Split Delivery Vehicle Routing Problem: Heuristic based Algorithms

Sandro Moscatelli
Instituto de Computación, Facultad de Ingeniería
Universidad de la República
moscatel@fing.edu.uy

Diciembre 2007

Resumen

El *Split Delivery Vehicle Routing Problem* es una relajación del clásico problema de optimización combinatoria *Vehicle Routing Problem*, en el que se excluye la restricción de que cada cliente sea visitado solamente una vez por un único vehículo. En este reporte se presenta una recopilación de algoritmos basados en heurísticas para la resolución de este problema.

Palabras clave: entregas partidas, vehicle routing problem, soluciones óptimas, heurísticas, metaheurísticas.

Abstract

The *Split Delivery Vehicle Routing Problem* is a relaxation of the classical problem of combinatorial optimization *Vehicle Routing Problem*, where the restriction that each customer must be visited only once by a single vehicle is removed. In this report we present a summary of algorithms based on heuristics for the resolution of this problem.

Keywords: split delivery, vehicle routing problem, optimal solutions, heuristics, metaheuristics.

Índice

1. INTRODUCCIÓN

El *VRP* (***V**ehicle **R**outing **P**roblem*) es un problema [1] sumamente conocido y estudiado del área de la optimización combinatoria. En este problema, dado un deposito *d*, una flota de *M* vehículos homogéneos (cada uno con una capacidad *Q*), un conjunto *C* de clientes (donde cada cliente $i \in C$ tiene una demanda d_i, $0 \leq d_i \leq Q$) y siendo c_{ij} el costo de ir del cliente *i* al cliente *j*, se debe construir un conjunto de rutas de costo mínimo que satisfaga las siguientes condiciones:

- cada cliente es visitado exactamente una vez.
- cada ruta comienza y finaliza en el depósito *d*.
- la demanda total de cada ruta no excede la capacidad del vehículo asignado a la misma.

En general se considera como costo la distancia y por lo tanto el costo de una ruta es la distancia recorrida por el vehículo asignado a la misma al visitar los clientes, aunque pueden usarse otras medidas.

El *SVRDP* (***S**plit **D**elivery **V**ehicle **R**outing **P**roblem*) es una relajación del *VRP*, en el cual se elimina la restricción de que cada cliente sea visitado una sola vez por un vehículo, admitiéndose que el mismo puede ser visitado cualquier cantidad de veces por distintos vehículos. Es decir que la demanda del cliente puede ser partida de forma tal que sea satisfecha por varios vehículos (*split delivery*). Además la condición de que cada cliente $i \in C$ tiene una demanda $d_i \leq Q$ no es necesaria en este problema.

Este problema fue planteado e investigado por Dror y Trudeau [2] entre 1989 y 1990, los cuales mostraron que al permitir entregas partidas en la resolución del *VRP* se puede lograr una disminución de la cantidad de rutas y por lo tanto una reducción del costo total de la solución. También mostraron que, a pesar de la relajación introducida, el *SDVRP* es un problema *NP-hard* [2], al igual que lo es el *VRP*.

Este reporte está organizado de la siguiente manera. En la sección 2 se muestran los potenciales beneficios al permitir entregas partidas mediante un ejemplo y se presentan algunas conclusiones en base a estudios empíricos sobre resultados experimentales. En la sección 3 se presentan varias propiedades de las soluciones óptimas del problema *SDVRP*. En la sección 4 se exploran las distintas alternativas y algoritmos existentes actualmente para resolver este problema. En particular se hace énfasis en los algoritmos basados en heurísticas o metaheurísticas considerando que al ser un problema *NP-hard*, en la práctica no siempre es factible utilizar métodos de resolución exactos.
En la última sección se presentan las conclusiones sobre este problema.

2. BENEFICIOS DE ENTREGAS PARTIDAS

Dror y Trudeau [2] se basaron en el ejemplo de la Figura 1 como motivación para su investigación. Este ejemplo muestra que potencialmente se podría usar una cantidad menor de rutas (vehículos) que en la resolución de un *VRP*, lo cual implica una reducción del costo total de las rutas.

