Tesis de Licenciatura en Ciencias de la Computación

Un Algoritmo Tabu Search Granular para el Problema de Ruteo de Vehículos con Ventanas de Tiempo y Entregas Parciales

Perosio, Leonardo Luis – LU:823/98
Zunino, Cristian Hernán – LU:851/98

Directora: Dra. Irene Loiseau

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Argentina

Diciembre 2008

Resumen

En esta tesis consideramos el problema de ruteo de vehículos con ventanas de tiempo y entregas parciales (Vehicle Routing Problem with Time Windows and Split Deliveries, VRPTWSD) y proponemos un algoritmo Tabu Search Granular para dicho problema. VRPTWSD es un caso particular del Problema de Ruteo de Vehículos (VRP), donde el inicio del abastecimiento de cada cliente debe estar incluido en una ventana de tiempo, y cada cliente puede ser abastecido por más de un vehículo. En nuestro algoritmo incorporamos dos nuevas operaciones utilizadas en la generación de vecindarios para Tabu Search y las incluimos en una nueva variante de la metaheurística Tabu Search Granular propuesta por Toth y Vigo. Para evaluar la conveniencia del uso de Tabu Search Granular utilizamos las conocidas instancias de Solomon con diversas cantidades de clientes. Logramos mejorar algunos de los mejores valores que aparecen en la literatura y redujimos significativamente el tiempo de ejecución en la mayoría de ellas.

Abstract

We deal here with the Vehicle Routing Problem with Time Windows and Split Deliveries (VRPTWSD). We evaluate the performance of a Granular Tabu Search (GTS) algorithm applied to this problem. VRPTWSD is an extension of Vehicle Routing Problem (VRP) where the service at each customer must start within a time window and a customer may be serviced by more than one vehicle. We propose two new operations between vehicles to generate neighborhoods for the Tabu Search and a new variant of Granular Tabu Search proposed by Toth and Vigo. To evaluate the performance of our Granular Tabu Search we use Solomon instances with different amounts of customers. With our algorithm we were able to improve some of the better results on the literature and we reduce significantly the execution time in all of them.

Índice

1. Introducción

Este trabajo trata el problema de ruteo de vehículos con ventanas de tiempo y entregas parciales. El problema consiste en la generación óptima de rutas para los vehículos que deben abastecer a todo un conjunto de clientes, teniendo en cuenta que cada cliente tiene una franja horaria de entrega y además está la posibilidad de que un cliente pueda ser abastecido por más de un vehículo.

Dado que el problema de ruteo de vehículos es NP-Hard, la variante abordada en este trabajo donde un cliente puede ser abastecido por más de un vehículo es también NP-Hard. Esto quiere decir que no se conoce ningún algoritmo polinomial que pueda encontrar una solución exacta.

Al ser este un problema computacionalmente intratable en la práctica para instancias medianas o grandes, deja como alternativa la utilización de técnicas algorítmicas que puedan encontrar soluciones aproximadas en tiempo polinomial. Algunos ejemplos de estas técnicas son Algoritmos Genéticos, Colonias de Hormigas y Tabu Search.

En este trabajo utilizamos como método general de búsqueda de soluciones una metaheurística basada en Tabu Search, que permite explorar el espacio de soluciones pudiendo finalmente encontrar una solución razonablemente buena (no necesariamente óptima) en un tiempo mucho menor al empleado por los algoritmos exactos.

El resto de este trabajo está organizado de la siguiente manera: en la sección 2 presentaremos el estado del arte y en la sección 3 la formulación matemática del problema tratado y algunas definiciones. En la sección 4 explicaremos las técnicas de Tabu Search y Tabu Search Granular, detallando su aplicación en problemas de ruteo de vehículos. En la sección 5 presentaremos nuestro algoritmo y detallaremos la implementación realizada, incluyendo las variantes de ejecución y las operaciones utilizadas en la generación de soluciones. En la sección 6 explicaremos las variantes implementadas de Tabu Search Granular y en la sección 7 veremos aspectos relacionados con la implementación general y la configuración de la ejecución del algoritmo. En la sección 8 presentaremos los resultados obtenidos y finalmente en la sección 9 presentaremos las conclusiones del trabajo y listaremos algunas posibles continuaciones del mismo.

2. Estado del arte

El Problema de Ruteo de Vehículos (*Vehicle Routing Problem*, VRP) es un problema de optimización combinatoria importante y muy estudiado que surge de aplicaciones logísticas de transporte y distribución reales, tales como ruteo de colectivos escolares, envíos postales y distribución de mercaderías. Toth y Vigo (2002) [18] (Capítulo 7) concluyen que el uso de métodos computarizados en los procesos de distribución reales suelen disminuir los costos de transporte entre un 5 % y un 20 %. Golden, Assad y Wasil (1988) [11] describen varios casos de estudio reales donde la aplicación de algoritmos VRP logra disminuciones sustanciales en los costos de transporte.

En esta tesis consideramos una variante de VRP conocida como Problema de Ruteo de Vehículos con Ventanas de Tiempo y Entregas Parciales (Vehicle Routing Problem with Time Windows and Split Deliveries, VRPTWSD). Dada una flota de vehículos homogéneos (es decir, todos con la misma capacidad) que se encuentran en un depósito y un conjunto de clientes que requieren satisfacer una demanda, el problema consiste en determinar las rutas de los vehículos que salen del depósito y retornan al mismo (es decir, *circuitos*) habiendo satisfecho la demanda de todos los clientes. Las rutas deben respetar las ventanas de tiempo definidas por cada cliente, que especifican cuando comienza y finaliza el inicio del abastecimiento. Algunas veces no es posible que un único vehículo satisfaga la demanda de un cliente, por ejemplo si la demanda supera la capacidad de los vehículos. Por esta razón surge la variante VRPTWSD donde se permiten entregas parciales y, por lo tanto, un cliente puede ser abastecido por más de un vehículo. El objetivo de VRPTWSD es minimizar la distancia total recorrida por los vehículos.

El problema VRP fue introducido por Dantzig y Ramser en 1959 y desde entonces hubo una firme y constante evolución en el diseño de metodologías para solucionar el problema, tanto exactas como aproximadas. Dado que tanto VRP como VRPTWSD son problemas NP-hard, (Lenstra y Rinnooy Kan 1981 [15]), las instancias de tamaño real son muy costosas de resolver óptimamente. Aún hoy en día no se conoce un algoritmo exacto que pueda resolver consistentemente instancias arbitrarias del problema que involucren más de 50 clientes (Golden et al., 1998 [12] y Naddef y Rinaldi, 2002 [16]) y por lo tanto en la mayoría de los casos se utilizan *metaheurísticas* que permiten encontrar buenas soluciones, aunque no óptimas, en un tiempo de ejecución razonable.

En los últimos veinte años se han utilizado varias metaheurísticas en la

solución de VRP. Gendreau, Laporte y Potvin (2002)[9] identifican seis familias de metahurísticas: *simulated annealing*, *deterministic annealing*, *tabu search*, *algoritmos genéticos*, *colonias de hormigas* y *redes neuronales*. Aarts y Lenstra (1997)[1] realizan una introducción general a las metaheurísticas y presentan las características básicas de los paradigmas más importantes. Cordeau y Laporte en [5] resumen diez de los algoritmos Tabu Search más importantes para VRP, brindando una descripción de cada uno y los resultados obtenidos. Los autores señalan además que, si bien el éxito de un método particular depende de las características y variantes implementadas, los resultados de una gran cantidad de experimentos computacionales extensivos llevados a cabo por diversos investigadores concluyen que Tabu Search es la mejor metaheurística para VRP.

En esta tesis abordaremos el problema VRPTWSD presentado por Ho y Haugland [13] y evaluaremos la conveniencia de aplicar la metaheurística Tabu Search Granular propuesto por Toth y Vigo [19]. Presentaremos además una nueva variante de búsqueda granular.

3. Formulación del problema y definiciones

En esta sección definiremos el problema de estudio y la notación utilizada en el resto del trabajo.

Clientes: En el problema VRPTWSD tenemos un conjunto de clientes $C = \{1, 2, ..., n\}$, que residen en n ubicaciones diferentes. Utilizamos el 0 para denotar la ubicación del depósito, con lo cual $N = C \cup \{0\}$ es el conjunto de ubicaciones consideradas por el problema. A cada par de clientes (i, j), donde $i, j \in N$, $i \neq j$, se le asocia un costo de viaje $d_{ij} > 0$ y un tiempo de viaje $t_{ij} > 0$. Cada cliente $i \in C$ tiene una demanda $w_i > 0$. El conjunto $\{d_{ij}\}$ define la matriz de costos del problema. La matriz de costos satisface la desigualdad triangular si y sólo si $d_{ik} + d_{kj} \geq d_{ij} \forall i, j, k \in N$. En nuestro caso, las instancias del problema definen las ubicaciones de los clientes como puntos en el plano y por lo tanto se definen los costos de viaje como la distancia euclidiana entre los puntos. La matriz de costos resultante es simétrica y satisface la desigualdad triangular.

Vehículos: El conjunto de camiones o vehículos homogéneos se denota V, siendo m la capacidad de cada vehículo. En algunas variantes del problema la cantidad de vehículos es una variable a minimizar.

Ventana de Tiempo: Cada cliente i tiene una *ventana de tiempo* o simplemente *ventana*, es decir, un intervalo $[a_i, b_i] \subseteq \Re$, donde a_i y b_i representan los horarios de comienzo y fin del abastecimiento del cliente i. Un vehículo puede arribar a la ubicación del cliente i antes de a_i, pero el abastecimiento no puede comenzar hasta el horario de apertura de la ventana de tiempo a_i. Un vehículo no puede arribar al cliente i después de la finalización de la ventana de tiempo b_i. El depósito también tiene una ventana de tiempo $[a_0, b_0]$, donde a_0 representa el horario en que los vehículos salen del depósito y b_0 representa el horario en que los vehículos deben retornar al depósito.

Entregas Parciales: En los problemas VRP tradicionales cada cliente debe ser abastecido por un único vehículo, mientras que en las variantes con Split Deliveries (VRPTWSD) la demanda de un cliente puede ser satisfecha por más de un vehículo. Esto claramente debe ocurrir en aquellas instancias donde la demanda de algún cliente sea mayor que la capacidad de los vehículos pues de lo contrario no podría construirse una solución válida

al problema, pero también puede aplicarse cuando resulta más económico abastecer al cliente con varios vehículos.

Para ejemplificar la idea, vamos a utilizar un caso presentado por Archetti, Speranza y Hertz en [2]. La figura 1 muestra una instancia con cuatro clientes 1, 2, 3 y 4 representados con círculos y el depósito representado con un cuadrado. Esta notación será utilizada a lo largo de todo el trabajo. Para esclarecer los ejemplos incluiremos dos veces el depósito en la representación de rutas indicando su inicio y su fin. En el ejemplo cada cliente tiene una demanda de valor 3 y cada vehículo tiene una capacidad de valor 4.

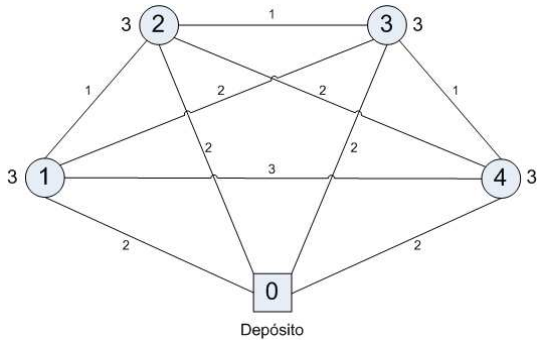


Figura 1: Instancia de ejemplo de VRP

La figura 2 muestra una posible solución al problema con cuatro rutas (o vehículos) R_1, R_2, R_3 y R_4. Cada vehículo visita un único cliente, lo abastece completamente y retorna al depósito. Cada ruta tiene un valor asociado de 4, siendo 16 el costo total de la solución.

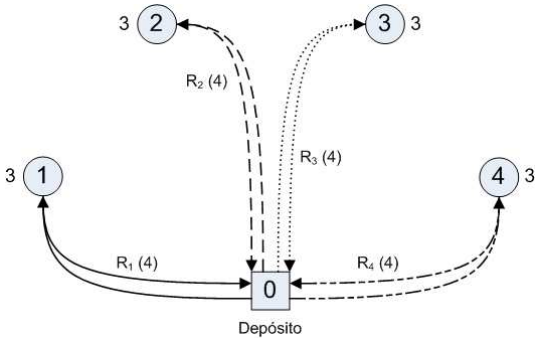


Figura 2: Una solución VRP

Sin embargo, a cada vehículo le queda una carga disponible de valor 1 que no puede aprovechar debido a que las demandas de los clientes tienen valor 3. Esta situación puede mejorarse si se permiten entregas parciales, y la solución anterior puede reemplazarse por una solución con 3 vehículos como se muestra en la figura 3.

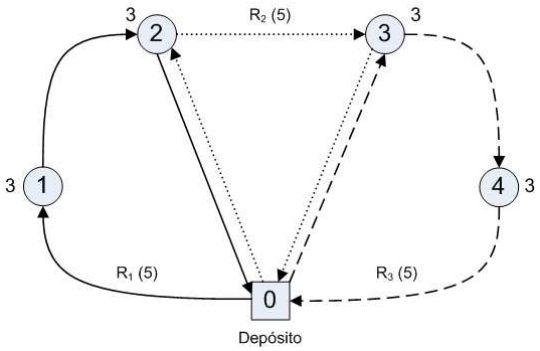


Figura 3: Una solución VRPSD

En este caso el costo de cada ruta es 5 (2+1+2), siendo 15 el costo total de la solución. Es decir, la inclusión de entregas parciales como parte de la solución permitió encontrar una solución mejor a una instancia que puede resolverse sin entregas parciales.

Objetivo: El objetivo del problema será determinar el conjunto de rutas de costo mínimo que satisfagan las demandas de los clientes respetando las restricciones de tiempo y capacidades de los vehículos.

3.1. Modelo de Programación Lineal Entera

Este problema puede modelarse matemáticamente como un problema de programación lineal entera. A continuación presentamos el modelo propuesto por Ho y Haugland en [13].

El modelo consta de tres conjuntos de variables de decisión: x, f y s. Para cada par de clientes (i, j), donde $i, j \in N, i \neq j$ y para cada vehículo k, definimos x_{ijk} como

$$x_{ijk} = \begin{cases} 1 & \text{si el vehiculo } k \text{ viaja directamente del cliente } i \text{ al cliente } j \\ 0 & \text{sino} \end{cases}$$

La variable de decisión f_{ik} está definida para cada cliente i y cada vehículo k y denota la fracción de demanda del cliente i abastecido por el vehículo k. La última variable de decisión s_{ik} indica el tiempo de inicio del abastecimiento del vehículo k al cliente i. Asumimos $s_{0k} = a_0 \forall k$.

VRPTWSD puede definirse matemáticamente como:

$$\text{minimizar } z(x) = \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ijk} \tag{1}$$

sujeto a

$$\sum_{j \in N} x_{0jk} = 1 \qquad \forall k \in V \tag{2}$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0 \qquad \forall h \in C, \forall k \in V \tag{3}$$

$$\sum_{k \in V} f_{ik} = 1 \qquad \forall i \in C \tag{4}$$

$$\sum_{i \in C} w_i f_{ik} \leq m \qquad \forall k \in V \tag{5}$$

$$\sum_{j \in N} x_{jik} \geq f_{ik} \qquad \forall i \in C, \forall k \in V \tag{6}$$

$$s_{ik} + t_{ij} - K_{ij}(1 - x_{ijk}) \leq s_{jk} \qquad \forall j \in C, \forall i \in N, \forall k \in V \tag{7}$$

$$a_i \leq s_{ik} \leq b_i \qquad \forall i \in N, \forall k \in V \tag{8}$$

$$s_{ik} + t_{i0} - K_{i0}(1 - x_{iok}) \leq b_0 \qquad \forall i \in C, \forall k \in V \tag{9}$$

$$f_{ik} \geq 0 \qquad \forall i \in C, \forall k \in V \tag{10}$$

$$s_{0k} = a_0 \qquad \forall k \in V \tag{11}$$

$$x_{iik} = 0 \qquad \forall i \in C, \forall k \in V \tag{12}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in N, \forall k \in V \tag{13}$$

Se define la *función objetivo* a minimizar (1) como la distancia total recorrida para abastecer a todos los clientes. Los siguientes dos conjuntos de restricciones (2) y (3) indican que cada vehículo sale del depósito, después de llegar a un cliente sale nuevamente, y finalmente regresa al depósito. La restricción (4) asegura que cada cliente recibe la demanda total y (5) que la suma de los abastecimientos de cada vehículo no supera su capacidad. El conjunto de condiciones en (6) asegura que un cliente puede ser abastecido solamente por un vehículo que visita al cliente. Las inecuaciones en (7) indican que el vehículo k no puede arribar al cliente j antes de $s_{ik} + t_{ij}$ si es que viaja directamente desde el cliente i al cliente j. Para esto, la constante K_{ij} se define lo suficientemente grande, como por ejemplo $K_{ij} = b_i + t_{ij} + a_j$. Las condiciones en (8) aseguran que cada cliente sea abastecido dentro de su ventana de tiempo. Finalmente la condición (9) obliga a cada vehículo a retornar al depósito antes de que su ventana de tiempo termine.

Puede notarse que la suma total de las condiciones en (6) sobre todos los vehículos combinada con la condición (4) derivan en la siguiente condición:

$$\sum_{k \in V} \sum_{i \in N} x_{ijk} \geq 1 \qquad \forall j \in C \qquad (14)$$

implicando que cada cliente es abastecido por lo menos una vez. El siguiente conjunto de restricciones indica que todos los vehículos deben retornar al depósito:

$$\sum_{i \in N} x_{i0k} = 1 \qquad \forall k \in V \qquad (15)$$

pero fueron omitidas del modelo porque está implicadas por las ecuaciones (2) y (3).

3.2. Notación y definiciones

Dada una solución factible (x, f, s) al problema VRPTWSD expresado por (1)-(13): $(x, f, s) \in B^{|N|\times|N|\times v} \times \Re_{\geqslant 0}^{|N|\times v} \times \Re_{\geqslant 0}^{|N|\times v}$, $B = \{0, 1\}$, cada vehículo $k \in V$ define un conjunto ordenado y único $R_k = (0, c_1^k, c_2^k, ..., c_g^k, 0)$ tal que $x_{0c_1^k k} = x_{c_1^k c_2^k k} = ... = x_{c_g^k 0k} = 1$. Nos referiremos a R_k como la *ruta* del vehículo k (inducida por x). Por lo tanto, a partir de ahora utilizaremos los términos *ruta* y *vehículo* de manera indistinguible.

Para simplificar la notación, asumimos en adelante que $c_i^k = R_k[i]$ indicando de esta manera el cliente en la i-ésima posición de la ruta k. Por

ejemplo, si $R_3 = (0, c_1^3, c_2^3, c_3^3, c_4^3, 0) = (0, 8, 17, 33, 11, 0)$, entonces el segundo cliente que visita R_3 es $R_3[2] = 17$.

Entonces una solución factible (x, f, s) al problema queda determinada por:

- un conjunto de rutas
- una asignación de clientes para cada ruta (representado por $x = x_{ijk}$)
- el abastecimiento de cada cliente en cada ruta (representado por $f = f_{ik}$)
- el inicio del abastecimiento de cada cliente en cada ruta (representado por $s = s_{ik}$)

A continuación definiremos algunos conceptos que serán utilizados ampliamente a lo largo del trabajo. Decimos que una ruta es *válida* si comienza y termina en el depósito, respeta los horarios de llegada a cada cliente (incluyendo el retorno al depósito) y respeta la carga del vehículo (es decir, la suma de los abastecimientos de los clientes que visita es menor o igual a la capacidad del vehículo). Una solución es *válida* o factible si todas las rutas que la componen son válidas y abastecen las demandas de todos los clientes. O sea, una solución es válida si cumple las condiciones (1)-(13) definidas previamente.

Una forma muy común de representar los problemas de este tipo es mediante un grafo $G = (N, A)$ donde N representa al conjunto de clientes más el depósito, es decir $N = C \cup \{0\}$, y los arcos en A representan la conectividad entre clientes. El costo de un arco se define como la distancia entre los clientes que interconecta.

3.3. Propiedad de las soluciones del VRPTWSD

Una característica de una solución óptima del VRPTWSD es que no hay dos rutas que comparten más de un cliente en común. Dror y Trudeau (1989, 1990) [6] [7] introducen esta propiedad y la demuestran en el caso de VRPSD. Ho y Haugland [13] demuestran que también se cumple en el caso de VRPTWSD y presentan una manera de redistribuir las cargas para obtener una solución de menor costo, la cual explicaremos a continuación.

Sea $S = (x, f, s)$ una solución del VRPTWSD donde $\{t_{ij}\}_{i,j \in N}$ satisfacen la desigualdad triangular, y sean $R_1, ..., R_v$ las rutas inducidas por x. Supongamos que $i, j \in R_k \cap R_l$, $i, j \in C$, $i \neq j$ y $1 \leqslant k < l \leqslant v$. Es decir, dos clientes distintos i y j son abastecidos en forma parcial por dos vehículos distintos k y l. Sea $q_{ik} = f_{ik} w_i$ el abastecimiento del cliente i por parte del vehículo k. Sin pérdida de generalidad, asumimos que $q_{ik} = \text{mín}\{q_{ik}, q_{il}, q_{jk}, q_{jl}\}$. Generamos entonces una nueva solución $\hat{S} = (\hat{x}, \hat{f}, \hat{s})$ donde redefinimos los abastecimientos de la siguiente manera:

$\hat{x} = x$ y $\hat{q} = q$ para todos los componentes con las siguientes excepciones:

- $\hat{q}_{jk} = q_{jk} + q_{ik}$
- $\hat{q}_{jl} = q_{jl} - q_{ik}$
- $\hat{q}_{il} = q_{il} + q_{ik}$
- $\hat{q}_{ik} = 0$
- $\hat{x}_{i-1,i+1,k} = 1$
- $\hat{x}_{i-1,i,k} = 0$
- $\hat{x}_{i,i+1,k} = 0$

De esta manera redistribuimos las cargas y eliminamos el abastecimiento del vehículo k al cliente i, como muestra el ejemplo de la figura 4.

