar un vehículo más que en los padres) y el proceso de selección y cruzamiento debe repetirse hasta tener éxito. Invirtiendo el orden en que se consideran los padres (es decir, reemplazando a r_2 con la nueva ruta) puede obtenerse otra solución tentativa. El operador RBX consiste en copiar una ruta de un padre en el otro. Nuevamente, puede darse que algunos clientes estén duplicados y otros queden sin ser visitados, en cuyo caso se aplican las mismas consideraciones que para el operador SBX.

Como operadores de mutación se proponen el One-Level Exchange (1M), Two-Level Exchange (2M) y Local Search (LSM), cuyo principal objetivo es decrementar la cantidad de vehículos de la solución. En el operador 1M se selecciona una ruta r y se inserta cada uno de sus clientes v_i en la ruta r_i que minimice el incremento en el costo total. La selección de la ruta es sesgada hacia las rutas con menos clientes. Puede ser difícil agregar v_i a r_i respetando las restricciones de capacidad y las ventanas de tiempo, por lo que en el operador 2M se considera la posibilidad de reubicar algún cliente de esa ruta, para "hacer lugar" a v_i. Si $r_i = (0, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, 0)$ y la ruta $(0, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, 0)$ es factible y además v_j puede ser insertado en otra ruta r' (que no sea r, pero pudiendo ser r_i), entonces el operador 2M permite insertar v_i en r_i y v_j en r'. Finalmente, el operador LSM realiza una búsqueda local mediante el Algoritmo or-opt (ver Sección 2.6.3).

Al Algoritmo de Berger, Barkaoui y Bräysy

En esta propuesta [78] se utilizan dos poblaciones que evolucionan en paralelo y operan directamente sobre las soluciones. En la población P_1 el objetivo es minimizar el costo total y siempre se tiene al menos una solución factible. La población P_2 intenta minimizar la violación de las restricciones. Los individuos de una misma población tienen la misma cantidad de rutas; los de P_1 tienen $R_{\min}$ y los de P_2 tienen $R_{\min} - 1$, siendo $R_{\min}$ la mínima cantidad de rutas para la que se ha conseguido una solución factible. Cuando se encuentra una nueva mejor solución en P_2, se actualiza $R_{\min}$ y ambas poblaciones.

Ambas poblaciones evolucionan de la misma manera, utilizando los mismos operadores y la misma función de fitness. La única diferencia radica en la cantidad de rutas impuesta a los individuos. Se utiliza un modelo evolutivo de Estado Estacionario: una generación consiste en agregar n_p individuos a la población y luego eliminar los n_p peores individuos. Luego de que ambas poblaciones evolucionan una generación, si P_2 contiene una solución factible, se copian los individuos de P_2 en P_1 y se aplica un operador de mutación llamado RSS_M a los individuos de P_2 para reducir la cantidad de rutas en uno. Los operadores de cruzamiento y mutación son muy complejos; por una descripción de los mismos referirse al artículo original [78].

El Algoritmo de Zhu

En el algoritmo propuesto por Zhu [79], 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. Se

utiliza Selección por Torneo.

Se combinan tres operadores de cruzamiento, el PMX y dos nuevos operadores: Heuristic Crossover (HC) y Merge Crossover (MC). En el HC se selecciona un punto de corte y se elige el cliente siguiente a dicho punto en alguno de los padres (sea v dicho cliente) para comenzar la ruta. En el otro padre, se intercambian dos clientes de modo que v quede en la misma posición en ambos. Si v_1 es el siguiente a v en uno de los padres y v_2 es el siguiente en el otro, entonces se agrega a la ruta el más cercano a v. El proceso se repite partiendo del cliente agregado, hasta completar una ruta. El MC opera de manera similar, pero selecciona el próximo cliente de acuerdo al tiempo y no a la distancia.

Para la mutación se intercambian las posiciones de algunos clientes dentro de una misma ruta.

El Algoritmo de Tan, Lee y Ou

El algoritmo propuesto por Tan, Lee y Ou [80] puede ser considerado un método Asignar Primero - Rutear Después. Cada individuo codifica una agrupación de clientes mediante una permutación y un vector que indica cuántos clientes tiene cada cluster. Por ejemplo, el individuo $(2, 4, 7, 1, 6, 5, 9, 3, 8)(3, 2, 4)$ representa los clusters $(2, 4, 7)$, $(1, 6)$ y $(5, 9, 3, 8)$.