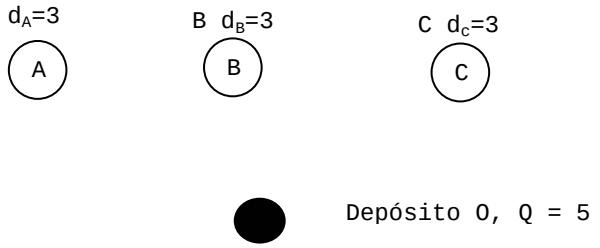


Figura 1 Potencial caso de ahorro resolviendo un *SDVRP*

Dada la matriz de costos $\mathbf{C_{i,j}}$:

	O	**A**	**B**	**C**
O	-	10	10	10
A	10	-	5	10
B	10	5	-	10
C	10	10	10	-

la solución si se resuelve un *VRP*, sin permitir entregas partidas, se muestra en la Figura 2 y tiene un costo de 60 unidades y requiere 3 vehículos (rutas).

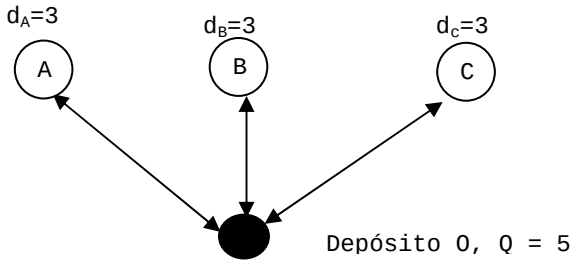


Figura 2 Solución resolviendo *VRP*

en cambio si se permiten entregas partidas, la solución se muestra en la Figura 3, tiene un costo de 50 unidades y requiere de 2 vehículos. Notar que la demanda del cliente *B* es satisfecha por ambos vehículos en partes iguales (2 unidades cada uno).

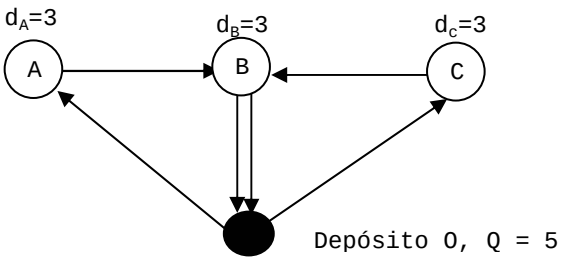


Figura 3 Solución resolviendo un *SDVRP*

Sin embargo Dror y Trudeau no analizaron la existencia de otros casos en los cuales se pudiese obtener una mayor reducción en el costo ni tampoco analizaron en profundidad la cuantía de dicha reducción.

Más recientemente Archetti *et al* [3] mostraron que $\frac{z(VRP)}{z(SDVRP)} \leq 2$ y esta cota es ajustada. Donde *z(VRP)* es el costo de una solución óptima para el problema *VRP* y *z(SDVRP)* es el costo de una solución óptima al problema *SDVRP*.

El resultado anterior significa que existen casos en los cuales la solución *VRP* óptima tiene un costo que es a lo sumo 2 veces mayor que el costo de la solución *SDVRP* óptima, o dicho de otra manera que al permitirse entregas partidas se puede lograr un ahorro de hasta el 50% del costo total.

A pesar del interés teórico que tiene el resultado anterior, en la práctica su utilidad es limitada, dado que no establece ninguna relación entre las características de un problema específico, tales como la distribución geográfica de los clientes y la distribución de la demanda de los mismos y la eventual reducción en los costos si se permiten entregas partidas.

Archetti *et al* [4] enfocan su análisis no sólo en el cociente anterior, sino también en la reducción de la cantidad de rutas requeridas para satisfacer la demanda de los clientes, cuando se permiten entregas partidas considerando que este es el principal beneficio que se obtiene. Para esto último demuestran la siguiente proposición:

$\frac{r(VRP)}{r(SDVRP)} \leq 2$ y la cota es ajustada, siendo *r(VRP)* y *r(SDVRP)* la mínima cantidad de rutas requeridas para satisfacer la demanda de los clientes en una solución del problema *VRP* y del problema *SDVRP* respectivamente. Notar que en ejemplo de la Figura 3 el cociente anterior es 3/2.

También analizan, de forma empírica, los eventuales beneficios que se podrían obtener al permitir entregas partidas estudiando ambos cocientes, basando dicho estudio en ciertas características de los clientes consideradas relevantes, tales como su ubicación geográfica y sus patrones de demanda. En particular para el estudio del cociente $\frac{z(VRP)}{z(SDVRP)}$ se utilizan heurísticas porque al ser problemas *NP-hard* se hace sumamente difícil hallar soluciones exactas para casos de cierto tamaño en tiempos razonables.