Es fácil notar que si S es válida (es decir, si respeta las condiciones de tiempos y cargas del problema) la nueva solución $\hat{S}$ también lo es ya que no se agregan nuevas condiciones de tiempo y las cargas se distribuyen en forma tal de mantenerse constantes.

Como puede verse en el ejemplo numérico de la figura 5, las cargas de los vehículos son iguales en ambas soluciones (k abastece $5 + 10 = 0 + 15$ y l abastece $20 + 30 = 25 + 25$), y lo mismo ocurre con los abastecimientos de los clientes (i recibe $5 + 20 = 0 + 25$ y j recibe $10 + 30 = 15 + 25$). Además, en $\hat{S}$ se eliminó la visita al cliente i por parte del vehículo k y por lo tanto los ejes $(R_k[i-1], R_k[i])$ y $(R_k[i], R_k[i+1])$ se reemplazan por el eje $(R_k[i-1], R_k[i+1])$. Como suponemos que las distancias son euclidianas, se cumple que $d_{i-1,i+1} \leqslant d_{i-1,i} + d_{i,i+1}$ y por lo tanto el costo de la nueva solución es menor, es decir, $z(\hat{S}) \leqslant z(S)$.

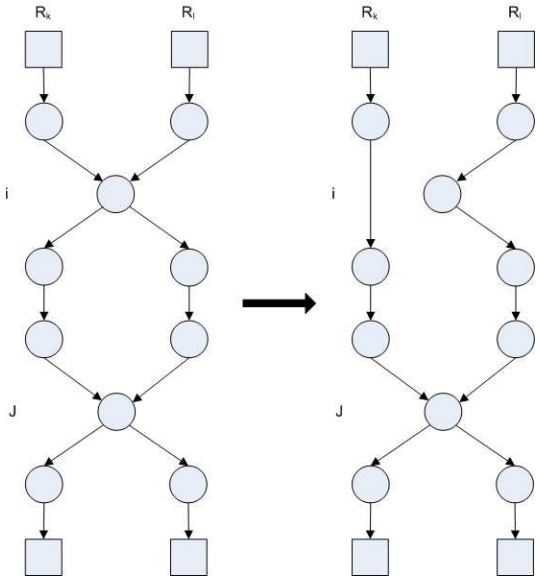


Figura 4: Eliminación de un cliente en común

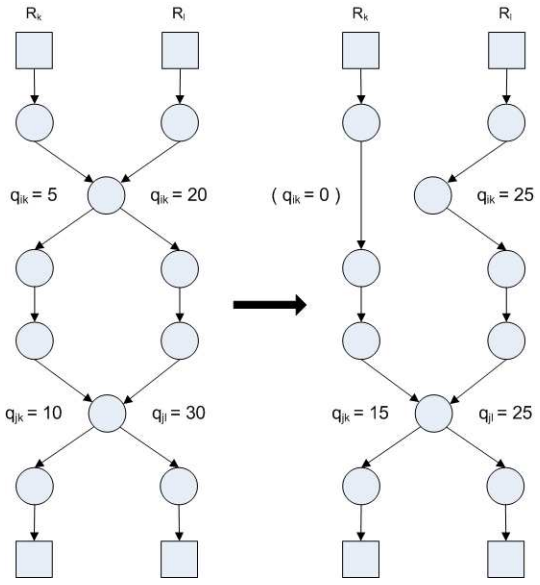


Figura 5: Ejemplo numérico de distribución de cargas

4. Tabu Search

Los métodos de búsqueda local (*local search*) son métodos iterativos que empiezan desde una solución inicial S_0, generalmente factible, y generan una secuencia de soluciones S_i eligiendo en cada paso la *mejor* solución vecina. Cada solución S en el espacio de soluciones tiene un costo asociado $z(S)$, siendo una solución S_1 mejor que otra S_2 sólo si $z(S_1) < z(S_2)$. Por lo tanto, la secuencia generada cumple la condición $z(S_i) < z(S_{i-i})\ \forall i$ siendo cada nueva solución mejor que la anterior. En algún punto de este proceso el método alcanzará indefectiblemente un mínimo local y no podrá encontrar una solución mejor, punto en el cual finalizará la búsqueda convirtiéndose el mínimo local en la mejor solución hallada.

El *vecindario* (*neighborhood*) de una solución es el conjunto de otras soluciones que pueden ser alcanzadas desde dicha solución por medio de algún *movimiento*, generalmente simple. La cardinalidad de un vecindario es la cantidad de soluciones vecinas a la solución actual. Según Toth y Vigo en [19] el tiempo de ejecución de cada iteración de un algoritmo de búsqueda local depende principalmente de la cardinalidad del vecindario y del tiempo necesario para generar soluciones, verificar las factibilidades y evaluar sus costos. En la mayoría de los casos el tiempo de cada iteración está acotado por una función polinómica en función del tamaño de la instancia. El número de iteraciones a ejecutar para alcanzar el mínimo local puede ser grande y en el peor caso crece exponencialmente en función del tamaño de la instancia.

La figura 6 ilustra un ejemplo de una búsqueda local que empieza en una solución inicial S_0 y luego de tres iteraciones finaliza en un mínimo local en S_3.

El mayor problema de este enfoque es que el mínimo local hallado puede estar bastante alejado del mínimo global o absoluto al que se quiere llegar. Una manera posible de evitar "trabarse" en el mínimo hallado es incluir movimientos que no mejoren y cambiar la condición de selección de soluciones vecinas permitiendo soluciones peores (es decir, de mayor costo). La figura 7 ilustra esta idea.

El problema que surge ahora es que en la próxima iteración (o eventualmente en alguna iteración futura) se volverá a seleccionar la solución del mínimo local ya que tendrá el menor costo, generándose un ciclo continuo. La figura 8 ilustra la aparición de un ciclo al permitir un movimiento de no-mejora: las soluciones pares intentan escapar del mínimo local pero las impares vuelven a seleccionar la solución anterior. La manera de evitar este

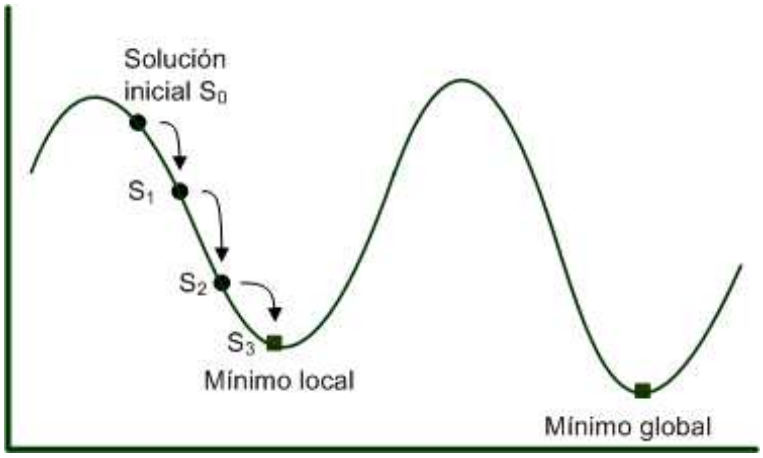


Figura 6: Método de búsqueda local

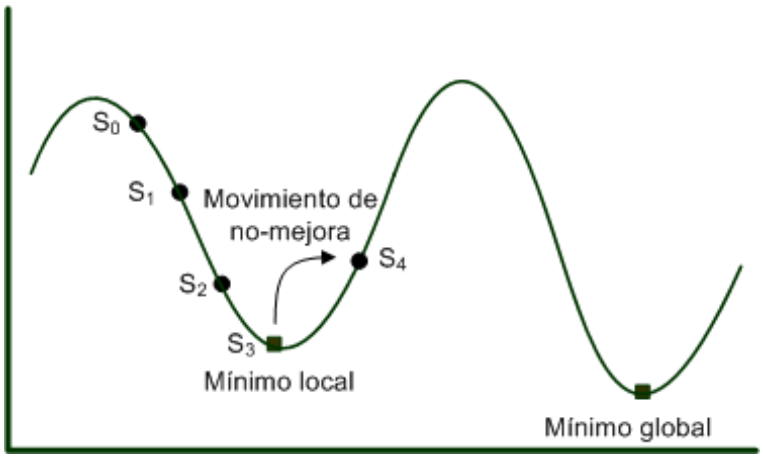


Figura 7: Aplicación de un movimiento de no-mejora

comportamiento es incluir algún mecanismo de "memoria"para recordar las soluciones visitadas y prohibirlas por un cierto número de iteraciones. De esta forma logramos la exploración por otras zonas del espacio de soluciones, como veremos a continuación.

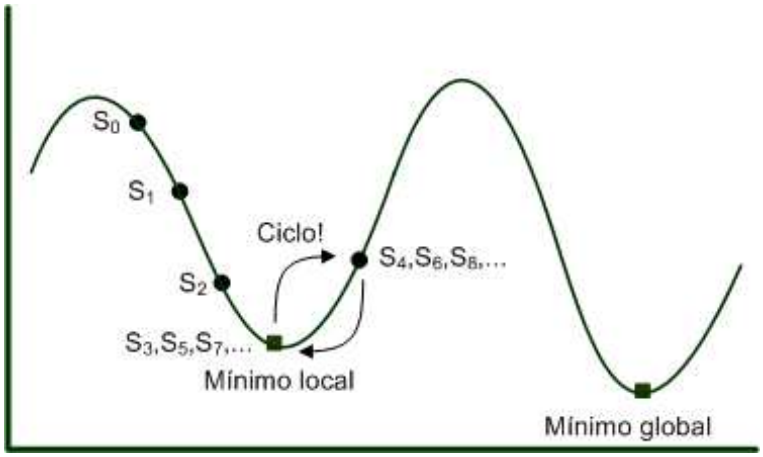


Figura 8: Aparición de un ciclo

Tabu Search es una estrategia de búsqueda propuesta originalmente por Glover en 1989 que adopta una estrategia específica para poder "escapar"de un mínimo local y continuar la búsqueda por otras soluciones posiblemente mejores. Desde su aparición Tabu Search se convirtió en una de las mejores y más utilizadas estrategias de búsqueda en problemas de optimización combinatoria debido a su gran versatilidad, eficacia y simplicidad de implementación. Glover y Laguna (1997) en [10] realizan un análisis extensivo de las aplicaciones de Tabu Search.

En cada iteración el método realiza una exploración del espacio de soluciones moviéndose de una solución S_i en la iteración i a una solución vecina S_{i+1} perteneciente al vecindario $N(S_i)$ de S_i, aún en caso de que S_{i+1} sea peor que S_i logrando de esta forma escapar del mínimo local. Sin embargo, como justamente S_{i+1} no necesariamente será una solución mejor se debe utilizar algún mecanismo para evitar ciclar sobre soluciones ya visitadas. Para este fin Tabu Search mantiene una estructura de memoria denominada *lista tabú* con el historial de soluciones visitadas o movimientos realizados en el pasado. Como almacenar el historial completo de soluciones visitadas es algo costoso

y engorroso, en la práctica se opta por almacenar los atributos que identifican los movimientos que generaron dichas soluciones. De esta forma, cuando el algoritmo intenta realizar un movimiento que figura en la lista tabú (en cuyo casos decimos que es un *movimiento tabú*), el movimiento se prohíbe obligando al algoritmo a explorar otras soluciones. Estas prohibiciones no son definitivas sino que se mantienen por una cierta cantidad de iteraciones (*tenure*). Sin embargo, la decisión de prohibir un movimiento no es estricta y puede ignorarse si se cumple algún *criterio de aspiración* preestablecido. Por ejemplo, uno de los criterios de aspiración más utilizados es el siguiente: si el valor de la solución tabú es mejor que el valor de la mejor solución hallada hasta el momento, es evidente que esta solución no ha sido visitada y por lo tanto es aceptada ignorándose la restricción de la lista tabú.

La figura 9 ilustra la inserción de Tabu Search en los ejemplos anteriores. Las primeras iteraciones hasta S_3 son las mismas que antes, luego se permite una secuencia de soluciones que empeoran y finalmente en S_6 se inicia otra secuencia de soluciones de mejora que finaliza en el mínimo absoluto buscado en S_9. Las flechas punteadas indican los movimientos considerados como tabú por intentar elegir soluciones ya visitadas. Estos movimientos prohibidos obligan a explorar nuevas soluciones (peores en este caso) que en el ejemplo resultan imprescindibles para hallar el mínimo absoluto.

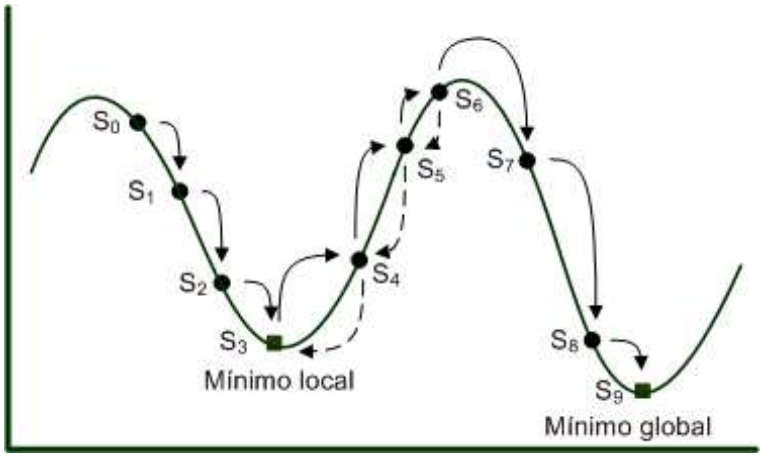


Figura 9: Exploración de otras soluciones

Aunque esta simple descripción de Tabu Search ya es una mejora con respecto a los métodos de búsqueda local, aún así pueden requerirse muchas

iteraciones antes de que se puedan detectar soluciones de buena calidad. Por lo tanto, las implementaciones prácticas generalmente se despegan de este enfoque básico e implementan ingredientes o características particulares que incrementan la efectividad y la performance a expensas de un agregado considerable de complejidad. Esto implica la existencia de varios parámetros en juego que deben ser ajustados experimentalmente.

Como mencionamos antes, los algoritmos Tabu Search son hoy en día una de las mejores opciones para la mayoría de los problemas difíciles de optimización. Sin embargo, los tiempos de ejecución necesarios para detectar buenas soluciones suelen ser muy grandes en comparación con los métodos tradicionales, como las heurísticas constructivas, que requieren tiempos de ejecución relativamente cortos. Toth y Vigo en [19] introducen un método de intensificación de búsqueda denominado *Tabu Search Granular (GTS)* que puede utilizarse en Tabu Search o en otras heurísticas/metaheurísticas y puede aplicarse en una gran cantidad de problemas de grafos y optimización combinatoria. El método, explicado en detalle en la próxima sección, se basa en el uso de vecindarios restringidos denominados *granular neighborhoods* obtenidos a partir de la eliminación de una gran cantidad de ejes de los vecindarios tradicionales que se consideren de poca probabilidad de pertenecer a buenas soluciones. De esta manera no sólo se orienta la búsqueda en un espacio mucho más acotado sino que además en cada iteración se evalúan movimientos más prometedores de buenas soluciones. Ya que la búsqueda se realiza sobre un conjunto de soluciones vecinas "de elite"y el criterio de selección se fija de antemano, los autores indican que los vecindarios granulares pueden verse como una implementación particular de la estrategia *candidate-list* (ver Glover y Laguna (1997) [10] que es un ejemplo de una posible característica a incluir en un algoritmo general Tabu Search.

4.1. Esquema General de Tabu Search

A continuación presentamos un pseudocódigo del esquema general de Tabu Search.

Algorithm 1: Esquema general de Tabu Search

Generar una solución inicial S_0
Setear $S^* = S_0$
Setear $i = 0$
Inicializar la lista tabú
repeat
  Generar un vecindario $N(S_i)$ de S_i
  Elegir el subconjunto N^* de vecinos en $N(S_i)$ que respetan las condiciones tabú o cumplen el criterio de aspiración
  Elegir el mejor vecino $j \in N^*$ con respecto a $z(j)$
  Actualizar la lista tabú
  Setear $i = i + 1$
  Setear $S_i = j$
  Si $z(j) < z(S^*)$, setear $S^* = j$

until *Hasta que se cumpla algún criterio de parada* ;

El algoritmo parte de una solución inicial S_0 desde la cual se empieza a recorrer el espacio de soluciones. La solución inicial puede generarse aleatoriamente, con algún método rápido, con algoritmos golosos o de cualquier otra manera. En cada paso el algoritmo genera un vecindario y elige la mejor solución vecina j que no sea tabú o que lo sea pero cumpla el criterio de aspiración para reemplazar a la solución actual S_i. Si además esta nueva solución supera en calidad a la mejor solución obtenida hasta el momento S^* (es decir, $z(j) < z(S^*)$) entonces se reemplaza S^* por j. Luego se actualiza el estado de la lista tabú que en general consiste en marcar como tabú por una cierta cantidad de iteraciones al movimiento inverso que se utilizó para pasar de S_i a j.

Este procedimiento iterativo se realiza hasta que se cumpla algún criterio de parada, como pueden ser cantidad máxima de iteraciones, cantidad de iteraciones consecutivas sin obtener una mejora o una combinación de ambas, y finalmente se obtiene en S^* la mejor solución hallada.

4.2. Tabu Search Granular

Las heurísticas Tabu Search para problemas de ruteo de vehículos necesitan realizar varios de cientos de iteraciones para obtener soluciones de buena calidad. Cada iteración consiste en explorar los vecindarios antes mencionados a través de operaciones de intercambio de vecinos. Las operaciones que utilizamos en este trabajo para generar soluciones vecinas serán explicadas en detalle en la siguiente sección.

Varios enfoques han sido propuestos en la literatura para reducir el tiempo computacional de la exploración de vecindarios, tales como *random sampling*, *candidate-list strategies* y *parallel implementations of the search*. Toth y Vigo en [19] definen una implementación efectiva perteneciente a la familia de candidate-list denominada Tabu Search Granular que lleva a una reducción drástica en el tiempo computacional requerido por cada iteración del algoritmo Tabu Search. El objetivo es alcanzado utilizando vecindarios que pueden ser examinados en un tiempo mucho menor al requerido en la forma tradicional pero sin afectar considerablemente la calidad de la soluciones halladas. El método propone generar vecindarios acotados descartando de los vecindarios tradicionales una gran cantidad de movimientos no prometedores y explorando sólo un subconjunto que contiene movimientos más prometedores.

Los autores indican algunos aspectos positivos de Tabu Search Granular. En primer lugar este método suele ser una de las formas menos intrusiva de modificar el enfoque de búsqueda de buenas soluciones manteniendo las características restantes intactas, particularmente la estructura básica de los vecindarios utilizados durante la búsqueda. Además, aumenta la aplicabilidad de Tabu Search ya que simplifica su extensión a otros problemas donde Tabu Search ha demostrado ser efectivo en la obtención de buenos resultados pero ineficiente en términos de requerimientos temporales. Por último, el método permite evaluar sus beneficios en forma directa ya que es fácil de comparar con un algoritmo Tabu Search que no aplique vecindarios granulares.

4.2.1. Tabu Search Granular en VRP

Cuando el problema VRP se define sobre un grafo completo (como suele ocurrir generalmente, y particularmente en nuestro caso) puede observarse que los arcos "largos"tienen poca probabilidad de formar parte de buenas soluciones. En este punto vamos a referirnos al costo de un arco como su

longitud. Toth y Vigo en [19] presenta la siguiente tabla con algunos problemas clásicos de VRP. La tabla contiene el nombre de la instancia, el número de clientes n, la cantidad de vehículos disponibles K y el valor de la mejor solución z^*. Además, el costo promedio de los arcos de la mejor solución, $\overline{z}^* = \frac{z^*}{(n+K)}$ es comparado con el costo mínimo, máximo y promedio del grafo completo.

					costo de los arcos		
Problema	n	K	z^*	$\overline{z}^*$	min	max	promedio
E051-05e	50	5	524.61	954	2.24	85.63	33.75
E076-10e	75	10	835.26	9.83	2.24	85.28	34.13
E101-08e	100	8	826.14	7.65	1.41	91.83	34.64
E151-12c	150	12	1028.42	6.35	0.00	91.83	33.92
E200-17c	199	17	1291.45	5.98	0.00	91.83	33.24
E101-10c	100	10	819.56	7.45	1.00	96.18	40.27
E121-07c	120	7	1042.11	8.21	0.00	114.98	54.53

La tabla muestra claramente que la gran mayoría de los arcos del grafo tienen un costo mucho mayor a $\overline{z}^*$. Los autores también indican que un comportamiento similar se observa en la gran mayoría de los problemas VRP conocidos. Por lo tanto, una manera de acelerar la exploración de los vecindarios es limitar lo mejor posible la evaluación de movimientos que intentan insertar arcos "largos", en la solución actual.