Para evaluar el fitness de un individuo se construye una ruta para cada cluster mediante una Heurística de Inserción Secuencial de Solomon (ver Sección 3.2) y realizando luego una búsqueda local mediante los λ-intercambios de Osman (ver Sección 4.2.1). Se utiliza Selección por Torneo, PMX y un operador de mutación que intercambia dos elementos del primer vector.

El Algoritmo de Bräysy y Dullaert

En la propuesta de Bräysy y Dullaert [54] se utiliza un procedimiento para construir diversas soluciones iniciales, de las que luego se selecciona la mejor como punto de partida para un Algoritmo Genético. El procedimiento de construcción consiste en generar una solución mediante una heurística de Inserción Secuencial similar a las de Solomon (ver Sección 3.2) y luego intentar reducir la cantidad de rutas buscando movidas basadas en Cadenas de Expulsiones (ver Sección 4.2.1).

El Algoritmo Genético propuesto no utiliza una población de soluciones sino una solución. Partiendo de dicha solución, se consideran todos los posibles 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 *pool* y luego de considerar todos los pares se construye una solución seleccionando rutas de dicho pool.

Los operadores de cruzamiento utilizados son I-CROSS y RRC. El operador I-CROSS consiste en intercambiar arcos entre las rutas permitiendo además invertir el orden de las visitas. En el RRC (Reinsert Related Customers) se buscan clientes que estén en rutas diferentes pero sean cercanos entre si para insertarlos nuevamente mediante una Heurística de Inserción en Paralelo. El operador I-CROSS también es utilizado como operador de mutación.

Tunning de Parámetros de Potvin y Dubé

Potvin y Dubé [81] utilizaron un Algoritmo Genético para ajustar los parámetros de la Heurística de Inserción en Paralelo para el VRPTW de Potvin y Rousseau (ver Sección 3.3). Se utiliza una codificación mediante un vector binario para representar un juego de parámetros de la heurística. El fitness de un individuo se define como el costo de la solución obtenida ejecutando la heurística con el juego de parámetros que el individuo representa. Se utiliza Selección por Ranking, SPX como operador de cruzamiento y el operador de mutación modifica probabilísticamente cada bit.

Bibliografía

[1] Toth, P., Vigo, D.: An Overview of Vehicle Routing Problems. Monographs on Discrete Mathematics and Applications. In: The Vehicle Routing Problem. SIAM (2000) 1–26

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

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

[4] Dantzig, G., Fulkerson, D., Johnson, S.: Solution of a large scale traveling salesman problem. Operations Research **2** (1954) 393–410

[5] C. Miller, A. Tucker, R. Zemlin: Integer programming formulation of traveling salesman problems. Journal of the ACM **7** (1960) 326–329

[6] Garey, M., Johnson, D.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Company (1979)

[7] Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley and Sons (1990)

[8] Golden, B., Assad, A., Levy, L., Gheysens, F.: The fleet size and mix vehicle routing problem. Computers & Operations Research **11** (1984) 49 – 66

[9] Cordeau, F., Desaulniers, G., Desrosiers, J., Solomon, M., Soumis, F.: The VRP with time windows. Technical Report Cahiers du GERAD G-99-13, École des Hautes Études Commerciales de Montréal (1999)

[10] Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley & Sons (1988)

[11] Laporte, G., Nobert, Y.: Exact algorithms for the vehicle routing problem. Annals of Discrete Mathematics **31** (1987) 147–184

[12] Laporte, G.: The vehicle routing problem: an overview of exact and approximate algorithms. European Journal of Operational Research **59** (1992) 345–358

[13] Yellow, P.: A computational modification to the savings method of vehicle scheduling. Operational Research Quarterly **21** (1970) 281–283

[14] Golden, B., Magnanti, T., Nguyen, H.: Implementing vehicle routing algorithms. Networks **7** (1977) 113–148

[15] Gaskell, T.: Bases for vehicle fleet scheduling. Operational Research Quarterly **18** (1967) 281–295