El estudio de los cocientes se hace sobre conjuntos de datos generados randómicamente y los autores extraen a las siguientes conclusiones:

- La reducción en el costo que se obtiene al permitir entregas partidas se debe a la posibilidad de reducir la cantidad de rutas (la reducción de la cantidad de rutas tiene como beneficio adicional la necesidad de una flota de vehículos más pequeña)

- Los mayores beneficios se obtienen cuando la demanda promedio (d_p) de los clientes verifica $0.5Q \leq d_p \leq 0.75Q$, siendo Q la capacidad de los vehículos y además la variación en la demanda de los clientes es relativamente pequeña.

- Los beneficios al permitir entregas partidas dependen principalmente de la relación entre la demanda promedio y la capacidad de los vehículos y no parece depender de la ubicación geográfica de los clientes.

3. PROPIEDADES

En esta sección se enumeran algunas propiedades relevantes de las soluciones óptimas, así como las principales contribuciones sobre la complejidad computacional del *SDVRP*.

Considerando la siguiente definición:

> Dado un conjunto de k clientes $\{i_1, i_2, \ldots, i_k\}$ y k rutas $r_1, r_2, \ldots, r_k$ con $k \geq 2$, tal que la ruta r_w contiene los clientes i_w y i_{w+1}, con $w = 1..k-1$ y la ruta r_k contiene los clientes i_k y i_1, se denomina ciclo *k-split* al subconjunto de clientes $i_1, i_2, \ldots, i_k$.

Un ejemplo de ciclo 3-split se muestra en la Figura 4.

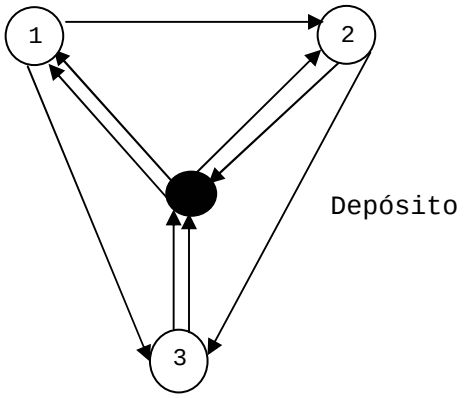


Figura 4 Ejemplo de ciclo 3-split

donde la primer ruta visita a los clientes 1 y 2, la segunda ruta visita a los clientes 2 y 3 y la tercer ruta a los clientes 1 y 3.

Dror y Trudeau [2] demostraron la siguiente propiedad estructural de las soluciones óptimas del *SDVRP*:

> **Propiedad 1**: Si la matriz de costos satisface la desigualdad triangular, entonces existe una solución óptima del *SDVRP* que no tiene ciclos *k-split*, para cualquier *k*.

Como corolario se deduce que si la matriz de costos satisface la desigualdad triangular entonces existe una solución óptima del *SDVRP* en la cual todo par de rutas tiene a lo sumo un cliente en común (*split delivery*).

Archetti *et al* [3] demostraron otra propiedad estructural de las soluciones óptimas del *SDVRP* en la cual relacionan la cantidad de entregas partidas con la cantidad de rutas:

> **Propiedad 2**: Si la matriz de costos satisface la desigualdad triangular, entonces existe una solución óptima del *SDVRP* en la cual la cantidad total de entregas partidas (suma de las entregas partidas de todos los clientes) es menor que la cantidad de rutas.

Archetti *et al* [5] analizan el caso en el cual la capacidad de los vehículos es $Q \in Z^+$, $Q \geq 2$ y la demanda de los clientes es un valor entero, eventualmente mayor que la capacidad de los vehículos. A modo de ejemplo, esta situación puede darse cuando se entregan contenedores.

Definen la propiedad de reducibilidad de una instancia del problema *SDVRP*, con las características anteriores, como: una instancia de *SDVRP* es reducible si existe una solución óptima en la cual cada cliente con una demanda mayor o igual a *Q* es servido por tantas rutas directas (del depósito al cliente) como es posible, transportando una carga *Q* en cada una, hasta que la demanda remanente del cliente es menor que *Q*.

La reducción se obtiene cambiando la demanda d_i de cada cliente por $d_i \ mod \ Q$ (lo cual representa la realización de *Q* viajes directos entre el depósito y el cliente *i*) y eliminado los clientes con $d_i \ mod \ Q = 0$. La reducción se realiza en un tiempo que es lineal en la cantidad de clientes.