El procedimiento de generación de vecindarios granulares consiste en comenzar con el grafo original completo $G = (N, A)$ para luego definir un nuevo grafo ralo $G' = (N, A')$ con $|A'| << n^2$. Este nuevo grafo ralo incluye todos los arcos que deben ser considerados en la solución actual, por ejemplo, todos los arcos "cortos", y un subconjunto relevante de arcos importantes I tales como los que inciden en el depósito y los que pertenecen a las mejores soluciones encontradas hasta el momento. En otras palabras,

$$A' = \{(i, j) \in A : c_{ij} \leq \nu\} \cup I$$

Luego, la búsqueda en un vecindario granular considera sólo los movimientos que pueden generarse mediante arcos pertenecientes a G', es decir, los movimientos que involucran al menos algún arco "corto". En la implementación utilizada en [19] adoptan una regla simple para definir y obtener los arcos "cortos". Un arco es "corto"y será incluido en G' si su costo es menor o igual a un *granularity threshold* definido como:

$$\nu = \beta \frac{z'}{(n + K)}$$

donde β es el *parámetro de esparcificación* (*sparsification*) y z' es el valor de la solución heurística determinada por el algoritmo de Clarke y Wright (1964).

Las experiencias computacionales llevadas a cabo por Toth y Vigo demostraron que un algoritmo básico Tabu Search que implemente vecindarios granulares es capaz de determinar soluciones de buena calidad en tiempos de ejecución comparables a los obtenidos mediante heurísticas constructivas. Además, los experimentos realizados sobre el rango de valores de β entre 0.5 y 5.0 demostraron, sorpresivamente, que la calidad de la solución no es una función monótona creciente del parámetro de esparcificación (es decir, mejores resultados a partir de mayores cálculos computacionales con valores mayores de β) sino que los mejores resultados fueron obtenidos con β entre 1 y 2.5. Con estos valores fueron seleccionados entre un 10 % y un 20 % de los arcos del grafo completo G.

4.2.2. Diversificación e intensificación en GTS

Una característica importante asociada con el paradigma granular es que una modificación dinámica de la estructura del grafo ralo asociado al vecindario granular provee una forma simple de incluir intensificación y diversificación durante la búsqueda. Por ejemplo, modificando el parámetro β se altera el número de arcos actualmente incluidos en el grafo ralo y como consecuencia es posible encontrar una solución diferente al finalizar la búsqueda. Para esto, el algoritmo Tabu Search puede alternar entre varias iteraciones intensificadas, asociadas con un valor pequeño de β, y algunas pocas iteraciones diversificadas donde β es incrementado en forma considerable para favorecer la inclusión de nuevos arcos (que serán más largos) en las soluciones evaluadas.

5. Nuestro Algoritmo

En esta sección mostraremos en forma detallada nuestro algoritmo, incluyendo la generación de la solución inicial y la forma en la que se crean y se modifican las rutas durante la ejecución. Además explicaremos la forma en que se generan los vecindarios mediante los movimientos definidos por las seis operaciones que desarrollamos y finalmente mencionaremos detalles de implementación, donde podemos destacar los criterios de movimiento Tabu que utiliza cada una de las operaciones.

5.1. Esquema General

Algorithm 2: Esquema general de nuestro algoritmo

Leer la instancia a ejecutar
Leer parámetros de criterios de parada
Leer operaciones a utilizar $op_1, ..., op_v$ y *tenure* de Tabu Search
Leer parámetros de ejecución asociados a Tabu Search Granular
Generar e inicializar el grafo G que representa la instancia
Generar la solución inicial S_0
Setear $S^* = S_0$
Inicializar la lista tabú
Setear $i = 0$
while *no se cumpla algún criterio de parada* **do**
    Ajustar G según los parámetros de Tabu Search Granular
    for *cada operación* $op \in \{op_1, ..., op_v\}$ *especificada* **do**
        $N_{op}(S_i) = GenerarVecindario(S_i, op, G)$
    end
    Setear $m_{op} = ElegirMejorMovimiento(\{N_{op}(S_i)\})$
    Generar una nueva solución $S_{i+1} = AplicarMovimiento(m_{op}, S_i)$
    Descontar el tenure de los movimientos presentes en la lista tabu
    Marcar como tabú el movimiento inverso de m_{op} por *tenure* iteraciones
    if $z(S_{i+1}) < z(S^*)$ **then**
        Setear $S^* = S_{i+1}$
    end
    Setear $i = i + 1$
end
Aplicar *Propiedad* a S^*
Retornar S^*

Al comenzar la ejecución el programa lee los parámetros especificados en los archivos de configuración, inicializa la lista tabú y genera el grafo que

representa la instancia a ejecutar. El algoritmo puede aplicarse sobre una instancia en particular o en forma secuencial sobre una lista de instancias.

El primer paso del algoritmo consiste en generar la solución inicial S_0 que sirve de base en la búsqueda de soluciones. Nuestra solución inicial, que será explicada en detalle en las próximas secciones, consiste en un método goloso que genera una solución inicial factible con posibles entregas parciales. La solución hallada se almacena inicialmente en S^*, que representa la mejor solución encontrada hasta el momento y contendrá la solución definitiva al finalizarse el procedimiento.

Luego comienza el proceso iterativo basado en Tabu Search que recorre el espacio de soluciones intentando en cada paso mejorar la solución hallada. En cada iteración i el algoritmo verifica si debe ajustar las condiciones de vecindad entre clientes según la variante de granular especificada, la iteración actual y el comportamiento de la ejecución observado hasta el momento. Este ajuste se aplica sobre el grafo de vecindad y el resultado impacta directamente en el método $esVecino(cli_1, cli_2)$ que será explicado más adelante.

A continuación empieza el proceso de generación y evaluación de vecindarios. Por cada operación *op* especificada en el archivo de parámetros, el algoritmo genera un vecindario de la solución actual S_i aplicando cada combinación posible de los movimientos definidos por *op* sobre S_i. En nuestro caso implementamos seis operaciones diferentes para generar soluciones vecinas de diversas características. Claramente cada operación define un vecindario distinto, siendo algunos más prometedores de buenas soluciones que otros dependiendo principalmente de la instancia del problema y de la operación misma. En las secciones siguientes explicaremos cada operación en detalle.

A modo de ejemplo sencillo y minimalista, supongamos que se aplica sobre S_i la operación Relocate que consiste en mover un cliente de un vehículo a otro. Si S_i contiene tres rutas R_1, R_2, y R_3, cada una con un único cliente c_1, c_2 y c_3 respectivamente, entonces los posibles movimientos son pasar cada c_i al principio o al final de cada otra ruta, generándose entonces 12 soluciones vecinas a S_i que constituyen su vecindario $N(S_i)$.

Volviendo al método general, el vecindario resultante contiene sólo las soluciones válidas que son vecinas a S_i ya que las soluciones inválidas son descartadas en el momento de la generación. Las soluciones válidas resultantes pueden tener un costo menor, igual o mayor al costo de la solución actual, e incluso pueden encontrarse soluciones tabú generadas a partir de movimientos tabú que serán eventualmente descartadas en el siguiente paso.

Una vez finalizada la generación de los vecindarios de S_i por cada ope-

ración, digamos $N_{op}(S_i)$ para toda *op* especificada, se procede a identificar la nueva solución S_{i+1} que reemplazará a S_i en el procedimiento de búsqueda. Para esto se elige la solución S' de menor costo que no sea tabú o que lo sea pero cumpla el criterio de aspiración. Es decir, $S_{i+1} = S' = argmin\{z(S) \| \neg(EsTabu(S)) \vee (EsTabu(S) \wedge z(S) < z(S^*)), S \in N_{op}(S_i) \forall op \in \{op_1, ..., op_v\}\}$. Recordamos que el criterio de aspiración permite seleccionar una solución tabú en caso de que su costo sea menor que el costo de la mejor solución S^* hallada hasta el momento.

Una vez elegida la nueva solución S_{i+1} se procede a marcar como tabú por *tenure* iteraciones al movimiento inverso del utilizado para generar dicha solución evitándose que se vuelva a la solución anterior S_i por *tenure* iteraciones. Esto es necesario debido a que el proceso de selección anterior no puede garantizar que S_{i+1} sea mejor que S_i: si todos los vecindarios contienen soluciones peores, es decir $z(S_i) < z(S') \forall S' \in N_{op}(S_i), op \in \{op_1, ..., op_v\}$, es necesario elegir una solución peor con $z(S_{i+1}) > z(S_i)$ para poder salir del mínimo local en S_i y poder continuar la exploración por otras zonas del espacio de soluciones. Tal como se explicó en la sección anterior, esta es la característica principal por la cual Tabu Search puede continuar la exploración por soluciones peores sin necesidad de trabarse en mínimos locales.

Por último, se descuenta en uno el valor del *tenure* de los movimientos restantes presentes en la lista tabú y se verifica si la nueva solución es mejor que la encontrada hasta el momento; en caso de ser así se actualiza el valor de S^* reemplazándolo por S_{i+1}.

Este procedimiento iterativo se realiza hasta que se cumpla algún criterio de parada, que pueden ser: cantidad máxima de iteraciones, cantidad de iteraciones consecutivas sin obtener una mejora o una combinación de ambas. Una vez cumplido el criterio de parada el algoritmo procede a aplicar la propiedad explicada en la sección 3.2 que intenta optimizar la mejor solución hallada en S^*. Si las condiciones de la propiedad se cumplen (es decir, existen dos rutas que comparten al menos dos nodos en común) se procede a repartir las cargas de la manera indicada para generar una nueva y mejor solución S'^*.

5.2. Construcción y modificación de rutas

Tal como dijimos en la sección 3, decimos que una ruta es válida si comienza y termina en el depósito, respeta los horarios de llegada a cada cliente (inluyendo el regreso al depósito) y respeta la carga del vehículo.

En nuestro algoritmo las rutas son válidas por construcción. Es decir, toda ruta que se manipula en el procedimiento cumple las condiciones de carga y horarios de llegada al crearse y al modificarse en cada paso. Por lo tanto, nuestro algoritmo no permite soluciones intermedias inválidas (es decir, que incluyan al menos una ruta inválida) como sí lo hacen otras propuestas algorítmicas.

Una nueva ruta R_k se crea con la forma $R_k = (0, 0)$ indicándose de este modo que empieza y termina en el depósito. Claramente el depósito no tiene una demanda asociada y el horario de inicio y fin de ruta son los mismos que los de apertura y fin del depósito, así que la condiciones de carga y horarios de una ruta inicial se cumplen trivialmente. Eventualmente los horarios de inicio y fin de ruta serán modificados a medida que la ruta se va modificando. Luego, un cliente i se podrá agregar en la primera posición sólo en caso de que la ruta $R_k = (0, i, 0)$ siga cumpliendo las condiciones de carga y horarios, que en este caso serán: el abastecimiento de i no supera la carga del vehículo, el horario de salida del depósito cumple la venta del depósito (trivial en todos los casos), el horario de llegada a i cumple la ventana de i y el retorno al depósito cumple la ventana del depósito (no trivial pues debe adecuarse a los valores de i). Si alguna de estas condiciones no se cumple entonces el cliente no será agregado.

Con este esquema, cada vez que un cliente se inserta en la posición j se debe recalcular la carga disponible del vehículo y los horarios de llegada a cada cliente posterior a j (incluyendo el retorno al depósito) para poder determinar la factibilidad de la inserción. Es decir, la factibilidad de inserción de un cliente no depende sólo del cliente en cuestión sino también de los clientes ya incluidos en la ruta en posiciones posteriores. Este mismo procedimiento también se realiza al borrarse un cliente para mantener actualizadas la carga disponible del vehículo y los horarios de llegada a los clientes restantes.

Si $R_k = (0, ..., i, j, ..., 0)$, el cálculo del inicio del abastecimiento del cliente j, representado por θ_j, viajando directamente desde el cliente i es el siguiente:

$$\theta_j = \theta_i + t_{ij} + max\{a_j - \theta_i - t_{ij}, 0\}$$

Esto significa que el horario de inicio del abastecidimiento de j será la suma entre el horario de inicio del abastecimiento de i, el tiempo de viaje de i a j y un posible tiempo de espera según el horario de apertura de j. En caso de que el vehículo llegue antes del horario de apertura de j deberá esperar precisamente $a_j - \theta_i - t_{ij}$. Una manera equivalente de definir el inicio del

abastecimiento es la siguiente:

$$\theta_j = \begin{cases} \theta_i + t_{ij} & \text{si } a_j \leq \theta_i + t_{ij} \\ a_j & \text{sino} \end{cases}$$

Una vez calculado θ_j se verifica además la condición $\theta_j \leq b_j$. Si esta condición no se cumple la inserción se prohíbe, y en caso de cumplirse se actualizan y verifican las condiciones de cada cliente posterior en la ruta. Si alguna de estas condiciones deja de cumplirse la inserción también se prohíbe.

5.3. Solución Inicial

La solución inicial consiste en generar rutas válidas con posibles entregas parciales que satisfagan las demandas de todos los clientes. La construcción de las rutas se realiza con la siguiente idea: dada una nueva ruta R_k que empieza y termina en el depósito $R_k = (0, 0)$ determinamos en cada paso cuál es el cliente con demanda pendiente más cercano al último cliente agregado para agregarlo a continuación, si es que es factible. La inserción de cada cliente se realiza con el procedimiento antes explicado verificándose en cada caso las condiciones de validez de rutas. En caso de que sea factible agregar un cliente desde el punto de vista de tiempos pero la capacidad disponible del vehículo sea menor a la demanda del cliente, se procede a abastecerlo con la carga restante disponible y se deja que otro vehículo (eventualmente más de uno) también lo abastezca hasta satisfacer su demanda completa. De esta manera la solución inicial puede contener entregas parciales que no necesariamente serán convenientes. Las decisiones de conveniencia de las entregas parciales se realizarán más adelante durante el desarrollo de las operaciones del algoritmo.

El algoritmo general de la solución inicial es el siguiente:

Algorithm 3: Solución Inicial

Setear $k = 1$
repeat
  Crear una nueva ruta R_k que empiece y termine en el depósito $R_k = (0, 0)$
  Setear $j = 0$ y $\theta_0 = a_0$
  repeat
    $i = j$
    Hallar el cliente con demanda pendiente j más cercano a i que cumpla las condiciones de tiempo de la ruta
    Agregar j en R_k en la ante última posición
    $\theta_j = \theta_i + t_{ij} + max\{a_j - \theta_i - t_{ij}, 0\}$

  until *Hasta que* $u_k = 0$ *o no exista* j ;
  Setear $k = k + 1$

until *Hasta que todos los clientes sean abastecidos completamente* ;

El algoritmo genera una solución inicial válida donde la cantidad de vehículos puede exceder la cantidad disponible enunciada en el problema. La ejecución futura de la heurística disminuirá eventualmente la cantidad de vehículos en la búsqueda de soluciones de menor costo. Si al finalizar el procedimiento la cantidad de vehículos de la mejor solución encontrada sigue excediendo la cantidad disponible en la instancia particular el programa indicará que no se pudo hallar una solución que respete la disponibilidad vehicular.

5.4. Generación de vecindarios

Una vez generada la solución inicial se procede a ejecutar el algoritmo iterativo basado en Tabu Search. El procedimiento general intenta en cada paso mejorar la solución actual según el costo visitando soluciones vecinas definidas por un conjunto de operaciones. En nuestro caso implementamos seis operaciones de diferentes características y propósitos. Las primeras cuatro operaciones son las que utilizan Ho y Haugland en [13]; estas operaciones son muy utilizadas en el desarrollo de heurísticas para VRP por su simplicidad y suelen aparecen en la literatura con diferentes nombres y variantes. Además, decidimos implementar dos nuevas operaciones (un poco más complejas) pensadas con propósitos específicos a partir de la evaluación de nuestro algoritmo con las primeras cuatro.

A continuación explicaremos en detalle cada operación implementada y su aplicación para generar nuevas soluciones en el ámbito del vecindario. Utilizaremos la notación R'_k para referirnos al estado de la ruta R_k luego de aplicarse una operación. La condición *"Se puede agregar"* que aparece en los algoritmos se refiere a la posibilidad de insertar un cliente en una ruta desde el punto de vista de horarios y carga disponible del vehículo.

5.4.1. Operación 1 - Relocate

Sean R_k y R_l dos rutas distintas. Dados los clientes $i \in R_k \cap C$ y $j \in R_l$, el objetivo es pasar el cliente i en R_k a continuación de j en R_l. Los nuevos vehículos serán $R'_k = (0, ..., i-1, i+1, ..., 0)$ y $R'_l = (0, ..., j, i, j+1, ..., 0)$. Si el cliente i se encuentra en ambas rutas, es decir $i \in R_k \cap R_k \cap C$, y R_l tiene una carga disponible suficiente, entonces puede eliminarse una entrega parcial y las rutas quedarán $R'_k = (0, ..., i-1, i+1, ..., 0)$ y $R'_l = R_l$. Ver Figura 10.

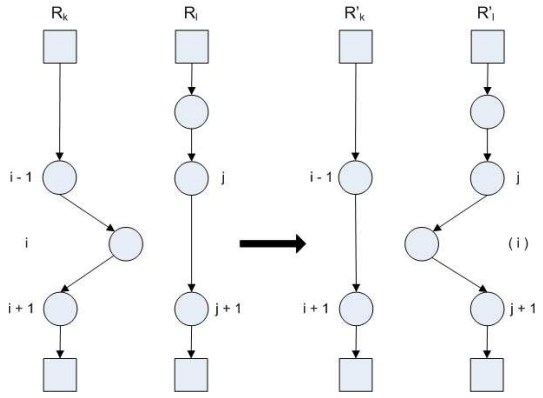


Figura 10: Operación Relocate

Una característica particular de esta operación es la de poder considerar R_k igual a R_l permitiendo la reubicación de los clientes en una misma ruta.

Durante las etapas de pruebas de esta operación notamos que a medida que avanzan las iteraciones del algoritmo algunos vehículos se van quedando sin clientes. La existencia de rutas vacías implica la generación de movimientos futuros de clientes a rutas vacías. Este hecho es interesante porque puede pensarse como una nueva alternativa en donde se agrega un nuevo vehículo

para generar nuevas soluciones en el espacio de búsqueda. Más aún, en algunos casos puntuales obtuvimos mejoras en el transcurso de las iteraciones a partir de esta situación. El problema surge en casos donde existan más de una ruta vacía ya que el movimiento de un cliente a cualquiera de estas rutas vacías produce el mismo resultado. Por esta razón, decidimos considerar en cada iteración solamente la primer ruta vacía para realizar los movimientos ignorando el resto de las rutas vacías. De esta forma logramos que la lista tabú no se llene de movimientos equivalentes innecesarios y evitamos la aparición de ciclos en casos con varios vehículos vacíos.

5.4.2. Operación 2 - 2-opt*

Esta operación intercambia los finales de dos rutas R_k y R_l. Dados los clientes $i \in R_k \cap C$ y $j \in R_l \cap C$, se generan dos nuevos vehículos $R'_k = (0, .., i, j+1, j+2, ..., 0)$ y $R'_l = (0, ..., j, i+1, i+2, ..., 0)$. En casos donde $i \in R_k \cap R_j \cap C$ y el cliente i coincide con $j+1$, una entrega parcial entre R_k y R_l puede ser eliminada. Ver Figura 11.

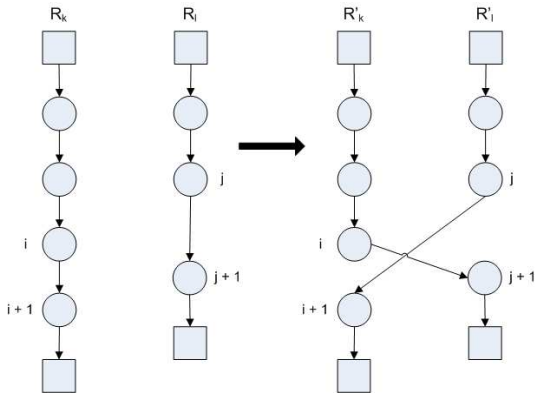


Figura 11: Operación 2-opt*

5.4.3. Operación 3 - Relocate Split

Esta operación reemplaza un split por otro, es decir, reemplaza un cliente i compartido por dos vehículos distintos R_k y R_l por un nuevo cliente j inicialmente sólo presente en R_l. Dados los clientes $i \in R_k \cap R_l \cap C$ y $j \in R_l \cap C$, el objetivo es quitar al cliente i de R_k y abastecerlo sólo con R_l, es decir

$R'_l = R_l$ conteniendo los mismos clientes pero con un abastecimiento mayor a i, y compensar esta nueva demanda moviendo parte del abastecimiento de R_l a j hacia R_k, es decir $R'_k = (0, ..., i-1, i+1, ..., 0) \cup \{j\}$. Ver Figura 12.

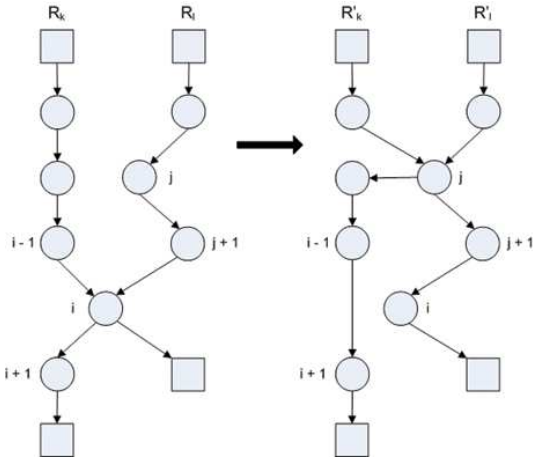


Figura 12: Operación Relocate Split

5.4.4. Operación 4 - Exchange

Dadas dos rutas distintas R_k y R_l el objetivo es intercambiar los clientes $i \in R_k \cap C$ y $j \in R_l \cap C$. El cliente i es insertado en $R_l \backslash \{j\}$ pero no necesariamente en la posición que tenía del cliente j. En forma similar, el cliente j es insertado en alguna posición de $R_k \backslash \{i\}$. En los casos donde $i \in R_k \cap R_j \cap C, i \neq j$ se puede eliminar una entrega parcial. Ver Figura 13.