[16] Lin, S.: Computer solutions of the traveling salesman problem. Bell System Technical Journal **44** (1965) 2245–2269

[17] Gabow, H.: An efficient implementation of Edmonds' algorithm for maximum matching on graphs. Journal of the ACM **23** (1976) 221–234

[18] Desrochers, M., Verhoog, T.: A matching based savings algorithm for the vehicle routing problem. Technical Report Cahiers du GERAD G-89-04, École des Hautes Études Commerciales de Montréal (1989)

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

[20] Wark, P., Holt, J.: A repeated matching heuristic for the vehicle routing problem. Journal of Operational Research Society **45** (1994) 1156–1167

[21] Mole, R.H., Jameson, S.R.: A sequential route-building algorithm employing a generalised savings criterion. Operational Research Quarterly **27** (1976) 503–511

[22] Potvin, J.Y., Rousseau, J.M.: A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research **66** (1993) 331–340

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

[24] Christofides, N., Mingozzi, A., Toth, P.: The Vehicle Routing Problem. In: Combinatorial Optimization. Wiley, Chichester (1979) 315–338

[25] Wren, A.: Computers in transport planning and operation. Ian Allan (1971)

[26] Wren, A., Holliday, A.: Computer scheduling of vehicles form one or more depots to a number of delivery points. Operational Research Quarterly **23** (1972) 333–344

[27] Gillett, B., Miller, L.: A heuristic algorithm for the vehicle-dispatch problem. Operations Research **22** (1974) 340–349

[28] Fisher, M., Jaikumar, R.: A generalized assignment heuristic for the vehicle routing problem. Networks **11** (1981) 109–124

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

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

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

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

[33] Foster, B., Ryan, D.: An integer programming approach to the vehicle scheduling problem. Journal of the Operational Research Society **27** (1976) 367–384

[34] Ryan, D., Hjorring, C., Glover, F.: Extensions of the petal method for vehicle routing. Journal of Operational Research Society **44** (1993) 289–296

[35] Renaud, J., Boctor, F., Laporte, G.: An improved petal heuristic for the vehicle ruting problem. Journal of Operational Research Society **47** (1996) 329–336

[36] Johnson, D., McGeoch., L.: The Traveling Salesman Problem: a case study in local optimization. In: Local Search in Combinatorial Optimization. John Wiley and Sons (1997) 215–310

[37] Renaud, J., Boctor, F., Laporte, G.: A fast composite heuristic for the symmetric travelling salesman problem. INFORMS Journal on Computing **8** (1996) 134–143

[38] Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. Operations Research (1973) 498–516

[39] Or, I.: Traveling salesman-type combinatorial optimization problems and theeir relation to the logistics of regional blood banking (1976)

[40] Breedam, A.V.: Improvement heuristics for the vehicle routing problem based on simulated annealing. European Journal of Operational Research **86** (1995) 480–490

[41] Gendreau, M., Hertz, A., Laporte, G.: New insertion and post optimization procedures for the traveling salesman problem. Operations Research **40** (1992) 1086–1094

[42] Thompson, P., Psaraftis, H.: Cyclic transfer algorithms for multivehicle routing and scheduling problems. Operations Research **41** (1993) 935–946

[43] Solomon, M.: Algorithms for the vehicle routing and scheduling problem with time window constraints. Operations Research **35** (1987) 254–264

[44] Dullaert, W., Janssens, G., Sorensen, K., Vernimmen, B.: New heuristics for the fleet size and mix vehicle routing problem with time windows. Journal of the Operational Research Society **53** (2002) 1232–1238

[45] Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In Varela, F., Bourgine, P., eds.: Proceedings of the European Conference on Artificial Life, Elsevier, Amsterdam (1991) 134–142

[46] Dorigo, M., Maniezzo, V., Colorni, A.: The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics **26** (1996) 29–41

[47] Bullnheimer, B., Hartl, R., Strauss, C.: A new rank based version of the ant system – a computational study. Technical report, University of Viena, Institute of Management Science (1997)

[48] Gambardella, L.M., Dorigo, M.: Ant-Q: A reinforcement learning approach to the traveling salesman problem. In: International Conference on Machine Learning. (1995) 252–260

[49] Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric TSPs by ant colonies. In: IEEE Conference on Evolutionary Computation, IEEE Press (1996) 622–627

[50] Stützle, T., Hoos, H.: Improving the ant system: A detailed report on the max-min ant system. Technical Report AIDA-96-12 (1996)

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

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

[53] Gambardella, L.M., Taillard, E.D., Agazzi, G.: MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. In: New Ideas in Optimization. McGraw-Hill (1999) Also available as technical report IDSIA-06-99, IDSIA, Lugano, Switzerland.