5.4.5. Operación 5 - Split Client

Esta operación intenta realizar un split de un cliente i que se encuentra abastecido por la ruta R_k distribuyéndolo en un conjunto de otras rutas $D = \{R_v\} \subseteq V \backslash \{R_k\}$ que posean carga disponible y en conjunto logren abastecer completamente a i. Luego $R'_k = R_k \backslash \{i\} y R'_v = R_v \cup \{i\} \forall R_v \in D$. Ver Algoritmo 4 y Figura 14.

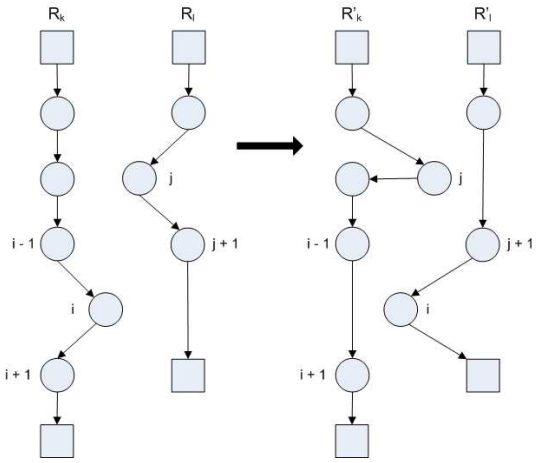


Figura 13: Operación Exchange

Algorithm 4: Operación 5 - Split Client

```
forall Ruta R_k do
    forall Cliente R_k[i] do
        CargaRestante ← Demanda de R_k[i]
        for j ← 1 to Cantidad de Rutas do
            CargaAAgregar ← máx{CargaRestante; Carga Disponible de R_j}
            if Se puede agregar R_k[i] en R_j con carga = CargaAAgregar then
                Agregar R_k[i] en R_j con carga = CargaAAgregar
                CargaRestante ← CargaRestante - CargaAAgregar
                if CargaRestante = 0 then
                    Eliminar R_k[i] de R_k
                end
            end
        end
    end
end
```

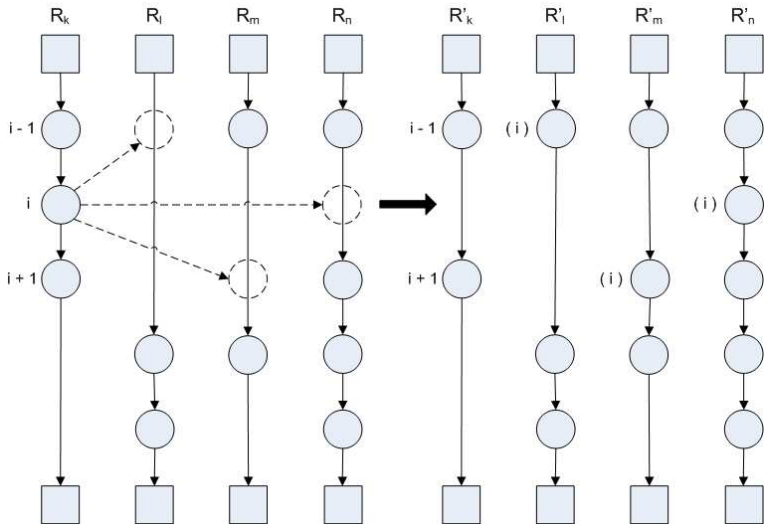


Figura 14: Operación Split Client

5.4.6. Operación 6 - Merge

Sean R_k y R_l dos rutas distintas. El objetivo de la operación es pasar la mayor cantidad posible de clientes de R_l a R_k en una única iteración del algoritmo. Esta operación puede pensarse como si se aplicaran varias operaciones Relocate en simultáneo. Si $I = \{c_j\} \subset R_l, I \neq \emptyset$ es el conjunto de clientes en R_l a mover, luego $R'_k = R_k \cup I$ y $R'_l = R_l \backslash I$. Ver Algoritmo 5 y Figura 15.

Algorithm 5: Operación 6 - Merge

```
forall par de rutas R_k y R_l do
    for i ← 1 to Cantidad de Clientes de R_l do
        for j ← 1 to Cantidad de Clientes de R_k do
            if Se puede agregar R_l[i] en R_k[j] then
                Agregar R_l[i] en R_k[j]
                Eliminar R_l[i] de R_l
            end
        end
    end
end
```

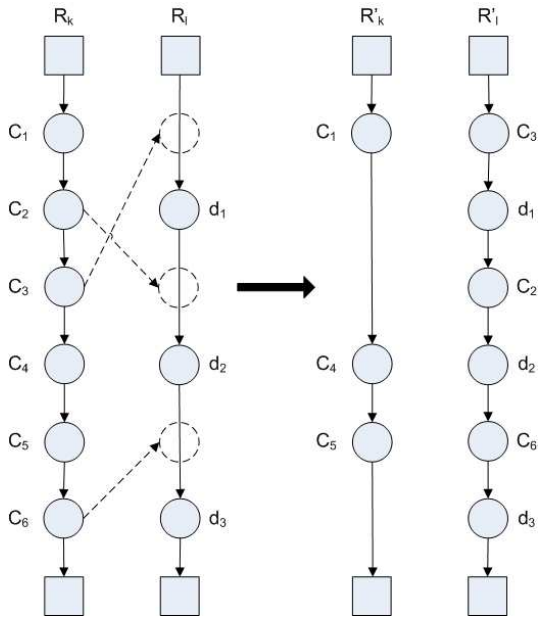


Figura 15: Operación Merge

5.5. Modos de ejecución

El ciclo principal de nuestro algoritmo puede ejecutarse de dos maneras: con o sin prioridades. El modo de ejecución es un parámetro configurable en el archivo inicial de parámetros. La ejecución sin prioridades es la metodología tradicional y consiste en ejecutar una única vez cada operación especificada sobre la solución actual y elegir la mejor opción entre todas ellas. Por otro lado, la ejecución con prioridades consiste en ejecutar la primer operación y verificar si con esta operación ya se consigue generar una solución vecina de menor costo. Si este es el caso, se elige esta solución como la actual y se ejecuta otra iteración del algoritmo ignorándose las operaciones restantes. Si el movimiento no logra obtener una nueva solución de menor costo se procede a ir ejecutando las operaciones en orden hasta encontrar una solución vecina de menor costo. Si ningún movimiento logra mejorar se elige entonces la solución vecina "que menos empeora". Este último caso también ocurre en la ejecución sin prioridades en caso de que ninguna operación logra mejorar. En las próximas secciones veremos estos modos de ejecución en detalle.

5.6. Detalles de Implementación

Como mencionamos antes, implementar un algoritmo Tabu Search requiere determinar e implementar sus características particulares y ajustar sus parámetros. Laguna en [14] presenta una guía de implementación de Tabu Search y describe alternativas de diseño, algunas de las cuales hemos considerado en nuestro trabajo. En esta subsección explicaremos los detalles de implementación de nuestro algoritmo general y de Tabu Search en particular.

5.6.1. Tenure

Un parámetro de vital importancia en la búsqueda tabú es el tenure. Algunas implementaciones utilizan un valor de tenure fijo a lo largo de la ejecución del algoritmo mientras que otros autores realizan adaptaciones dinámicas o selecciones al azar en un rango $[minTenure, maxTenure]$ (como Archetti, Speranza y Hertz en [2]). En nuestro caso utilizamos valores fijos determinados empíricamente, y en las instancias de 100 clientes nos basamos en el valor utilizado por Ho y Haugland [13].

5.6.2. Estructura de Tabu Search

Una de las decisiones más importantes a la hora de implementar un algoritmo Tabu Search es la elección de la estructura Tabu que será utilizada para llevar a cabo el historial de la exploración. Como dijimos antes, en la práctica es imposible almacenar el historial completo de soluciones visitadas ya que la cantidad de soluciones y el tamaño de mismas pueden ser arbitrariamente grandes dependiendo el tamaño del problema y la cantidad de iteraciones que se consideren, lo que llevaría a un overhead muy grande en las operaciones de inserción, actualización y comparación. Por este motivo optamos por almacenar atributos que definen las operaciones que generan dichas soluciones. En la mayoría de los casos no existe una única manera de definir atributos y la elección queda en manos de los investigadores, siendo algunas elecciones mejores que otras según diferentes punto de vista. En general sucede que las elecciones más restrictivas suelen ser más fáciles de implementar y menos costosas en términos de performance, aunque claramente una elección restrictiva implica un espacio de búsqueda más acotado que podría dejar afuera buenas soluciones.

A modo de ejemplo, supongamos que el algoritmo utiliza la operación de movimientos de nodos Relocate para generar soluciones vecinas. Es de-

cir, dado un conjunto de vehículos que conforman una solución factible, el vecindario de soluciones vecinas se construye a partir del movimiento de un cliente de una ruta a otra. Supongamos entonces que la mejor solución vecina se genera mediante el movimiento del cliente c del vehículo R_1 en posición p_1 al vehículo R_2 en posición v_2, como indica la figura 16.

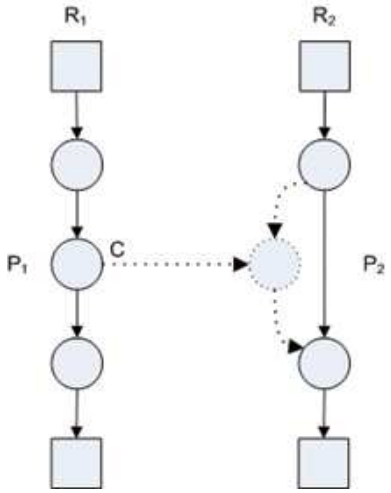


Figura 16: Movimiento de un cliente entre vehículos

En este caso el atributo trivial que define la generación de soluciones es el movimiento de clientes, y existen varias maneras de representar el movimiento en una estructura tabú que impida el movimiento inverso en exploraciones futuras, como por ejemplo:

a) prohibir que c vuelva a R_1 en la posición p_1 durante una cantidad *tenure* de iteraciones

b) prohibir que c vuelva a R_1 durante una cantidad *tenure* de iteraciones

c) prohibir el movimiento de c durante una cantidad *tenure* de iteraciones

La primer alternativa posee la mayor granularidad, o sea, es la menos restrictiva ya que permite explorar otras soluciones que involucren al cliente c, incluso dentro de R_1, pero es las más costosa ya que deben almacenarse más atributos por cada movimiento. En el otro extremo, la tercer alternativa

es altamente restrictiva pero es muy fácil de administrar y tiene muy bajo costo de almacenamiento.

En nuestro caso optamos por la opción intermedia e implementamos la lista tabú como una matriz $Tabu$ de $K \times N$, siendo K la cantidad de vehículos de la solución inicial y N la cantidad de clientes del problema. La matriz se inicializa en 0, es decir, $Tabu[k][c] = 0, \forall k \in V, \forall c \in C$, indicando que inicialmente todo movimiento es posible.

Siguiendo el ejemplo anterior, si se elige la solución vecina generada al moverse el cliente c del vehículo R_1 en p_1 al vehículo R_2 en p_2 entonces en la estructura Tabu marcamos $Tabu[R_1][c] = tenure$. De esta manera prohibimos que el cliente c vuelva a moverse al vehículo R_1 por $tenure$ iteraciones. En nuestro caso las posiciones p_1 y p_2 son irrelevantes y no las tenemos en cuenta.

Luego, al explorar el vecindario de soluciones vecinas verificamos la posibilidad de movimiento examinando el valor en la lista Tabu para cada cliente y para cada vehículo destino. Si el valor de $Tabu[vehiculoDestino][c]$ es 0 el movimiento es permitido, mientras que si el valor es mayor a 0 se lo prohibe ya que es considerado como un movimiento tabú. Sin embargo, recordamos que de cumplirse el criterio de aspiración adoptado la restricción es ignorada y el movimiento es aceptado, volviéndose a marcar como tabú nuevamente por $tenure$ iteraciones.

Para mantener actualizadas las restricciones de la lista tabú, en cada iteración del algoritmo se descuenta en uno las restricciones ya almacenadas, es decir $Tabu[i][j] = Tabu[i][j] - 1, \forall i, j$ tal que $Tabu[i][j] > 0$. De esta manera, al marcarse un movimiento como tabú su estado vuelve a permitirse luego de $tenure$ iteraciones ya que su valor tabú vuelve a ser 0.

Para simplificar la notación en las secciones siguientes, resumimos los nombres y las especificaciones de los métodos asociados a la lista tabú:

1. $inicializarTabu()$: $Tabu[k][c] = 0, \forall k \in V, c \in C$
2. $marcarTabu(k, c)$: $Tabu[k][c] = tenure$
3. $esTabu?(k, c)$: return $(Tabu[k][c] > 0)$
4. $descontarTabu()$: $Tabu[k][c] = Tabu[k][c] - 1, \forall k \in V, \forall c \in C$ tal que $Tabu[k][c] > 0$

5.6.3. Criterios de movimientos tabú

Aunque en la sección anterior presentamos un ejemplo de uso tabú relacionado con la operación Relocate, la estructura tabú que elegimos es la misma para todas las operaciones. Es decir, las seis operaciones que implementamos trabajan sobre la misma lista tabú y cada una utiliza de manera diferente los métodos $marcarTabu$ y $esTabu?$ explicados previamente de acuerdo a los movimientos que efectúan. A continuación presentamos una descripción de los criterios utilizados para marcar e identificar movimientos tabú por cada operación implementada.

Operación 1 - Relocate: Cuando se realiza un movimiento de esta operación se verifica que el cliente a pasar no haya estado en el vehículo destino. Si se quiere pasar el cliente i del vehículo R_k al vehículo R_l se verifica que no se cumpla $esTabu?(R_l, i)$. En caso de no ser tabú se efectúa el movimiento y se marca tabú el movimiento del cliente i al vehículo R_k, es decir $marcarTabu(R_k, i)$.

Operación 2 - 2-opt*: Esta operación intercambia los finales de dos vehículos. Si se quieren intercambiar los finales de los vehículos R_k y R_l a partir de los clientes i y j respectivamente, se verifica que no se cumplan $esTabu?(R_l, i+1)$ & $esTabu?(R_k, j+1)$. Al efectuarse el movimiento se marcan como tabú $marcarTabu(R_k, i+1)$ y $marcarTabu(R_l, j+1)$.

Operación 3 - Relocate Split: Dados dos vehículos R_k y R_l que comparten un cliente i en común, la operación intenta reubicar el split en otro cliente, es decir, elimiar el split en i y generar un nuevo split en otro cliente $j \neq i$. Las verificaciones y marcas de movimientos tabú de esta operación son los mismos que se utilizan en la operación Relocate.

Operación 4 - Exchange: Esta operación intercambia un cliente de ambas rutas. Desde el punto de vista de factibilidad de movimiento, se puede ver esta operación como una doble aplicación de la operación Relocate. Si la operación Exchange intercambia el cliente i del vehículo R_k con el cliente j del vehículo R_l, la condición necesaria para que se pueda efectuar el movimiento es que no se cumplan $esTabu?(R_l, i)$ & $esTabu?(R_k, j)$. Si el movimiento es factible se marcan como tabú $marcarTabu(R_k, i)$ y $marcarTabu(R_l, j)$.

Operación 5 - Client Split: El objetivo de esta operación es realizar un *split* de un cliente en varios vehículos dstinos. La condición que debe cumplirse es que ningún movimiento del cliente a cada vehículo destino sea tabú. Si se quiere dividir el cliente i del vehículo R_k en los vehículos B_1, B_2,..., B_v se verifica que $\forall B_j$ no se cumpla $esTabu?(B_j, i)$ con $j \in [1, v]$. Luego de realizarse el movimiento se marca como tabú $marcarTabu(R_k, i)$.

Operación 6 - Merge: Esta operación intenta pasar la mayor cantidad posible de clientes de una ruta R_k a otra ruta destino R_l. La condición que debe cumplirse es que todo movimiento de cada cliente involucrado al vehículo destino sea no tabú. Si la operación consiste en mover los clientes c_1, c_2,...,c_n del vehículo R_l al vehículo R_k, se verifica que $\forall c_j$ no se cumpla $esTabu?(R_l, c_j)$ con $j \in [1, n]$. Luego de realizarse el movimiento, $\forall c_j$ se marcan como tabú $marcarTabu(R_k, c_j)$ con $j \in [1, n]$.

5.6.4. Atributos de los movimientos

Debido a que las operaciones tienen diferentes complejidades y alcances, cada movimiento fue implementado de manera específica según sus atributos particulares. A continuación presentamos la lista de atributos que utilizamos para identificar movimientos de cada tipo de operación.

Operación 1 - Relocate:

- id del vehículo origen R_k
- id del vehículo destino R_l
- cliente i a mover de R_k a R_l
- posición de i en R_l

Operación 2 - 2-opt*:

- id del primer vehículo R_k
- id del segundo vehículo R_l
- cliente i en R_k a partir del cual se intercambia la ruta
- cliente j en R_l a partir del cual se intercambia la ruta

Operación 3 - Relocate Split:

- id del primer vehículo R_k

- id del segundo vehículo R_l
- cliente i común en ambos vehículos R_k y R_l que será abastecido sólo por R_l
- cliente j en R_l que será abastecido además por R_k
- posición p en R_k donde se agregará j
- cantidad de carga "pivote"t entre los clientes

Operación 4 - Exchange:

- id del primer vehículo R_k
- id del segundo vehículo R_l
- cliente i a mover de R_k a R_l
- cliente j a mover de R_l a R_k
- posición p_l en R_l donde se moverá i
- posición p_k en R_k donde se moverá j

Operación 5 - Client Split:

- id del vehículo origen R_k
- cliente i en R_k a dividir en varios vehículos $D = B_j \subseteq V \backslash \{R_k\}, j \neq k$
- lista ordenada $L =< B_j >, j \neq k$ de vehículos destinos que incluirán a i
- lista ordenada $P =< p_j >$ de las posiciones en que se incluirá i en cada vehículo destino. Es decir, $\forall j$ el cliente i se pasa a B_j en la posición p_j.

Operación 6 - Merge:

- id del vehículo origen R_k
- id del vehículo destino R_l
- lista ordenada $C =< c_j >$ de clientes en R_k que pasarán a R_l

5.6.5. Vecindarios, pilas y modos de ejecución

En esta subsección presentaremos los detalles de implementación y de evaluación de vecindarios utilizados por el algoritmo general de Tabu Search. Además veremos en detalle los mecanismos de selección de próximas soluciones presentes en las dos variantes de ejecución de nuestro algoritmo

Como mencionamos anteriormente, dada una operación *op* en particular, la exploración del vecindario consiste en ir generando soluciones vecinas aplicando a la solución actual S_i toda combinación posible de los movimientos definidos por *op*. Los atributos de cada movimiento han sido detallado en la sección anterior. Luego se evalúa la factibilidad de cada solución vecina S' generada y en caso de ser válida se calcula su costo $z(S')$ y lo que denominamos *valor de mejora* definido como $z(S') - z(S_i)$. El valor de mejora es negativo si S' es mejor que S_i, positivo en caso contrario o 0 si tienen el mismo costo. Cada solución vecina S' queda entonces determinada por los valores particulares de los atributos del movimiento que la ha generado. Estos valores junto al valor de mejora resultante se almacenan en una *pila* de soluciones ordenada por el valor de mejora. Concluyendo, la pila $pila_{op}$ representa el vecindario $N_{op}(S_i)$ de soluciones válidas de la solución actual S_i de acuerdo a la operación *op*.

Optamos por implementar los vecindarios mediante pilas para simplificar el proceso de generación y selección de soluciones. La pila de la operación *op*, $pila_{op}$, se inicializa vacía y cuando se agrega una solución se utiliza el método $pila_{op}.push(< atributos\_del\_movimiento, valor\_de\_mejora >)$ para almacenar el movimiento generador y el valor de mejora. La pila contiene las soluciones ordenadas en todo momento por el valor de mejora en orden ascendente, con lo cual al realizarse una operación $m_{op} = pila_{op}.pop$ se obtiene el movimiento que al aplicarse sobre S_i genera la mejor solución factible vecina a S_i. Puede ocurrir que todas las soluciones en la pila tengan un valor de mejora negativo (lo que significa que toda solución vecina es mejor que S_i), que todas tengan un valor positivo (indicando que no hay soluciones vecinas mejores, con lo cual se debe introducir una solución que no mejora la secuencia de búsqueda) o que haya valores de distinto signo e incluso en cero.

Los procesos de construcción y actualización de pilas y de selección de la próxima solución dependen exclusivamente del modo de ejecución del algoritmo, que puede ser con o sin prioridades. En el caso sin prioridades se construye una pila por cada operación, y luego se elige el mejor movimien-

to entre los mejores movimientos de cada pila. Es decir, se toma el mínimo entre los primeros movimientos de cada pila obtenibles mediante el *pop* de cada pila. Si este movimiento no es tabú o es tabú pero cumple el criterio de aspiración entonces se lo utiliza para generar la próxima solución S_{i+1}. Si en cambio es tabú y no cumple el criterio de aspiración, entonces se vuelve a realizar un *pop* sobre esa misma pila para obtener un segundo candidato de esa operación. El siguiente algoritmo ilustra el esquema general de la ejecución sin prioridades.

Algorithm 6: Ejecución sin prioridades

Sean S_i la solución actual y G el grafo de vecindarios,
for *cada operación* $op \in \{op_1, ..., op_v\}$ *especificada* **do**
    $pila_{op} = GenerarVecindario(S_i, op, G)$
end
for *cada operación* $op \in \{op_1, ..., op_v\}$ *especificada* **do**
    $mov_{op} = pila_{op}.pop()$
end
Setear $hallada = false$
while *not(hallado)* **do**
    Buscar el movmiento $mov_{op}\prime$ entre $\{mov_{op}\}$ que tiene el menor valor de mejora asociado. Es decir,
    $mov_{op}\prime = argmin\{< mov_{op}, valorMejora > \| mov_{op} \in \{mov_{op}\}\}$
    if $mov_{op}\prime$ *no es tabú o es tabú pero cumple el criterio de aspiración* **then**
        $mov_{op}* = mov_{op}\prime$
        Setear $hallado$ = true
    else
        Setear $mov_{op} = pila_{op}.pop()$
    end
end
Generar una nueva solución $S_{i+1} = AplicarMovimiento(m_{op}*, S_i)$

La ejecución con prioridades modifica levemente las condiciones de creación de pilas y selección de movimientos. El objetivo de este modo de ejecución es evitar la ejecución completa de todas las operaciones si es que se consigue una solución mejor con menos operaciones. Para esto es necesario establecer un orden prioritario entre las operaciones.

Si $op = [op_1, ..., op_n]$ es la lista ordenada de operaciones a ejecutar, el método obtiene la primer pila $pila_{op[1]}$ como consecuencia de la generación del vecindario de S_i mediante la primer operación $op[1]$. Luego busca un

movimiento de esta primera operación que tenga un valor de mejora negativo (es decir, que logre mejorar la solución actual) y que no sea tabú o lo sea pero cumpla el criterio de aspiración. La búsqueda se realiza aplicando $pila_{op[1]}.pop$ consecutivos y verificando las condiciones tabúes. Si el movimiento existe, se genera la nueva (y mejor) solución y se termina la iteración actual ignorándose el resto de las operaciones. De esta manera se reduce drásticamente el tiempo de ejecución de la iteración pero con la posibilidad de descartar mejores soluciones que habrían sido encontradas en caso de haberse ejecutado las operaciones restantes.

Algorithm 7: Ejecución con prioridades (Primera Parte)

Sean S_i la solución actual, G el grafo de vecindarios y $op = [op_1, ..., op_v]$ la lista ordenada de operaciones,
Setear $hallada = false$
Setear $i = 1$
// Esta primer parte busca un movimiento de mejora $mov_{op}*$ ejecutando
// las operaciones en orden sólo en caso de ser necesario.
while $i \leq v$ *y not(hallado)* **do**
    Setear $op = op[i]$
    $pila_{op} = GenerarVecindario(S_i, op, G)$
    Setear $sigoEnOperacion = true$
    while *sigoEnOperacion y not(hallado)* **do**
        $mov_{op} = pila_{op}.pop()$
        if mov_{op} *no es tabú o es tabú pero cumple el criterio de aspiración y tiene un valor de mejora negativo* **then**
            $mov_{op}* = mov_{op}$
            Setear $hallado = true$
        else
            if mov_{op} *tiene un valor de mejora positivo* **then**
                Setear $sigoEnOperacion = false$
            else
                Setear $mov_{op} = pila_{op}.pop()$
            end
        end
    end
    Setear $i = i + 1$
end

Algorithm 8: Ejecución con prioridades (Segunda Parte)

// Esta segunda parte elige el movimiento "que menos empeora" $mov_{op}*$
// de forma similar al algoritmo anterior.
if *not(hallado)* **then**
    Setear $i = 1$
    for *cada operación* $op \in [op_1, ..., op_v]$ *especificada* **do**
        $mov_{op} = pila_{op}.pop()$
    end
    while *not(hallado)* **do**
        Buscar el movmiento $mov_{op}\prime$ entre $\{mov_{op}\}$ que tiene el menor valor de mejora asociado. Es decir,
        $mov_{op}\prime = argmin\{< mov_{op}, valorMejora > \| mov_{op} \in \{mov_{op}\}\}$
        if $mov_{op}\prime$ *no es tabú o es tabú pero cumple el criterio de aspiración* **then**
            $mov_{op}* = mov_{op}\prime$
            Setear $hallado = true$
        else
            Setear $mov_{op} = pila_{op}.pop()$
        end
    end
end
Generar una nueva solución $S_{i+1} = AplicarMovimiento(m_{op}*, S_i)$

Si por el contrario no se logra encontrar un movimiento en $pila_op[1]$ con un valor de mejora negativo se procede a realizar el mismo procedimiento sobre las operaciones siguientes respetando sus prioridades hasta encontrar un movimiento con valor de mejora negativo. Si aún así no se logra encontrar este movimiento, se llega a un estado en donde todas las pilas contienen movimientos que empeoran. Aplicando una idea similar, se busca entonces aquel movimiento no tabú "que menos empeore"la solución actual. El algoritmo ilustrado en 7 y 8 muestra el esquema general de la ejecución con prioridades.

6. Variantes de Tabu Search Granular

En esta sección presentaremos las variantes de Tabu Search Granular implementadas. Cada una de estas variantes fueron utilizadas en las pruebas realizadas. Recordamos que el objetivo de esta técnica es restringir el vecindario de cada cliente, filtrando la condición de vecindad entre vecinos mediante algún criterio. Para esto implementamos una matriz binaria de $N \times N$ llamada *vecindario* donde:

$$vecindario(i,j) = \begin{cases} 1 \text{ si los clientes } i \text{ y } j \text{ son vecinos} \\ 0 \text{ si no} \end{cases}$$

y para abstraer la estructura de datos utilizados implementamos el método $esVecino(i,j)$ que retorna un valor booleano indicando si ambos clientes pasados como parámetros son vecinos.

6.1. Tabu Granular por Distancia (GTSxD)

Toth y Vigo en [19] proponen una fórmula para definir los arcos "largos". La fórmula propuesta es $\overline{z}^* = \frac{z^*}{(n+K)}$ donde z^* es el valor de la mejor solución conocida y $\overline{z}^*$ representa una estimación de la longitud promedio de los arcos de la mejor solución. Al no contar con los valores de la mejor solución, los autores proponen el uso del algoritmo Savings de Clarke y Wright (1964) para obtener una buena solución rápida en la cual basarse. En nuestro caso, al estar utilizando instancias de prueba conocidas, contamos con los mejores resultados obtenidos para aplicarlos en la fórmula. En los casos donde no se cuenta con los valores de una buena solución conocida nuestro algoritmo utiliza el valor de la solución inicial como mejor solución, evitando el costo computacional del algoritmo de Savings ($O(n^3)$).

Si bien estos valores son distintos de los mejores, se pueden encontrar buenos resultados utilizando la fórmula en [19] $\nu = \beta\frac{z'}{(n+K)}$ donde z' es el costo de la solución inicial y K la cantidad de vehículos utilizados por la solución inicial. En la sección 8 presentaremos los valores de β con que realizamos las pruebas y con los que obtuvimos mejores resultados, así como los resultados de evaluar el algoritmo sin conocer la mejor solución a cada instancia.

6.2. Tabu Granular por Cantidad (GTSxC)

El filtro propuesto en la subsección anterior cubre todos los arcos cuyas distancias sean menores a ν; en otras palabras, si estamos situados en un cliente i los clientes que podrán ser alzanzados desde i son los que se se encuentran dentro del un círculo de radio ν centrado en i. En los casos donde un cliente tiene pocos vecinos dentro del círculo, por ejemplo en el caso de clientes *outliers*, la variante GTSxD limita demasiado la búsqueda.

Por esta razón proponemos un enfoque de filtro alternativo por cantidad de clientes en lugar de distancia. Para cada cliente se considera que otro cliente es vecino del primero si está dentro de los C más cercanos sin importar la distancia que se encuentre. Esta forma de determinar si un cliente es vecino de otro permite también contar con precisión la cantidad de ejes utilizados en el Tabu, que a diferencia del método anterior se estimaba una cantidad aproximada.

6.3. Intensificación y Diversificación

Como se mencionó en la seccion anterior, para GTSxD la intensificación y diversificación se pueden lograr fácilmente mediante el cambio del valor de β en tiempo de ejecución. Para lograr este cambio, una de las variables que se setean desde el archivo de parámetros es la que denominamos *factor granular* Fg que se utiliza para modificar el valor de β. El radio de vecinos disminuye o aumenta cada 10 iteraciones consecutivas en que se mejora la mejor solución encontrada hasta el momento o no se logra mejorar dicha solución respectivamente. Si $z(S_i)$ es el valor de la solución encontrada por el algoritmo en la iteración i y $z(S*)$ es el valor de la mejor solución encontrada los criterios utilizados para diversificar e intensificar son los siguientes:

Para Diversificar:
si $z(S*) \leq z(S_{i+h})\ \forall h \in [0.,9]$ $\Rightarrow$ $\beta = \beta + Fg$ y recalcular ν

Para Intensificar:
si $z(S*) > z(S_i) > z(S_{i+1}) ... > z(S_{i+9})$ $\Rightarrow$ $\beta = \beta - Fg$ y recalcular ν

Para GTSxC, el criterio de intensificación y diversificación utilizado también es simple. Lo que hacemos es disminuir o aumentar la cantidad de vecinos

C según el factor Fg siguiendo el mismo criterio anterior de tener 10 iteraciones seguidas que mejoran la mejor solución actual o no logran mejorar la dicha solución. Es decir:

Para Diversificar:
si $z(S*) \leq z(S_{i+h})\ \forall h \in [0.,9]$ $\Rightarrow$ $C = C + Fg$ y recalcular ν

Para Intensificar:
si $z(S*) > z(S_i) > z(S_{i+1}) ... > z(S_{i+9})$ $\Rightarrow$ $C = C - Fg$ y recalcular ν

7. Implementación

7.1. Archivo Inicial

Al comenzar, el programa lee el archivo *archivoInicial.txt* que contiene las instancias del problema a ejecutar. En nuestro caso utilizamos las instancias de Solomon que serán explicadas en la sección 8.1. En una única ejecución del programa pueden ejecutar varias instancias en forma secuencial. El formato del archivo para cada instancia es:

- Nombre de la instancia a ejecutar
- Mejor valor conocido de la instancia
- Cantidad de vehículos de la mejor instancia

Los valores de mejor valor y cantidad de vehículos son utilizados por el módulo granular para calcular la distancia máxima entre clientes y de esta forma determinar si son o no vecinos. En los casos en que no se cuenta con un valor conocido, se puede poner el valor 0 (cero) y el algoritmo utiliza los valores de la solución inicial para realizar estos cálculos.

El formato de las instancias de Solomon está descripto en [20].

7.2. Parámetros de ejecución

El archivo *achivoParametros.txt* contiene parámetros de ejecución que pueden ser seteados en cada simulación. Los parámetros y sus posibles valores son los siguientes:

Tenure: Contiene el tenure del Tabu Search, es decir, la cantidad de iteraciones por las cuales un movimiento se considera tabu.

Cantidad de iteraciones: Indica la cantidad máxima de iteraciones del ciclo principal del algoritmo.

Cantidad de iteraciones sin mejorar: Indica la cantidad máxima de iteraciones consecutivas que el ciclo principal puede ejecutar sin encontrar una solución mejor a la encontrada hasta el momento.

Porcentaje de mejora: Se utiliza como método de corte del algoritmo. Si pasaron "Cantidad de iteraciones sin mejora"donde no se mejoró el "Porcentaje de mejora'él algoritmo finaliza. Si el valor es 0 no se utiliza.

Tipo de Granular: Según el valor seteado indica si se utiliza GTS o no, y qué tipo de granular utiliza. Los posibles valores son:

0. indica que no se utiliza granular

1. indica que se utiliza granular por distancia (GTSxD)

2. indica que se utiliza granular por cantidad (GTSxC)

3. indica que se utiliza granular mixto (por distancia + por cantidad)

Tipo de Incremento: Indica si se utiliza diversificación e intensificación o no. Dependiendo del tipo de granular ingresado, el algoritmo aplica el tipo de incremento correspondiente. Los valores posibles son:

0. no se usa diversificación ni intensificación

1. usa diversificación

2. usa diversificación e intensificación

Valor Beta: Contiene el valor de beta (β) para el cálculo del valor máximo de las distancias entre los clientes que van a ser consideradas en el granular.

Factor Incremento Beta: El factor de incremento se suma al valor de β para calcular la distancia máxima considerada. Este factor se incrementa a β cada 10 iteraciones seguidas en las que no se logra mejorar la mejor solución encontrada hasta el momento.

Valor Cantidad: Contiene la cantidad de clientes que van a ser considerados como vecinos para cada cliente.

Factor Incremento Cantidad: El factor de incremento se suma al valor *cantidad* para determinar los cliente más cercanos. Este factor se incrementa a *cantidad* cada 10 iteraciones seguidas en las que no se logra mejorar la mejor solución encontrada hasta el momento.

Operaciones: Especifica las operaciones que se van a utilizar en la ejecución del algoritmo. Los valores de las operaciones pueden separarse por comas. Los posibles valores de las operaciones son:

1. Relocate

2. 2-opt*

3. Relocate Split

4. Relocate Exchange

5. Split Client

6. Merge

Con/Sin Prioridad: Indica si la ejecución se realiza con o sin prioridades. Los valores posibles son "si"y "no". En caso de ejecutarse con prioridades el orden de ejecución es el establecido en el punto anterior.

Salida por Pantalla: Los valores posibles son "si"y "no". En caso de estar habilitado el programa realiza un dump por pantalla que incluye las operaciones utilizadas en cada iteración.

Pausa: Los valores posibles son "si"y "no". Durante el desarrollo lo utilizamos para realizar pausas en determinados puntos de ejecución, como por ejemplo entre iteraciones consecutivas o entre instancias particulares cuando ejecutamos una serie de archivos de prueba.

Detalles: Los valores posibles son "si"y "no". En caso de estar habilitado el programa realiza un dump detallado en cada iteración de la solución actual incluyendo los detalles de cada vehículo, a qué clientes abastece y sus abastecimientos y horarios de llegada, entre otros.

7.3. Lenguaje y metodología

Para implementar el algoritmo decidimos utilizar el lenguaje C# de Microsoft en su versión 2.0. La elección se basó en la experiencia con la que contamos en este lenguaje y la facilidad del mismo para desarrollar sistemas orientados a objetos. El desarrollo de la tesis pasó por las etapas típicas de cualquier proyecto. En primer lugar realizamos el relevamiento de las necesidades reales del programa y en la segunda etapa realizamos el análisis y el diseño comenzando por diagramas más generales y concluyendo hacia los más particulares. Utilizamos diagramas de Objetos, de Clase y de Interacción para reflejar los distintos aspectos del sistema y poder contar con una idea detallada de la futura implementación.

7.4. Hardware y Sistema Operativo

Durante el desarrollo y las pruebas se utilizó una PC Toshiba Genuine Intel T2300 1.66GHz con 1 Giga de memoria RAM. El sistema operativo sobre el cual se trabajó fue Windows XP Professional.

8. Resultados Computacionales

En esta sección presentaremos los resultados obtenidos con nuestro algoritmo a partir de varias pruebas realizadas según los diferentes parámetros que consideramos. Esta sección está organizada de la siguiente manera: primero presentaremos las instancias de prueba que utilizamos y contra qué valores comparamos nuestros resultados. Luego presentaremos varias subsecciones con resultados clasificadas por el tamaño de las instancias de prueba. Dentro de cada subsección se encuentran los parámetros de prueba utilizados con sus resultados, los mejores resultados conocidos y en algunos casos resaltamos las instancias en las cuales obtuvimos un resultado mejor del publicado hasta el momento.

Los parámetros que evaluamos son el tipo de granular utilizado con los diferentes valores, la diversificación e intensificación de GTS y la incorporación de prioridades en las operaciones. El valor del tenure que utilizamos en las instancias de 100 clientes es el utilizado por Ho y Haugland en [13] y para el resto de las instancias utilizamos un valor proporcional al mismo que será presentado en cada subsección correspondiente según el tamaño de las instancias.

8.1. Instancias de Prueba

Para realizar las pruebas utilizamos los archivos de instancias propuestas por Solomon para el VRPTW disponibles en *The VRP Web* [17] y *Vehicle Routing and Travelling Salesperson Problems* [20] con diferentes cantidades de clientes: 25, 50, 100, 200, 400, 600, 800 y 1000 clientes. Las instancias están divididas en 6 grupos (conjuntos de pruebas) denotadas R1, R2, C1, C2, RC1 y RC2. Cada conjunto de pruebas contiene entre 8 y 12 instancias. En R1 y R2 la ubicación de los clientes fue generada en forma aleatoria con una distribución uniforme. En los conjuntos de prueba C1 y C2 los clientes fueron ubicados en clusters, y finalmente en los conjuntos RC1 y RC2 algunos clientes fueron ubicados en clusters mientras que otros fueron ubicados en forma aleatoria. La diferencia entre los conjuntos de prueba R1, C1 y RC1 y los conjuntos R2, C2 y RC2 radica en que los datos están preparados para que las rutas generadas por las instancias R1, C1 y RC1 contengan una cantidad menor de clientes que las rutas R2, C2 y RC2 porque las ventanas de tiempo son mucho más acotadas. Por ejemplo, en las soluciones de instancias de 100 clientes las rutas R1, C1 y RC1 contienen entre 5 y 10 clientes mientras que

las rutas R2, C2 y RC2 llegan a incluir más de 30 clientes.

En primer lugar comparamos nuestros resultados con los presentados por Ho y Haugland en [13] quienes testearon su algoritmo para el VRPTW en instancias de 100 clientes. Luego, al no contar con otros resultados de VRPTWSD, comparamos nuestros resultados con los mejores resultados de VRPTW disponibles en *The VRP Web* [17] para instancias de 25 y 50 clientes y *Vehicle Routing and Travelling Salesperson Problems* [20] para instancias de 200, 400, 600, 800 y 1000 clientes.

8.2. Instancias de 25 clientes

Como mencionamos al comienzo de la sección, en la primer etapa realizamos una evaluación de los parámetros a utilizar en la ejecución de las pruebas. Para realizar esta tarea seleccionamos arbitrariamente un subconjunto de instancias de prueba de cada conjunto y aplicamos diversas variantes para ajustar los parámetros. Las instancias de prueba que seleccionamos fueron: c101.txt, r101.txt, rc101.txt, c102.txt, r102.txt y rc102.txt.

La siguiente tabla presenta, para cada instancia de prueba, los valores de la cantidad de vehículos en la parte superior y la distancia recorrida en la parte inferior. Los parámetros que fueron fijados son:

- Tenure = 10
- Método de Corte = 50 iteraciones sin mejora

Parámetros	r101	r201	c101	c201	rc101	rc201
Con las operaciones 1, 2, 3 y 4	8 618.33	4 465.18	3 198.60	2 219.10	6 530.94	4 471.16
Con las operaciones 1, 2, 3, 4, 5, 6	8 623.52	4 465.91	3 198.60	2 219.10	5 477.93	3 362.86
GTSxD $\beta = 1$ Todas las op.	9 627.13	5 489.47	3 198.60	2 219.10	5 477.93	3 363.12
GTSxD $\beta = 1.5$ Todas las op.	8 618.33	4 474.16	3 198.60	2 219.10	5 477.93	3 363.12
GTSxD $\beta = 2$ Todas las op.	8 618.33	3 488.45	3 198.60	2 219.10	5 477.93	3 363.12
GTSxC $C = 5$ Todas las op.	9 627.13	4 468.51	3 198.60	2 219.10	6 530.69	5 468.63
GTSxC $C = 10$ Todas las op.	9 627.13	4 469.49	3 198.60	2 219.10	5 477.93	3 361.62
GTSxC $C = 15$ Todas las op.	8 618.33	4 465.91	3 198.60	2 219.10	5 477.93	3 361.62

Los resultados muestran que no hay mucha diferencia entre los distintos valores para cada parámetro. Notamos además que cuando el vecindario es muy chico hay casos donde no se llega a encontrar una buena solución.

Como mencionamos en la sección de variantes de granular, la variante por distancia (GTSxD) utiliza una fórmula que incluye el mejor valor conocido de la instancia en cuestión. Cuando no contábamos con dicho valor, utilizamos la solución inicial generada por el algoritmo para aproximar el valor de $\overline{z}^*$. Gracias a que las instancias de prueba son comúnmente utilizadas para evaluar estos problemas, utilizamos el valor conocido de la mejor solución. Con el objetivo de evaluar el comportamiento del algoritmo cuando no se cuente con este resultado, corrimos las instancias muestrales con los parámetros de granular GTSxD utilizando la solución inicial. Los resultados obtenidos son:

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	9	5	3	2	5	3
$\beta = 1$						
Todas las op.	627.13	479.79	198.60	219.10	477.93	363.12
GTSxD	8	5	3	2	5	3
$\beta = 1.5$						
Todas las op.	618.33	483.73	198.60	219.10	477.93	362.86
GTSxD	8	3	3	2	5	3
$\beta = 2$						
Todas las op.	618.33	485.99	198.60	219.10	477.93	362.86

Los resultados muestran que el uso de la solución inicial del algoritmo para aproximar el mejor valor conocido es conveniente ya que los valores no difieren demasiado.

Uno de los parámetros que introdujimos es el uso de prioridades entre las operaciones en el ciclo principal de ejecución. Para determinar las prioridades más convenientes entre las operaciones medimos en cada instancia y por cada operación: el porcentaje de tiempo de ejecución de la operación, el porcentaje de veces que la operación obtiene un mejor resultado que las restantes y el porcentaje de mejora obtenido. Los valores presentados a continuación se obtuvieron al ejecutar el algoritmo con todas las operaciones y sin granular.