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

[55] Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers & Operations Research **13** (1986) 533–549

[56] Osman, I.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research **41** (1993) 421–451

[57] Gendreau, M., Hertz, A., Laporte, G.: A tabu search heuristic for the vehicle routing problem. Management Science **40** (1994) 1276–1290

[58] Taillard, E.D.: Parallel iterative search methods for vehicle routing problems. Networks **23** (1993) 661–673

[59] Volgenat, T., Jonker, R.: The symmetric traveling salesman problem and edge exchanges in minimal 1-tree. European Journal of Operational Research (1983) 394–403

[60] Rochat, Y., Taillard, E.: Probalistic diversification and intensification in local search for vehicle routing. Journal of Heuristics **1** (1995) 147–167

[61] Xu, J., Kelly, J.: A network-flow based tabu search heuristic for the vehicle routing problem. Transportation Science **30** (1996) 379–393

[62] Glover, F.: Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Technical report, Graduate School of Business and Administration, University of Colorado (1991)

[63] Rego, C., Roucariol, C.: A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Meta-Heuristics: Theory and Applications. Kluwer Academic Publishers (1996) 661–675

[64] Rego, C.: A subpath ejection method for the vehicle routing problem. Management Science **44** (1998) 1447–1459

[65] Toth, P., Vigo, D.: The granular tabu search and its application to the vehicle-routing problem. INFORMS Journal on Computing **15** (2003) 333–346

[66] Barbarosoglu, G., Ozgur, D.: A tabu search algorithm for the vehicle routing problem. Computers & Operations Research **26** (1999) 255–270

[67] Cordeau, F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the operational research society **52** (2001) 928–936

[68] Gendreau, M., Hertz, A., Laporte, G., Stan, M.: A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research **43** (1998) 330–335

[69] Holland, J.: Adaptation in Natural and artificial systems. The University of Michigan Press (1975)

[70] Potvin, J.Y.: Genetic algorithms for the traveling salesman problem. Annals of Operations Research **63** (1996) 339–370

[71] Francisco Pereira, Jorge Tavares, P.M.E.C.: Gvr: A new genetic representation for the vehicle routing problem. In: Proceedings of the 13th. Conference on Artificial Intelligence and Cognitive Science (AICS 2002). (2002) 95–102

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

[73] Thangiah, S.: Vehicle Routing with Time Windows using Genetic Algorithms. In: Application Handbook of Genetic Algorithms: New Frontiers, Volume II. CRC Press (1995) 253–277

[74] Thangiah, S., Osman, I., Sun, T.: Hybrid genetic algorithm, simulated annealing and tabu search methods for vehicle routing problems with time windows. Technical Report SRU-CpSc-TR-94-27, Computer Science Department, Slippery Rock University (1994)

[75] Blanton, J., Wainwright, R.: Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: Proceedings of the 5th International Conference on Genetic Algorithms. (1993) 452–459

[76] Potvin, J.Y., Bengio, S.: The vehicle routing problem with time windows – part II: Genetic search. INFORMS Journal on Computing **8** (1996) 165–172

[77] Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Inc., Reading, MA (1989)

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

[79] Zhu, K.: A new genetic algorithm for VRPTW. Presented at IC-AI 2000, Las Vegas, USA (2000)

[80] Kay Chen Tan, Loo Hay Lee, Ke Ou: Hybrid genetic algorithms in solving vehicle routing problems with time windows. Asia-Pacific Journal of Operational Research **18** (2001) 121–130

[81] Potvin, J.Y., Dubé, D.: Improving a vehicle routing heuristic through genetic search. In: International Conference on Evolutionary Computation. (1994) 194–199