Instancia r101	Op1	Op2	Op3	Op4	Op5	Op6
Tiempo Ejecución	21.62 %	8.11 %	0 %	33.78 %	12.16 %	24.32 %
Iteraciones Ganadas	70.15 %	13.43 %	0 %	4.48 %	0 %	11.94 %
Valores Mejora Negativos	59.56 %	5.11 %	0 %	2.4 %	0 %	32.93 %
Instancia r201	Op1	Op2	Op3	Op4	Op5	Op6
Tiempo Ejecución	12.03 %	10.7 %	0 %	62.57 %	4.55 %	10.16 %
Iteraciones Ganadas	79.14 %	10.79 %	0 %	6.47 %	0 %	3.60 %
Valores Mejora Negativos	63.83 %	13.7 %	0 %	12.29 %	0 %	10.18 %
Instancia c101	Op1	Op2	Op3	Op4	Op5	Op6
Tiempo Ejecución	14 %	12 %	0 %	50 %	12 %	12 %
Iteraciones Ganadas	54.39 %	14.04 %	0 %	14.04 %	1.75 %	15.79 %
Valores Mejora Negativos	23.11 %	12.07 %	0 %	15.55 %	0.71 %	48.56 %
Instancia c201	Op1	Op2	Op3	Op4	Op5	Op6
Tiempo Ejecución	10.71 %	7.65 %	0 %	67.35 %	4.08 %	10.2 %
Iteraciones Ganadas	53.33 %	18.33 %	0 %	21.67 %	1.75 %	6.67 %
Valores Mejora Negativos	56.86 %	0.23 %	0 %	14.77 %	0.53 %	28.15 %

En estos resultados podemos notar que:

- la operación 1 es la que obtiene valores mejores en la mayor cantidad de iteraciones
- en proporción a las iteraciones ganadas, la operación 6 es la que obtiene los mejores valores de mejora
- la operación 3 no obtiene buenos valores en ningún caso. Esto se debe a que en ningún momento se cumplió la precondición para que la operación pudiera ejecutarse
- la operación 4 ayuda a encontrar buenos resultados pero el tiempo de ejecución es muy superior al resto

Con estas conclusiones proponemos entonces que las prioridades de las operaciones siga la siguiente secuencia: 1,6,2,4,5. En la siguiente tabla presentamos algunos resultados obtenidos al ejecutar las instancias de Solomon con la secuencia de prioridades mencionada.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	8	4	3	3	5	3
$\beta = 1.5$						
Todas las op.	618.33	476.41	198.60	238.31	477.93	362.65
GTSxC	8	4	3	2	5	3
$C = 15$						
Todas las op.	618.33	468.68	198.60	219.10	477.93	362.65

Los valores obtenidos casi no difieren comparándolos con los obtenidos sin utilizar prioridades. Lo que sí cambia es el tiempo de ejecución al ejecutarse menos operaciones en cada iteración. Esta conclusión se hará más notoria en instancias de mayor cantidad de clientes ya que los tiempos de ejecución de instancias de 25 clientes son pequeños.

Finalmente el último parámetro que queda por evaluar en instancias de este tamaño es el uso de diversificación e intensificación. En la siguiente tabla presentamos los resultados obtenidos al evaluar dichos parámetros para GTSxD con β=1.5 y GTSxC con C=15, que hasta el momento son los valores que mejores resultados arrojaron.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	8	4	3	3	5	3
$\beta = 1.5$						
Todas las op.	618.33	468.51	198.60	219.10	477.93	363.12
GTSxC	8	3	3	2	5	3
$C = 15$						
Todas las op.	618.33	485.99	198.60	219.10	477.93	361.62

Comparando los valores con diversificación e intensificación contra los valores sin utilizarlos podemos ver que son los mismos salvo para la instancia r201. Utilizando diversificación e intensificación con GTSxD con β=1.5 se logró mejorar el valor mientras que para GTSxC con C=15 se empeoró el resultado.

Resumiendo los resultados obtenidos para este conjunto de instancias, podemos ver que los resultados no difieren demasiado (excepto en algún caso puntual) utilizando los diferentes parámetros. Tomando en cuenta esto y que el tiempo necesario para ejecutar las instancias de 25 clientes es verdaderamente corto (ejecutar 56 instancias lleva aproximadamente 5 minutos) decidimos ejecutar las 56 instancias con cada uno de los parámetros vistos anteriormente. Las siguientes tablas muestran los mejores resultados obtenidos para las 56 instancias de 25 clientes y los parámetros con los que se obtuvieron.

Como mencionamos al comienzo de la sección, los valores contra los cuales comparamos nuestros resultados fueron obtenidos en *The VRP Web* [17]. Aquellos resultados obtenidos que superan los mejores conocidos están resaltados en negrita.

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia	Parámetros
R101.txt	617.1	8	618.33	8	0.2	GTSxC C=15
R102.txt	547.1	7	548.11	7	0.18	GTSxC C=10
R103.txt	454.6	5	470.02	6	3.39	GTSxD β=1
R104.txt	416.9	4	423.49	4	1.58	GTSxD β=1.5
R105.txt	530.5	6	536.88	6	1.2	GTSxC C=5
R106.txt	465.4	5	467.85	5	0.53	GTSxC C=10
R107.txt	424.3	4	427.89	4	0.85	GTSxC C=10
R108.txt	397.3	4	400.2	4	0.73	Operaciones 1,2,3,4
R109.txt	441.3	5	443.43	5	0.48	GTSxC C=5
R110.txt	444.1	4	449.84	5	1.29	GTSxD β=1.5
R111.txt	428.8	4	432.32	4	0.82	GTSxC C=15 con Div + Int
R112.txt	393	4	399.29	4	1.6	GTSxC C=5
C101.txt	191.3	3	198.6	3	3.82	GTSxC C=5
C102.txt	190.3	3	191.53	3	0.64	GTSxC C=10
C103.txt	190.3	3	191.53	3	0.64	GTSxC C=10
C104.txt	186.9	3	188.24	3	0.72	GTSxD β=1.5
C105.txt	191.3	3	193.04	3	0.91	GTSxC C=10
C106.txt	191.3	3	195.14	3	2.01	GTSxC C=5
C107.txt	191.3	3	192.64	3	0.7	GTSxC C=15
C108.txt	191.3	3	192.18	3	0.46	GTSxC C=10
C109.txt	191.3	3	192.18	3	0.46	GTSxC C=10
RC101.txt	461.1	4	477.93	5	3.65	GTSxC C=10
RC102.txt	351.8	3	402.7	4	14.47	GTSxC C=15 con Div + Int
RC103.txt	332.8	3	334.96	3	0.65	Operaciones 1,2,3,4
RC104.txt	306.6	3	307.94	3	0.44	GTSxC C=10
RC105.txt	411.3	4	412.38	4	0.26	GTSxC C=10
RC106.txt	345.5	3	348.88	3	0.98	GTSxC C=15 con Prioridad
RC107.txt	298.3	3	300.26	3	0.66	Operaciones
RC108.txt	294.5	3	296.51	3	0.68	Operaciones 1,2,3,4

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia	Parámetros
R201.txt	463.3	4	468.51	4	1.12	GTSxD β=1.5 con Div + Int
R202.txt	410.5	4	418.9	**3**	2.05	GTSxC C=5
R203.txt	391.4	3	393.49	3	0.53	GTSxC C=15
R204.txt	355	2	374.38	2	5.46	GTSxC C=15
R205.txt	393	3	411.28	4	4.65	GTSxC C=15 con Prioridad
R206.txt	374.4	3	377.41	4	0.8	GTSxC C=5
R207.txt	361.6	3	368.85	**2**	2	GTSxC C=15 con Prioridad
R208.txt	328.2	1	341.02	2	3.91	GTSxD β=2
R209.txt	370.7	2	373.62	2	0.79	Operaciones 1,2,3,4
R210.txt	404.6	3	415.78	4	2.76	GTSxD β=1.5 con Div + Int
R211.txt	350.9	2	357.6	2	1.91	GTSxD β=1.5 con Div + Int
C201.txt	214.7	2	219.1	2	2.05	GTSxC C=5
C202.txt	214.7	2	216.86	2	1	GTSxC C=5
C203.txt	214.7	2	216.86	2	1	Operaciones 1,2,3,4
C204.txt	213.1	1	215.37	2	1.07	GTSxC C=15
C205.txt	214.7	2	216.39	2	0.79	GTSxC C=10
C206.txt	214.7	2	216.39	2	0.79	GTSxC C=10
C207.txt	214.5	2	215.37	2	0.41	GTSxC C=15
C208.txt	214.5	2	218.93	2	2.07	GTSxC C=5
RC201.txt	360.2	3	361.62	3	0.39	GTSxC C=10
RC202.txt	338	3	338.87	3	0.26	GTSxD β=1
RC203.txt	326.9	3	328.73	3	0.56	GTSxD β=1
RC204.txt	299.7	3	300.23	3	0.18	GTSxC C=15
RC205.txt	338	3	339.69	3	0.5	GTSxC C=10
RC206.txt	324	3	326.87	3	0.89	GTSxC C=10
RC207.txt	298.3	3	300.47	3	0.73	Operaciones 1,2,3,4
RC208.txt	269.1	2	294.59	2	9.47	GTSxC C=10

Observando los resultados obtenidos, notamos que con los parámetros de GTSxC se obtienen la gran mayoría de los mejores resultados de nuestro algoritmo. Si bien no se logró mejorar la distancia de ninguna instancia, podemos destacar que se encontraron 2 muy buenos resultados que son en las instancias r202 y r207. Estos resultados los destacamos ya que el criterio contra con el cual se obtuvieron los mejores resultados es minimizar primero la cantidad de vehículos y luego la distancia recorrida.

8.3. Instancias de 50 clientes

Es esta sección presentamos las evaluaciones de parámetros y resultados obtenidos en las instancias de 50 clientes. La forma en la que evaluamos la conveniencia de los parámetros y presentaremos los resultados será igual que en la sección anterior. El subconjunto de instancias muestrales que seleccionamos son: c101.txt, r101.txt, rc101.txt, c102.txt, r102.txt, rc102.txt. Los parámetros del Tabu Search que fueron prefijados son:

- Tenure = 15
- Método de Corte = 50 iteraciones sin mejora

La siguiente tabla muestra los resultados obtenidos en estas instancias para los diferentes valores de tipo de granular.

Parámetros	r101	r201	c101	c201	rc101	rc201
Con las operaciones 1, 2, 3 y 4	13 1048.04	9 834.44	5 370.03	3 365.36	11 1041.97	8 841.33
Con las operaciones 1, 2, 3, 4, 5, 6	13 1078.94	7 848.05	5 370.03	3 365.36	9 956.17	5 709.74
GTSxD $\beta = 1$ Todas las op.	12 1060.65	7 820.60	5 370.03	4 411.23	9 956.11	6 790.75
GTSxD $\beta = 1.5$ Todas las op.	13 1061.71	8 850.51	5 370.03	3 365.36	9 956.11	5 742.73
GTSxD $\beta = 2$ Todas las op.	12 1061.09	7 837.40	5 370.03	3 365.36	9 956.11	5 742.73
GTSxC $C = 10$ Todas las op.	12 1060.87	7 848.47	5 370.03	3 365.36	9 956.11	5 742.73
GTSxC $C = 15$ Todas las op.	13 1048.04	7 846.90	5 370.03	3 365.36	9 956.11	5 686.66
Con GTSxC $C = 20$ Todas las op.	13 1049.09	7 850.52	5 370.03	3 365.36	9 956.11	5 718.35

Al igual que con las instancias de 25 clientes, podemos ver que los resultados obtenidos para los distintos valores de β y C son relativamente iguales salvo en algunos casos puntuales.

Siguiendo con la evaluación de los parámetros, la siguiente tabla muestra los valores de las instancias de prueba con los parámetros de GTSxD utilizando la solución inicial.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	12	7	5	4	9	6
$\beta = 1$						
Todas las op.	1067.65	820.69	370.03	411.23	956.11	790.76
GTSxD	13	8	5	3	9	5
$\beta = 1.5$						
Todas las op.	1061.71	850.51	370.03	365.36	956.11	742.73
GTSxD	12	7	5	3	9	5
$\beta = 2$						
Todas las op.	1061.09	837.40	370.03	365.36	956.11	742.73

Comparando los resultados obtenidos al utilizar la solución inicial y la mejor solución podemos ver que al igual que en las instancias de 25 clientes los valores con ambos parámetros son casi iguales por lo que es conveniente el uso de la solución inicial cuando no se cuenta con la mejor solución.

A continuación presentamos los resultados de evaluar las instancias con prioridades. Para determinar las prioridades, de la misma manera en que lo hicimos en la sección anterior, medimos en cada una de las instancias el porcentaje de tiempo de ejecución de cada una de las operaciones, el porcentaje de veces que la operación obtuvo un resultado mejor que los obtuvidos por las operaciones restantes y el porcentaje de mejora obtenido. Los valores presentados a continuación se obtuvieron al ejecutar el algoritmo con todas las operaciones y sin granular. En la tabla no se muestra la operación 3 debido a que para ninguna instancia se cumplió la precondición de ejecución de la operación. Como notamos que la operación 3 no logra aportar resultados y produce cálculos innecesarios, la dejaremos de utilizar.

Instancia r101	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	20.86 %	9.82 %	35.79 %	13.29 %	19.84 %
Iteraciones Ganadas	51.49 %	24.75 %	17.82 %	0 %	5.94 %
Valores Mejora Negativos	64.74 %	6.92 %	0.06 %	0 %	29.28 %
Instancia r201	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	9.82 %	9.52 %	65.76 %	6.13 %	8.63 %
Iteraciones Ganadas	72.73 %	9.09 %	12.12 %	0 %	6.06 %
Valores Mejora Negativos	64.88 %	7.54 %	4.56 %	0 %	23.03 %
Instancia c101	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	11.41 %	19.13 %	51.85 %	8.72 %	8.89 %
Iteraciones Ganadas	55.88 %	17.65 %	17.65 %	0 %	8.82 %
Valores Mejora Negativos	59.06 %	5.19 %	5.18 %	0 %	30.57 %
Instancia c201	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	7.84 %	10.30 %	72.34 %	3.47 %	6.05 %
Iteraciones Ganadas	49.18 %	19.67 %	19.67 %	0 %	11.48 %
Valores Mejora Negativos	18.21 %	7.24 %	12.93 %	0 %	61.61 %

Las conclusiones que obtenemos a partir de estos resultados son los siguientes:

- la operación 1 es la que obtiene valores mejores en la mayor cantidad de iteraciones
- en proporción a las iteraciones ganadas, la operación 6 es la que obtiene los mejores valores de mejora, y en la instancia particular c201 el valor proporcional es notoriamente mayor.
- la operación 4 ayuda a encontrar buenos resultados pero el tiempo de ejecución es muy superior al resto.

Evaluando los valores obtenidos para determinar las prioridades, llegamos a las mismas conclusiones de las instancias de 25 clientes, por lo que la secuencia de prioridades de las operaciones es nuevamente 1,6,2,4,5. En la siguiente tabla vemos los resultados de ejecutar las instancias muestrales con prioridades.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	13	7	5	4	10	5
$\beta = 1.5$						
Todas las op.	1061.21	807.72	370.03	411.13	976.08	739.31
GTSxC	13	7	5	3	10	5
$C = 15$						
Todas las op.	1053.85	807.72	370.03	365.36	976.08	739.31

Como podemos observar, la mayoría de los valores son similares a los obtenidos sin el uso de prioridades. En algunos casos puntuales (como r201) hay mejoras notables con respecto a los valores anteriores, mientras que en otros casos (como rc201) los valores empeoran.

Por último evaluaremos las instancias con el parámetro de diversificación e intensificación.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	13	8	5	3	9	5
$\beta = 1.5$						
Todas las op.	1053.32	848.99	370.03	365.36	956.10	742.73
GTSxC	13	7	5	3	9	5
$C = 15$						
Todas las op.	1049.67	846.84	370.03	365.36	956.10	686.57

Al igual que en las instancias de 25 clientes, podemos ver que los resultados obtenidos en las instancias de muestra con diferentes parámetros no difieren mucho entre sí, con la excepción de algún caso puntual. Teniendo en cuenta esto y que el tiempo necesario para ejecutar las instancias de 50 clientes no es muy alto decidimos ejecutar las 56 instancias con cada uno de los parámetros vistos anteriormente. Las siguientes tablas muestran los mejores resultados obtenidos para las 56 instancias de 50 clientes y los parámetros con los que se obtuvieron.

Como mencionamos al comienzo de la sección, los valores contra los cuales compararemos los resultados obtenidos los obtuvimos en *The VRP Web* [17]. Los resultados obtenidos que superen los mejores conocidos se resaltan en negrita.

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia	Parámetros
R101.txt	1047	13	1048.04	13	0.1	GTSxC C=10
R102.txt	944.9	12	**918.43**	**12**	**-2.8**	GTSxC C=5
R103.txt	772.9	9	782.18	9	1.2	GTSxD β=2 sin mejor solucion
R104.txt	631.2	6	648.02	7	2.66	GTSxC C= 2 con div + int
R105.txt	906.6	10	919.24	10	1.39	GTSxC C=15 con prioridad
R106.txt	793.6	8	798.09	8	0.57	GTSxC C= 2 con div + int
R107.txt	720.4	7	**718.41**	**7**	**-0.28**	GTSxC C=5
R108.txt	618.2	6	641.56	6	3.78	Todas las operaciones
R109.txt	803.2	8	804.39	9	0.15	GTSxC C=5
R110.txt	724.9	8	**719.25**	**8**	**-0.78**	GTSxC C=10
R111.txt	724.9	8	**722.67**	**8**	**-0.31**	GTSxC C=15 con prioridad
R112.txt	651.1	6	652.05	7	0.15	GTSxC Ct=15 con prioridad
C101.txt	362.4	5	370.03	5	2.11	GTSxC C=5
C102.txt	361.4	5	362.96	5	0.43	GTSxC C=5
C103.txt	361.4	4	362.96	5	0.43	GTSxD β=2 sin mejor solucion
C104.txt	359	5	383.92	5	6.94	Todas las operaciones
C105.txt	362.4	5	364.47	5	0.57	GTSxC C=5
C106.txt	362.4	5	364.68	5	0.63	GTSxC C=5
C107.txt	362.4	5	364.47	5	0.57	GTSxC C=15
C108.txt	362.4	5	363.25	5	0.23	GTSxC C=5
C109.txt	362.4	5	363.25	5	0.23	GTSxC C=10
RC101.txt	957.9	9	**956.11**	**9**	**-0.19**	GTSxC C=5
RC102.txt	844.3	8	886.47	9	4.99	GTSxC C=5
RC103.txt	712.6	6	807.58	8	13.33	Todas las operaciones
RC104.txt	546.5	5	556.21	5	1.78	GTSxC C=10
RC105.txt	888.9	9	890.56	9	0.19	GTSxC C=5
RC106.txt	791.9	7	**772.54**	**7**	**-2.44**	Con todas las operaciones
RC107.txt	664.5	6	684.85	6	3.06	GTSxC C=10
RC108.txt	598.1	6	604.35	6	1.05	Todas las operaciones

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia	Parámetros
R201.txt	800.7	6	807.72	7	0.88	GTSxD $\beta = 1.5$ con prioridad
R202.txt	712.2	5	**708.98**	6	**-0.45**	GTSxC C=5
R203.txt	606.4	5	608.99	5	0.43	GTSxC cant=15 con prioridad
R204.txt	509.5	2	514.27	2	0.94	GTSxD β=2
R205.txt	703.3	5	703.72	5	0.06	GTSxC C=15
R206.txt	647	5	**644.69**	**4**	**-0.36**	GTSxC C=15 con prioridad
R207.txt	584.6	4	**582.24**	**4**	**-0.4**	GTSxD β=1.5
R208.txt	487.7	2	516.78	2	5.96	GTSxC C=10
R209.txt	600.6	4	604.66	4	0.68	GTSxC C=15
R210.txt	663.4	5	**654.9**	**5**	**-1.28**	GTSxC C=5
R211.txt	551.3	3	557.71	3	1.16	GTSxC C=5
C201.txt	360.2	3	365.36	3	1.43	GTSxC C=5
C202.txt	360.2	3	363.42	3	0.89	GTSxD β=2
C203.txt	359.8	3	362.73	3	0.81	Operaciones 1,2,3,4
C204.txt	353.4	2	365.71	3	3.48	GTSxD β=1 sin mejor solucion
C205.txt	359.8	3	361.8	3	0.55	Con GTSxD β=1.5
C206.txt	359.8	3	361.8	3	0.55	GTSxC C=10
C207.txt	359.6	3	364.69	3	1.42	GTSxC C=15
C208.txt	350.5	2	359.34	2	2.52	Todas las operaciones
RC201.txt	684.8	5	686.57	5	0.26	GTSxC cant= 2 con div + int
RC202.txt	613.6	5	649.16	5	5.79	GTSxC C=5
RC203.txt	555.3	4	602.54	5	8.51	GTSxD β=1
RC204.txt	444.2	3	500.08	4	12.58	GTSxD β=1 sin mejor solucion
RC205.txt	631	5	635.59	5	0.73	GTSxD β=2 sin mejor solucion
RC206.txt	610	5	616.48	5	1.06	GTSxC C=5
RC207.txt	558.6	4	568	4	1.68	GTSxD β=2
RC208.txt	-	-	535.88	3	-	GTSxC C=10

Observando los resultados obtenidos, notamos que con los parámetros de GTSxC se obtienen la gran mayoría de los mejores resultados de nuestro algoritmo. Para este conjunto de instancias se lograron mejorar 10 de los mejores valores publicados hasta el momento en [17]. También podemos destacar que para la instancia r206 además de mejorar el valor se encontró una solución que mejora la cantidad de vehículos necesaria.

8.4. Instancias de 100 clientes

Es esta sección presentamos las evaluaciones de parámetros y resultados obtenidos con las instancias de 100 clientes. La forma en la que probamos la conveniencia de los parámetros y la presentación de los resultados será la misma de las secciones anteriores. En particular, para las instancias de 100 clientes conocemos resultados del algoritmo para VRPTWSD propuesto por Ho y Haugland en [13] con lo que también compararemos estos resultados con los nuestros. El subconjunto de instancias de prueba que fueron seleccionadas son: c101.txt, r101.txt, rc101.txt, c102.txt, r102.txt y rc102.txt.

Los parámetros que fueron prefijados son:

- Tenure = 15
- Método de Corte = 70 iteraciones sin mejora

Parámetros	r101	r201	c101	c201	rc101	rc201
Con las operaciones 1, 2, 3 y 4	24 1757.04	13 1249.54	10 829.70	3 594.32	19 1786.33	12 1437.26
Con las operaciones 1, 2, 3, 4, 5, 6	23 1712.45	9 1204.32	10 829.70	3 595.12	18 1737.20	9 1331.07
GTSxD $\beta = 1$ Todas las op.	22 1709.50	11 1199.41	10 829.70	5 936.92	18 1746.33	11 1421.95
GTSxD $\beta = 1.5$ Todas las op.	23 1708.38	10 1183.71	10 829.70	4 635.86	17 1713.63	9 1354.21
GTSxD $\beta = 2$ Todas las op.	22 1694.27	10 1199.83	10 829.70	3 603.77	18 1741.70	10 1304.65
GTSxC $C = 10$ Todas las op.	23 1733.08	12 1223.31	10 829.70	3 594.32	18 1769.07	12 1330.73
GTSxC $C = 15$ Todas las op.	24 1754.26	12 1206.44	10 829.70	3 594.32	18 1749.51	10 1322.66
GTSxC $C = 20$ Todas las op.	23 1708.70	9 1189.01	10 829.70	3 594.32	18 1738.13	10 1358.14

La siguiente tabla muestra los resultados de las instancias de muestra al utilizar la solución inicial en GSTxD.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	23	10	10	3	17	11
$\beta = 1$						
Todas las op.	1708.38	1197.19	829.70	594.32	1713.63	1435.12
GTSxD	22	10	10	3	18	10
$\beta = 1.5$						
Todas las op.	1694.27	1199.83	829.70	594.32	1744.77	1304.65
GTSxD	22	10	10	3	18	10
$\beta = 2$						
Todas las op.	1684.27	1190.72	829.70	594.32	1744.69	1310.60

Observando los resultados notamos que no existe una diferencia significativa con respecto a los valores obtenidos con la mejor solución. En algunas instancias el uso de la solución inicial permite obtener mejores resultados (por ejemplo en r101) mientras que en otras instancias los resultados empeoraron (por ejemplo en rc101). Para los valores de β=1 se encontraron valores significativamente mayores, esto se debe a que resulta muy restrictivo quedarse sólo con los vecinos que se encuentran a una distancia menor al promedio de las distancias de la mejor solución.

Examinando los resultados obtenidos en la primera tabla, notamos que en líneas generales los valores de β=2 para GTSxD y C=20 para GTSxC obtienen buenos resultados. Si bien estos valores no siempre garantizan el mejor resultado, parecen ser los más convenientes y los fijamos para continuar evaluando el resto de los parámetros en función de estos valores.

A continuación presentamos los resultados de evaluar las instancias con prioridades. Para determinar las prioridades realizamos el mismo procedimiento que en las instancias de 25 y 50 clientes. Los valores presentados a continuación se obtuvieron al ejecutar el algoritmo con todas las operaciones y sin granular. Recordamos que la operación 3 fue descartada porque demostró no ser útil en el procesamiento.

Instancia r101	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	19.08 %	11.44 %	39.28 %	11.97 %	18.23 %
Iteraciones Ganadas	64.91 %	20.61 %	7.02 %	0 %	7.46 %
Valores Mejora Negativos	56.60 %	5.62 %	4.54 %	0 %	33.24 %
Instancia r201	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	6.63 %	10.65 %	71.89 %	4.29 %	6.34 %
Iteraciones Ganadas	82.18 %	9.41 %	3.47 %	0 %	4.95 %
Valores Mejora Negativos	52.42 %	11.50 %	7.76 %	0 %	28.32 %
Instancia c101	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	9.39 %	20.80 %	53.96 %	7.46 %	8.40 %
Iteraciones Ganadas	63.54 %	9.38 %	8.33 %	0 %	18.75 %
Valores Mejora Negativos	33.05 %	2.72 %	14.07 %	0 %	50.16 %
Instancia c201	Op1	Op2	Op4	Op5	Op6
Tiempo Ejecución	5.54 %	17.08 %	69.59 %	2.36 %	4.43 %
Iteraciones Ganadas	45.45 %	13.13 %	27.27 %	0 %	14.14 %
Valores Mejora Negativos	29.69 %	1.08 %	13.82 %	0 %	55.40 %

Evaluando los datos obtenidos para determinar las prioridades, podemos realizar las mismas conclusiones que las obtenidas en las instancias de 25 y 50 clientes; por lo que la secuencia de prioridades entre las operaciones es nuevamente 1,6,2,4,5. Teniendo en cuenta que la operación 6 cuando encuentra un valor que mejora la solución es mucho mejor que el encontrado por el resto de las operaciones (esto se puede ver con los valores de mejora negativa), decidimos evaluar la secuencia 6,1,2,4,5. En la siguiente tabla veremos algunos resultados obtenidos al ejecutar las instancias de Solomon con las dos secuencias prioridades.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD β=2 con prioridad 1,6,2,4,5	24 1724.78	10 1189.48	10 829.70	3 594.32	19 1749.70	10 1328.29
GTSxD β=2 con prioridad 6,1,2,4,5	22 1705.24	10 1223.03	10 829.70	4 632.29	18 1739.52	8 1377.09
GTSxC C=20 con prioridad 1,6,2,4,5	23 1700.13	11 1202.17	10 829.70	3 594.32	17 1748.97	9 1387.70
GTSxC C=20 con prioridad 6,1,2,4,5	24 1759.91	9 1208.06	10 829.70	3 594.32	17 1733.30	9 1337.91

Evaluando los resultados obtenidos notamos que para algunas instancias es mejor una secuencia y para otras es mejor utilizar otra secuencia de operaciones, por lo que no podemos concluir que una sea mejor que otra.

Finalmente evaluamos las instancias con el parámetro de diversificación e intensificación. La siguiente tabla contiene los resultados obtenidos con estos parámetros.

Parámetros	r101	r201	c101	c201	rc101	rc201
GTSxD	23	10	10	4	17	10
β=2						
Todas las Op.	1711.16	1199.83	829.70	632.29	1742.96	1304.65
GTSxC	23	10	10	3	18	9
C=20						
Todas las Op.	1709.32	1206.81	829.70	594.32	1738.86	1361.61

Observando los resultados obtenidos podemos ver que la mayoría son relativamente mayores a los resultados obtenidos sin el uso de diversificación e intensificación; más específicamente, el desvío promedio para cada instancia es de 0.64.

A continuación presentamos una tabla que contiene los resultados obtenidos por nuestro algoritmo para todas las instancias de Solomon de 100 clientes. Los resultados se comparan con los mejores valores encontrados para el problema VRPTW y VRPTWSD que se muestran en las primeras columnas de la tabla. Los valores VRPTW que aparecen en la segunda columna fueron obtenidos teniendo como objetivo minimizar primero la cantidad de vehículos y luego la distancia recorrida. En la tercera columna aparecen los valores VRPTWSD obtenidos en [13] teniendo como objetivo minimizar la distancia recorrida. En la cuarta columna se muestran los valores obtenidos por nuestro algoritmo y en la quinta columna aparece el porcentaje de diferencia que compara nuestros valores contra los mejores de VRPTW. Resaltaremos en negrita los resultados que mejoran los mejores valores conocidos según nuestro criterio de minimización.

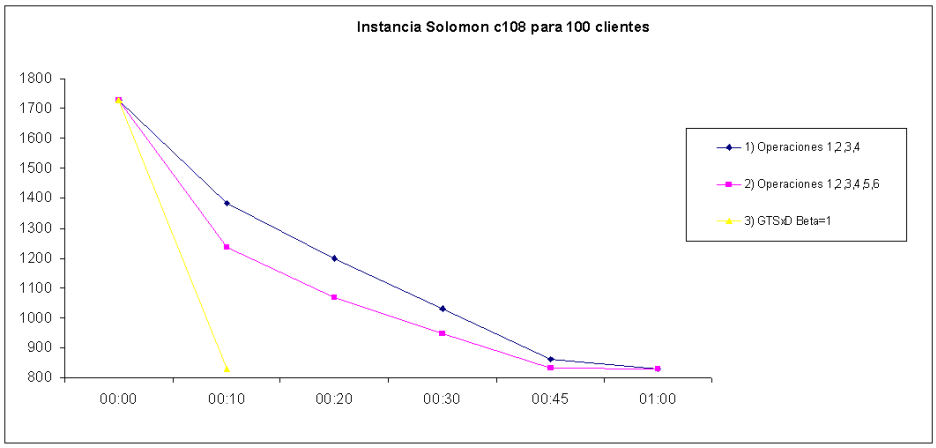
Instancia	Mejor VRPTW		Mejor VRPTWSD		Nuestro Algoritmo		% diferencia	Parámetros
R101.txt	1650.8	19	1648.96	20	1694.27	22	2.63	GTSxD β=2
R102.txt	1486.12	17	1506.61	18	1490.24	19	0.28	Todas las operaciones
R103.txt	1292.68	13	1278.93	15	**1227.98**	15	**-5.01**	GTSxD β=2 con prio 6,1,2,4,5
R104.txt	1007.31	9	1099.93	13	**1006.06**	11	**-0.12**	GTSxD β=1 sin mejor solucion
R105.txt	1377.11	14	1453.58	16	1384.43	16	0.53	GTSxC C=20
R106.txt	1252.03	12	1292.29	14	1277.78	15	2.06	GTSxC C=20
R107.txt	1104.66	10	1150.06	12	**1103.78**	12	**-0.08**	Todas las operaciones
R108.txt	960.876	9	1004.42	11	967.57	11	0.7	GTSxD β=2 con div + int
R109.txt	1194.73	11	1270.66	13	**1174.06**	13	**-1.73**	GTSxC C=20 con div + int
R110.txt	1118.84	10	1232.53	13	**1116.33**	12	**-0.22**	GTSxD β=1.5 sin mejor solucion
R111.txt	1096.72	10	1106.37	12	1103.55	13	0.62	GTSxC C=20
R112.txt	982.14	9	1042.44	11	1007.65	11	2.6	GTSxD β=2 con prio 6,1,2,4,5
C101.txt	828.94	10	828.94	10	829.7	10	0.09	GTSxC C=10
C102.txt	828.94	10	842.02	10	829.7	10	0.09	GTSxC C=10
C103.txt	828.06	10	890.57	10	828.83	10	0.09	Todas las operaciones
C104.txt	824.78	10	894.41	10	825.11	10	0.04	GTSxD β=1.5
C105.txt	828.94	10	828.94	10	829.7	10	0.09	GTSxC C=10
C106.txt	828.94	10	828.94	10	829.38	10	0.05	GTSxC C=10
C107.txt	828.94	10	828.94	10	829.38	10	0.05	GTSxC C=20
C108.txt	828.94	10	830.33	10	828.94	10	0	GTSxD β=1
C109.txt	828.94	10	830.33	10	828.94	10	0	GTS β=2 sin mejor solución
RC101.txt	1696.94	14	1748.15	16	1713.63	17	0.98	GTSxD β=1.5
RC102.txt	1554.75	12	1568.86	15	1565.88	16	0.72	GTSxD β=2 sin mejor solución
RC103.txt	1261.67	11	1325.84	12	1282.72	12	1.67	GTSxD β=1.5
RC104.txt	1135.48	10	1222.79	11	1210.41	11	6.6	GTSxC C=15
RC105.txt	1629.44	13	1655.93	17	**1566.88**	16	**-3.84**	GTSxD β=2
RC106.txt	1424.73	11	1443.88	14	1434.98	14	0.72	Todas las operaciones
RC107.txt	1230.48	11	1278.21	12	1274.57	12	3.58	GTSxC C=20 con div + int
RC108.txt	1139.82	10	1314.01	12	1196.88	12	5.01	GTSxD β=2 con prio 6,1,2,4,5

Instancia	Mejor VRPTW		Mejor VRPTWSD		Nuestro Algoritmo		% diferencia	Parámetros
R201.txt	1252.37	4	1271.17	5	**1183.71**	10	**-5.48**	GTSxD β=1.5
R202.txt	1191.7	3	1171.38	4	**1071.79**	9	**-10.06**	Operaciones 1,2,3,4
R203.txt	941.408	3	1034.94	4	**918.74**	8	**-2.41**	GTSxC C=20
R204.txt	825.519	2	854.85	4	**784.63**	6	**-4.95**	GTSxC C=20
R205.txt	994.42	3	1078.79	4	1009.89	7	1.56	GTSxD β=2 sin mejor solución
R206.txt	912.97	3	1010.33	3	937.16	6	2.65	GTSxC C=10
R207.txt	893.328	2	1009.08	3	**860.4**	6	**-3.69**	Todas las operaciones
R208.txt	726.823	2	827.12	3	729.81	4	0.41	GTSxC C=20
R209.txt	909.163	3	1046.69	3	915.92	6	0.74	GTSxC C=20 con prio 6,1,2,4,5
R210.txt	939.37	3	1015.93	4	960.03	7	2.2	GTSxD β=2
R211.txt	892.713	2	839.85	3	**815.49**	5	**-8.65**	GTSxD β=1.5
C201.txt	591.56	3	591.56	3	594.32	3	0.47	GTSxC C=10
C202.txt	591.56	3	620.29	3	593.18	3	0.27	Todas las operaciones
C203.txt	591.17	3	623.24	3	619.11	3	4.73	Todas las operaciones
C204.txt	590.6	3	645.26	3	649.04	5	9.9	GTSxC C=20 con prio 6,1,2,4,5
C205.txt	588.88	3	596.89	3	589.72	3	0.14	Todas las operaciones
C206.txt	588.49	3	625.71	3	588.88	3	0.07	Operaciones 1,2,3,4
C207.txt	588.29	3	597.25	3	591.77	3	0.59	GTSxC C=20 con prio 6,1,2,4,5
C208.txt	588.32	3	613.96	3	588.71	3	0.07	GTSxC C=10
RC201.txt	1406.94	4	1379.53	5	**1304.65**	10	**-7.27**	GTSxD β=2
RC202.txt	1377.089	3	1390.44	4	**1120.39**	8	**-18.64**	Todas las operaciones
RC203.txt	1060.45	3	1150.76	4	**983.35**	7	**-7.27**	GTSxD β=1.5
RC204.txt	798.464	3	1021.6	3	827.23	5	3.6	GTSxD β=2 sin mejor solución
RC205.txt	1297.65	4	1482.17	5	**1193.65**	9	**-8.01**	GTSxD β=2 con div + int
RC206.txt	1146.32	3	1260.99	4	**1104.34**	8	**-3.66**	GTSxD β=1.5 sin mejor solución
RC207.txt	1061.14	3	1107.84	4	**1019.72**	7	**-3.9**	GTSxD β=1.5 sin mejor solución
RC208.txt	828.141	3	1029.76	3	831.9	5	0.45	GTSxC C=20

Analizando los mejores valores encontrados por nuestro algoritmo, podemos destacar que en la instancias c108.txt y c109.txt se obtuvieron los mismos mejores resultados conocidos hasta el momento. Por otro lado, al estar utilizando el criterio de minimización primero por distancia recorrida y luego por la cantidad de vehículos obtuvimos para un conjunto grande de instancias (17 de las 56 instancias) mejoras en los valores de distancia pero no así en la cantidad de vehículos. Un ejemplo claro es en la instancia rc105.txt donde el mejor valor encontrado es 1629.44 con 13 vehículos y nuestro algoritmo encontró el valor 1566.88 con 16 vehículos.

Como el objetivo de nuestro trabajo es evaluar las variantes de Tabu Search Granular y las conveniencias de uso en el problema de VRPTWSD, generamos dos gráficos en donde pueden observarse y compararse claramente los valores que va obteniendo el algoritmo con los diferentes parámetros en función del tiempo.

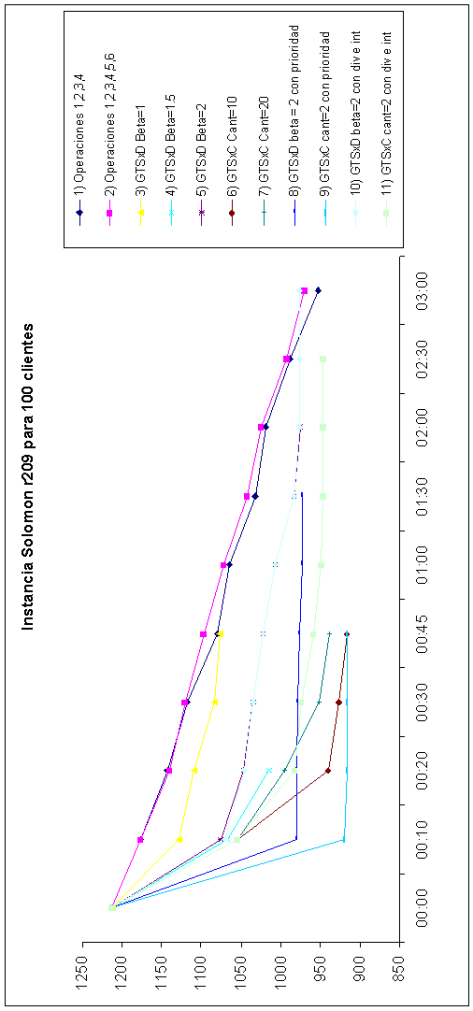
El primer gráfico ilustra los valores que fue obteniendo el algoritmo para la instancia c108.txt. Se seleccionaron las tres veriantes más significativas ya que el resto de los valores de los parámetros no diferían demasiado del mejor encontrado con GTS.



El gráfico permite notar claramente que el tiempo necesario para que el algoritmo llegue a encontrar el mejor valor es notoriamente menor si se utiliza GTS. La disminución del tiempo de ejecución se debe a la eliminación

de vecinos que no aportan buenos resultados lográndose la exploración de vecindarios reducidos y más prometedores de buenas soluciones.

En el segundo gráfico se pueden comparar los valores para la instancia r209.txt. Los valores presentados permiten comparar el comportamiento del algoritmo con diferentes parámetros, incluyendo el uso de prioridades. Claramente GTS en sus dos variantes logra una reducción importante del tiempo de ejecución.



8.5. Instancias de 200 clientes

Es esta sección presentamos las evaluaciones de parámetros y resultados obtenidos con las instancias de 200 clientes. La forma en la que probaremos la conveniencia de los parámetros y presentaremos los resultados es la misma de las secciones anteriores. El subconjunto de instancias de muestra que fueron seleccionadas son: r1_2_1.txt, r2_2_1.txt, c1_2_1.txt, c2_2_1.txt, rc1_2_1.txt y rc2_2_1.txt.

Parámetros	r1_2_1	r2_2_1	c1_2_1	c2_2_1	rc1_2_1	rc2_2_1
Con las operaciones 1, 2 y 4	34 5223.95	21 4032.3	20 2705.43	13 2240.29	24 3823.11	19 3464.23
Con las operaciones 1, 2, 4, 5, 6	30 5078.65	13 3690.42	20 2705.43	8 1977.99	22 3713.97	12 3032.01
GTSxD $\beta = 1$ Todas las op.	32 5219.39	16 3780.10	20 2711.92	17 2700.39	26 4171.07	20 3763.93
GTSxD $\beta = 1.5$ Todas las op.	28 4925.91	14 3720.01	20 2705.83	11 2145.77	24 3912.16	16 3251.40
GTSxD $\beta = 2$ Todas las op.	29 5000.94	14 3748.79	20 2708.30	10 2099.05	22 3798.93	13 3046.68
GTSxC $C = 20$ Todas las op.	30 4950.05	15 3789.40	20 2705.43	7 1949.54	24 3899.13	15 3293.01
GTSxC $C = 30$ Todas las op.	29 4968.89	16 3730.80	20 2715.01	8 1947.32	23 3859.86	12 2998.91
GTSxC $C = 40$ Todas las op.	29 4989.61	16 3761.41	20 2715.01	7 1937.20	23 3848.65	12 3112.20

Los parámetros que fueron fijados son:

- Tenure = 30
- Método de Corte = 120 iteraciones sin mejora

Estos resultados indican que los mejores parámetros son β=2 en GTSxD y C=30 en GTSxC. Sin embargo, de la misma manera en que ocurre en los casos con menores cantidades de clientes, la elección de estos parámetros no garantiza que el algoritmo vaya a comportarse de la mejor manera en todas las instancias.

En las secciones anteriores corroboramos la conveniencia de uso de diversificación e intensificación así como también el uso de prioridades entre las operaciones obteniendo buenos resultados con la secuencia 6,1,2,4,5. A continuación aplicaremos estas variantes a los parámetros de GTSxD y GTSxC que arrojaron los mejores resultados hasta el momento.

Parámetros GTSxD β=2						
Parámetros	r1_2_1	r2_2_1	c1_2_1	c2_2_1	rc1_2_1	rc2_2_1
Con Prioridades 6,1,2,4,5	28 4898.39	14 3710.63	20 2708.16	8 1987.91	23 3817.66	13 3028.92
Con Diversificación e Intensificación	30 5006.66	15 3714.26	20 2709.07	7 1937.70	20 3618.15	12 2974.05
Con Prioridades más Diversificación e Intensificación	27 4910.44	13 3673.40	20 2708.39	10 2084.81	22 3816.38	13 3045.42
Parámetros GTSxC C=30						
Parámetros	r1_2_1	r2_2_1	c1_2_1	c2_2_1	rc1_2_1	rc2_2_1
Con Prioridades 6,1,2,4,5	27 4903.54	15 3768.38	20 2705.43	7 1934.10	23 3845.49	12 3073.00
Con Diversificación e Intensificación	28 4917.32	14 3645.79	20 2715.01	7 1937.20	22 3774.84	12 3013.63
Con Prioridades más Diversificación e Intensificación	28 4905.99	16 3820.39	20 2707.85	7 1933.64	25 3939.94	13 3110.93

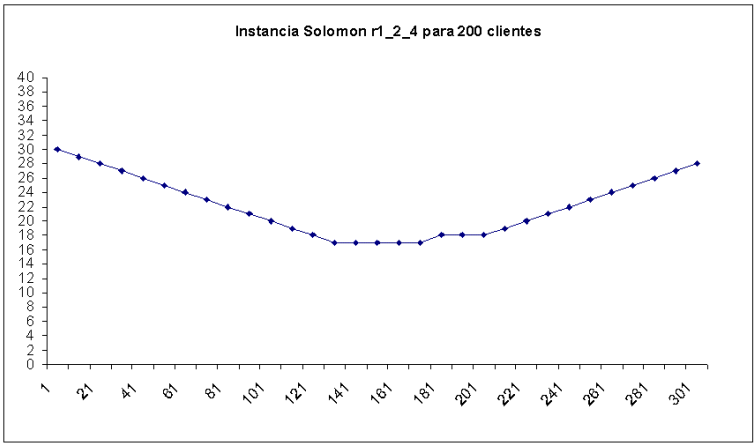
Comparando los resultados de utilizar prioridades junto a diversificación e intensificación contra los resultados sin el uso de los mismos podemos observar que en la mitad de los casos se obtuvieron mejores resultados, por lo que

podemos concluir que es conveniente el uso de los mismos. Una segunda razón (no menos importante) por la que conviene hacer uso de estos mecanismos es el tiempo de ejecución. En las instancias "chicas"la evaluación del tiempo no es un factor tan importante, pero al contar con instancias "grandes"de 200 o más clientes el tiempo empieza a jugar un papel fundamental a la hora de ajustar parámetros.

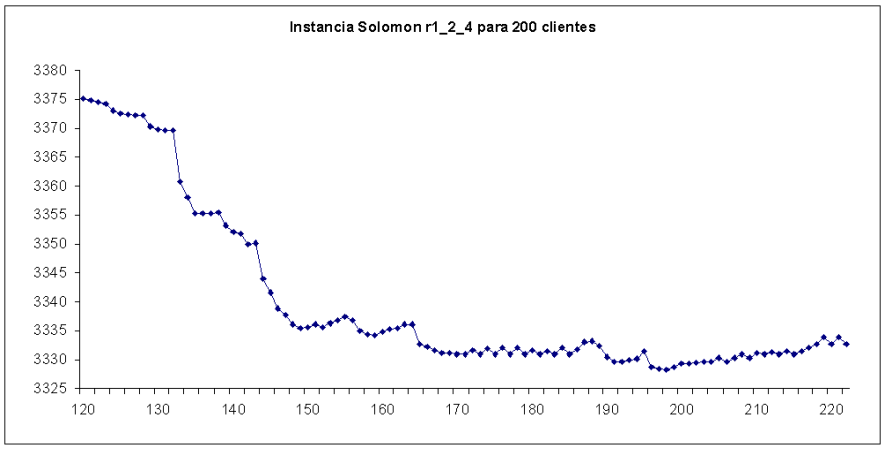
La siguiente tabla compara el tiempo transcurrido hasta que el algoritmo logra encontrar la mejor solución con diferentes parámetros de ejecución. En la tabla se nota claramente que todos los tiempos de ejecución con parámetros de prioridad, diversificación e intensificación son significativamente menores.

Parámetros	r1_2_1	r2_2_1	c1_2_1	c2_2_1	rc1_2_1	rc2_2_1
GTSxD β=2	2:32	5:26	1:38	3:00	2:29	7:15
GTSxD β=2 con Prio. más Div. e Int.	2:20	1:51	0:27	1:45	0:24	1:06
GTSxC C=30	2:59	7:17	0:56	1:47	1:55	4:13
GTSxC C=30 con Prio. más Div. e Int.	0:53	0:44	0:08	0:58	0:18	0:43

Para ejemplificar el comportamiento de la diversificación e intensificación propuesta en los problemas de ruteo, presentamos a continuación un gráfico de la cantidad de vecinos de las soluciones generadas en cada iteración y la evolución de estos valores. Analizando el gráfico, notamos que se pueden identificar 3 etapas. En la primera etapa el algoritmo intensifica disminuyendo el valor de la cantidad de vecinos. En la segunda etapa se nota que la cantidad de vecinos se mantiene constante o a lo sumo intensifica o diversifica levemente. Es decir, el algoritmo va encontrando mejores soluciones entrando y saliendo de mínimos locales.



El siguiente gráfico ilustra los valores de la segunda etapa que el algoritmo va obteniendo en función de la iteración en la que se encuentra.



Finalmente en la tercera etapa el algoritmo diversifica intentando aumentando la cantidad de clientes con el objetivo de intentar salir del mínimo local encontrado hasta ese instante. El primer gráfico nos ayuda a entender que el valor inicial de GTS debe ser tal que la primera etapa concluya obteniendo una cantidad de vecinos que permitan al Tabu Search encontrar una buena solución durante la segunda etapa. En algunos casos, observamos que en la tercera etapa se mejora la solución encontrada en la segunda etapa.

A continuación se muestran los mejores valores encontrados para todas

las instancias de 200 clientes utilizando prioridades más diversificación e intensificación. Los valores de la columna Mejor VRPTW se obtuvieron en *Vehicle Routing and Travelling Salesperson Problems* [20].

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia	Parámetros
r1_2_1.txt	5024.65	19	**4905.99**	28	**-2.36**	GTSxC C=30
r1_2_2.txt	4040.6	18	4192.12	24	3.75	GTSxC C=30
r1_2_3.txt	3381.96	18	3612.21	21	6.81	GTSxD β=2
r1_2_4.txt	3059.22	18	3308.93	19	8.16	GTSxC C=30
r1_2_5.txt	4107.86	18	4299.31	24	4.66	GTSxD β=2
r1_2_6.txt	3583.14	18	3777.58	22	5.43	GTSxD β=2
r1_2_7.txt	3150.11	18	3397.88	20	7.87	GTSxD β=2
r1_2_8.txt	2952.65	18	3137.93	19	6.28	GTSxC C=30
r1_2_9.txt	3760.58	18	3990	21	6.1	GTSxD β=2
r1_210.txt	3301.18	18	3653.66	20	10.68	GTSxC C=30
c1_2_1.txt	2704.57	20	2707.85	20	0.12	GTSxC C=30
c1_2_2.txt	2917.89	18	**2701.4**	20	**-7.42**	GTSxD β=2
c1_2_3.txt	2707.35	18	2720.66	20	0.49	GTSxD β=2
c1_2_4.txt	2643.31	18	2832.11	19	7.14	GTSxC C=30
c1_2_5.txt	2702.05	20	2728.85	20	0.99	GTSxC C=30
c1_2_6.txt	2701.04	20	2701.2	20	0.01	GTSxD β=2
c1_2_7.txt	2701.04	20	2702.48	20	0.05	GTSxC C=30
c1_2_8.txt	2769.19	18	2828.13	21	2.13	GTSxD β=2
c1_2_9.txt	2642.82	18	2755.89	20	4.28	GTSxD β=2
c1_210.txt	2643.51	18	2834.15	21	7.21	GTSxD β=2
rc1_2_1.txt	3602.8	18	3816.38	22	5.93	GTSxD β=2
rc1_2_2.txt	3249.5	18	3370.56	21	3.73	GTSxD β=2
rc1_2_3.txt	3008.33	18	3280.58	19	9.05	GTSxC C=30
rc1_2_4.txt	2852.62	18	3159.17	19	10.75	GTSxC C=30
rc1_2_5.txt	3385.88	18	3698.07	21	9.22	GTSxD β=2
rc1_2_6.txt	3324.8	18	3569.42	21	7.36	GTSxD β=2
rc1_2_7.txt	3189.32	18	3386.97	20	6.2	GTSxC C=30
rc1_2_8.txt	3083.93	18	3433.52	20	11.34	GTSxD β=2
rc1_2_9.txt	3083.41	18	3232.01	19	4.82	GTSxC C=30
rc1_210.txt	3008.53	18	3124.12	19	3.84	GTSxC C=30

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia	Parámetros
r2_2_1.txt	4483.16	4	**3673.4**	13	**-18.06**	GTSxD β=2
r2_2_2.txt	3621.2	4	**3156.86**	11	**-12.82**	GTSxC C=30
r2_2_3.txt	2880.62	4	**2792.8**	10	**-3.05**	GTSxC C=30
r2_2_4.txt	1981.29	4	2324.4	9	17.32	GTSxC C=30
r2_2_5.txt	3366.79	4	3439.01	12	2.15	GTSxD β=2
r2_2_6.txt	2913.03	4	2973.02	11	2.06	GTSxC C=30
r2_2_7.txt	2451.14	4	2496.55	8	1.85	GTSxC C=30
r2_2_8.txt	1849.87	4	2349.66	8	27.02	GTSxC C=30
r2_2_9.txt	3092.53	4	**3046.08**	11	**-1.5**	GTSxC C=30
r2_210.txt	2654.97	4	2818.21	9	6.15	GTSxC C=30
c2_2_1.txt	1931.44	6	1933.64	7	0.11	GTSxC C=30
c2_2_2.txt	1863.16	6	1945.77	8	4.43	GTSxC C=30
c2_2_3.txt	1775.11	6	1912.69	9	7.75	GTSxC C=30
c2_2_4.txt	1703.43	6	1944.95	10	14.18	GTSxC C=30
c2_2_5.txt	1878.85	6	1996.51	9	6.26	GTSxC C=30
c2_2_6.txt	1857.35	6	2086.75	11	12.35	GTSxD β=2
c2_2_7.txt	1849.46	6	2032.45	10	9.89	GTSxC C=30
c2_2_8.txt	1820.53	6	1941.19	9	6.63	GTSxC C=30
c2_2_9.txt	1830.05	6	2060.36	11	12.58	GTSxC C=30
c2_210.txt	1806.6	6	1974.4	9	9.29	GTSxC C=30
rc2_2_1.txt	3099.53	6	**3045.42**	13	**-1.75**	GTSxD β=2
rc2_2_2.txt	2825.24	5	**2719.91**	11	**-3.73**	GTSxC C=30
rc2_2_3.txt	2603.83	4	**2438.18**	9	**-6.36**	GTSxC C=30
rc2_2_4.txt	2043.05	4	2294.08	8	12.29	GTSxC C=30
rc2_2_5.txt	2911.46	4	**2793.25**	11	**-4.06**	GTSxC C=30
rc2_2_6.txt	2975.13	4	**2836.46**	12	**-4.66**	GTSxD β=2
rc2_2_7.txt	2525.83	4	2556.64	10	1.22	GTSxC C=30
rc2_2_8.txt	2293.35	4	2368.57	8	3.28	GTSxC C=30
rc2_2_9.txt	2175.04	4	2461.19	9	13.16	GTSxD β=2
rc2_210.txt	2015.6	4	2304.19	8	14.32	GTSxC C=30

Analizando los valores encontrados por nuestro algoritmo, podemos destacar que en 11 de las 60 instancias se logró mejorar el mejor valor encontrado pero en todos los casos la cantidad de vehículos es superior a la cantidad de vehículos de la mejor solución conocida. En instancias con esta cantidad de clientes empieza a ser notoria la diferencia en el criterio de minimización utilizado.

8.6. Instancias de más de 200 clientes

En esta sección presentamos los resultados obtenidos para las instancias con gran cantidad de clientes. A diferencia de las secciones anteriores, no ejecutamos todas las instancias con las que contamos sino que utilizamos solo un subconjunto muestral. Los valores que obtuvimos los comparamos con los publicados en [20].

Durante las pruebas con instancias de este tamaño notamos que el criterio de parada no favorecía al algoritmo. En las ejecuciones anteriores el algoritmo finalizaba luego de una cierta cantidad de iteraciones en donde no lograba encontrar una solución mejor. En cambio, en las instancias con una gran cantidad de clientes la ejecución llega a un momento en la cual el algoritmo va encontrando mejoras muy leves a expensas de un alto tiempo de ejecución. Por esta razón el criterio de parada utilizado fue el de una cierta cantidad de iteraciones en donde el valor obtenido no superara en un 1 % el valor de la mejor solución encontrada hasta el momento.

A continuación se presentan las pruebas realizadas para las instancias de más de 200 clientes. Para ejecutar estas instancias fijamos los parámetros que a lo largo de la tesis nos parecieron más convenientes, que son los siguientes: GTSxC con C igual al 12 % de la cantidad de clientes de la instancia, con prioridades entre las operaciones utilizando la secuencia 6,1,2,4,5 y con diversificación e intensificación.

La siguiente tabla muestra los resultados obtenidos para instancias de 400 clientes.

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia
r1_4_1.txt	11084	38	11464.11	52	3.43
r2_4_1.txt	9213.68	8	**8144.13**	24	**-11.61**
c1_4_1.txt	7152.02	40	7152.29	40	0
c2_4_1.txt	4116.05	12	4340.24	15	5.45
rc1_4_1.txt	8630.94	36	9459.73	44	9.6
rc2_4_1.txt	6688.31	11	6820.26	23	1.97

La siguiente tabla muestra los resultados obtenidos para instancias de 600 clientes.

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia
r1_6_1.txt	21131.09	59	23840.36	79	12.82
r2_6_1.txt	18291.18	11	**16655.95**	35	**-8.94**
c1_6_1.txt	14095.64	60	14353.2	61	1.83
c2_6_1.txt	7774.1	18	8028.51	20	3.27
rc1_6_1.txt	17317.13	55	18721.87	63	8.11
rc2_6_1.txt	13163.03	15	13271.76	29	0.83

La siguiente tabla muestra los resultados obtenidos para instancias de 800 clientes.

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia
r1_8_1.txt	39612.2	79	41010.09	104	3.53
r2_8_1.txt	28396.87	15	**27611.73**	42	**-2.75**
c1_8_1.txt	25030.36	80	25752.25	82	2.88
c2_8_1.txt	11654.81	24	12274.86	27	5.32
rc1_8_1.txt	35102.79	72	**32880.01**	81	**-6.33**
rc2_8_1.txt	20520.49	19	22134.49	38	7.87

La siguiente tabla muestra los resultados obtenidos para instancias de 1000 clientes.

Instancia	Mejor VRPTW		Nuestro Algoritmo		% diferencia
r110_1.txt	53904.23	100	60863.36	131	12.91
r210_1.txt	42467.87	19	**41072.94**	45	**-3.28**
c110_1.txt	42478.95	100	42479.08	100	0.00
c210_1.txt	16879.24	30	18529.38	35	9.78
rc110_1.txt	47143.9	90	50646.30	99	7.43
rc210_1.txt	29754.06	21	33570.43	48	12.83

Como podemos ver, el algoritmo se comporta muy bien para las instancias de la clase c1, encontrando valores muy similares a los mejores conocidos en cuanto a distancia y cantidad de vehículos. Para el resto de las clases de instancias, en algunos casos el algoritmo encuentra buenos resultados, mejorando el costo pero con una mayor cantidad de vehículos.

9. Conclusiones y Trabajos Futuros

En esta tesis evaluamos el comportamiento de Tabu Search Granular en el problema de ruteo de vehículos con ventanas de tiempo y entregas parciales. El algoritmo implementado se basó en los trabajos de Ho y Haugland [13] y Toth y Vigo [19]. En nuestra versión propusimos dos nuevas operaciones entre vehículos junto a una nueva variante de Tabu Search Granular y modificaciones al Tabu Search que aumentan su performance.

Como vimos en los resultados obtenidos, la variante de Tabu Search Granular introducida por nosotros (GTSxC) consigue muy buenos resultados para una gran cantidad de instancias, superando en muchos casos los valores obtenidos con la variante GTSxD introducida en [19]. Incluso se logró mejorar los mejores valores publicados de algunas pocas instancias. Cabe destacar que en todos los casos el tiempo de ejecución de aplicar alguna variante granular es mucho menor que el tiempo requerido sin el uso de esta técnica. Particularmente en instancias de más de 200 clientes esta diferencia en los tiempos es muy notoria.

En cuanto a nuestras operaciones, las conclusiones son que la operación *Split Client* fue elegida por el algoritmo en muy pocas iteraciones y que la operación *Merge* resultó muy buena: aproximadamente en un 15 % de los casos la operación logró imponerse sobre las otras con porcentajes de mejora que llegaron a ser de hasta el 60 %. El comportamiento observado sobre el resto de las operaciones coincide con el descripto en [13]: la operación *Relocate* fue la que más utilizada por el algoritmo y la operación *Relocate Split* impone condiciones muy restrictivas que impiden que se puedan realizar movimientos de este tipo.

En la evaluación de GTS encontramos resultados que mejoran los publicados. Resumiendo los mejores resultados obtenidos, logramos mejorar 10 de las 56 instancias de 50 clientes, 17 de 56 instancias de 100 clientes y 11 de 60 instancias de 200 clientes. Cabe destacarse que en nuestro caso el objetivo fue minimizar la distancia recorrida y los valores contra los cuales comparamos tenían como primer objetivo minimizar la cantidad de vehículos y luego la distancia.

Algunos posibles trabajos futuros son los siguientes:

- Crear una interfaz gráfica que permita visualizar las rutas resultantes y la evolución de las soluciones.
- Relajar las condiciones de la operación *Relocate Split* de forma tal que sea útil para el algoritmo.
- Cambiar la función objetivo para minimizar primero la cantidad de vehículos y luego la distancia recorrida. Algunas formas posibles de realizar esta tarea son:
  - agregar una penalización por cantidad de vehículos utilizados.
  - modificar y/o diseñar las operaciones en función de buscar minimizar la cantidad de vehículos.
- Evaluar el comportamiento del algoritmo utilizando una lista tabú por cada operación en lugar de una estructura compartida.
- Ajustar el criterio de ejecución con prioridades seleccionando la primer solución vecina que mejore la actual sin necesidad de continuar explorando el vecindario de la solución.
- Completar las pruebas para todas las instancias de Solomon de más de 200 clientes.
- Realizar un análisis de conveniencia combinando GTSxD y GTSxC.
- Evaluar el desempeño de alguna técnica granular en VRPTW.
- Evaluar el desempeño de Tabu Search Granular sobre las instancias de Breedam, Cordeau, Homberger y Russell que se encuentran disponibles en [17].

Bibliografía

[1] Aarts E.H., Lenstra J.K., (1997), "Local Search in Combinatorial Optimization". Wiley, Chichester, UK.

[2] Archetti C., Speranza M.G., Hertz A (2003), "A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem".

[3] Archetti C., Savelsbergh M., Speranza M.G. (2005), "Worst-Case Analysis for Split Delivery Vehicle Routing Problems".

[4] Braysy, O. and Gendreau, M. (2001), "Metaheuristics for the Vehicle Routing Problem with Time Windows", Technical Report STF A01025, SINTED Applied Mathematics, Department of Optimization, Osclo, Norway.

[5] Cordeau J.F., Laporte G. (2002), "Tabu Searh Heuristics for the VRP", Canada Research Chair in Distribution Managment and GERAD Ecole des Hautes Commerciales.

[6] Drop M., Trudeau P. (1989), "Saving by Split delivey routing", Transportation Science 23, 141-145.

[7] Drop M., Trudeau P. (1990), "Split delivery routing", Naval Research Logistics 37, 383-402.

[8] Gendreau M., Giffin J.W. (1994), "A Tabu Search heuristic for the vehicle routing problem", Management Science 40, 1276-1290.

[9] Gendreau M., Laporte G., and Potvin J.-Y. (2002), "Metaheuristics for the VRP". In The Vehicle Routing Problem, P. Toth and D. Vigo (eds) SIAM Monographs on Discrte Mathematics and Applications, Philadelphia, 129-154.

[10] Glover F., Laguna M. (1997), "Tabu Search", Kluwer Academic, Boston.

[11] Golden, B. L. and Assad, A.A, eds(1988), "Vehicle routing: methods and studies", North-Holland, Amsterdam.

[12] Golden, B. L. and Wasil, E.A, eds(1998), "Metaheuristics in Vehicule Routing". In Fleet Management and Logistics, T.G. Crainic and G. Laporte (eds), Kluwer, Boston 33-56.

[13] Ho, Sin C. and Haugland, Dag (2002), "A Tabu Search Heruristic for the Vehicle Routing Problem with Time Windows and Split Deliveries", Department of Informatics, University of Berger, Norway.

[14] Laguna M., (1994), "A Guide to Implementing Tabu Search"

[15] Lenstra, J. K. and Rinnooy Kan, A.H.G. (1981), "Complexity of vehicule routing and scheduling problems", Networks 11, 221-227.

[16] Naddef D. and Rinaldi G.(2002), "Branch and Cut Algorithms for the VRP". In The Vehicle Routing Problem, P. Toth and D. Vigo (eds) SIAM Monographs on Discrte Mathematics and Applications, Philadelphia, 109-138.

[17] The VRP Web, http://neo.lcc.uma.es/radi-aeb/WebVRP/

[18] Toth, P. and Vigo, D., eds(2002), "The vehicle routing problem", SIAM Society for Industrial and Applied Mathematics.

[19] Toth, P. and Vigo, D., "The Tabu Search Granular and its Application to the Vehicle-Routing Problem", INFORMS Journal on Computing 15, No. 4, 333-346.

[20] Vehicle Routing and Travelling Salesperson Problems, http://www.top.sintef.no/vrp/benchmarks.html