En los casos $Q=2$ y $Q=3$ prueban que el problema es reducible a un nuevo problema en el cual la demanda de cada cliente es menor que la capacidad del vehículo si se cumplen ciertas restricciones sobre los costos. Para *Q* mayores no queda determinado si el problema es reducible o no.

Cuando $Q=2$, si los costos son simétricos y satisfacen la propiedad de la desigualdad triangular, el problema es reducible a un nuevo problema en el cual la demanda de cada cliente es 1. En el caso más general de costos asimétricos esto no se cumple.

Cuando $Q=3$, si los costos son simétricos y satisfacen la desigualdad triangular generalizada con $\alpha=2/3$ (el costo satisface la desigualdad triangular generalizada para algún $\alpha \in (0,1]$ cuando $c_{ij} \leq \alpha \ (c_{ik} + c_{kj}) \ \forall \ i,j,k, \ i \neq j \neq k$), el problema es reducible a un nuevo problema en el cual la demanda de cada cliente es 1 ó 2. Notar que si se considera como costo la distancia euclidiana el problema no es reducible.

Claramente la reducción implica una simplificación del problema y una eventual disminución en la cantidad de clientes. Notar que para $Q=2$ el problema reducido es un caso particular del *VRP* porque cada cliente debe ser visitado una sola vez. Esto implica que podrían aplicarse los algoritmos conocidos para su resolución.
Lo anterior no es aplicable cuando $Q=3$, porque aún en el problema reducido un cliente podría ser visitado más de una vez.

Además, en el caso particular de $Q=2$ demuestran que el problema es resoluble en tiempo polinomial, si los costos entre clientes (incluyendo el depósito) son simétricos ó bien si los costos satisfacen la propiedad de la desigualdad triangular.

Archetti *et al* [6] plantean una formulación matemática para el problema *SDVRP*, en la cual realizan los siguientes supuestos:

- los costos entre clientes (incluyendo el depósito) son no negativos y satisfacen la desigualdad triangular.
- La demanda d_i de cada cliente es un valor entero.
- La capacidad de cada vehículo es $Q \in Z^+$

y demuestran que si el problema tiene soluciones factibles, entonces existe una solución óptima en la cual la cantidad entregada por cada vehículo cuando visita un cliente es un valor entero.

4. ALTERNATIVAS DE SOLUCIÓN

Analizando la literatura disponible se encuentran, básicamente, los siguientes enfoques para resolver el *SDVRP*:

- Métodos de búsqueda de soluciones exactas.
- Algoritmos basados en la búsqueda de soluciones aproximadas.
- Combinaciones de los dos métodos anteriores.

4.1 Soluciones exactas

Esta alternativa tiene el inconveniente de que, a pesar de los progresos realizados a lo largo del tiempo, sigue siendo un gran desafío resolver de forma óptima problemas *NP-hard*. Si bien la búsqueda de soluciones exactas tiene valor desde el punto de vista teórico, los problemas que pueden ser resueltos son generalmente pequeños y por lo tanto estos métodos no son aplicables en la práctica.

Dror *et al* [7] propusieron en 1994 una formulación mediante programación lineal entera del *SDVRP*, a partir de la cual derivaron un conjunto de desigualdades válidas que fueron utilizadas en un algoritmo del tipo *branch and bound* (en particular *cutting plane*) usado para resolver óptimamente instancias de a lo sumo 20 clientes.

Sierksma *et al* [8] presentaron en 1998 una aplicación real de *SDVRP* para encontrar los vuelos de helicópteros hacia plataformas marinas a los efectos de intercambiar las tripulaciones de las mismas. Para resolver esta aplicación realizaron una formulación de programación lineal que fue resuelta mediante la técnica de *column generation*. En su artículo los autores indican que esta forma de solución solamente es aplicable para planificaciones a largo plazo debido a que requiere mucho tiempo de procesamiento.
Para planificaciones de corto plazo proponen una heurística específicamente diseñada para la realidad considerada basada en definir *clusters* de plataformas y varios procedimientos de mejoras.

Belenguer *et al* [9] presentaron en el año 2000 una formulación mediante programación lineal entera del *SDVRP*. Esta formulación se diferencia de la propuesta en [7] debido a que considera que la cantidad de vehículos disponible es la que surge de dividir el total de la demanda de los clientes por la capacidad de los mismos, es decir que la cantidad de vehículos disponibles es fija, mientras que en [7] la cantidad de vehículos no está limitada. A partir de esta formulación derivan

un conjunto de desigualdades válidas que son utilizadas en un algoritmo *cutting plane* para resolver óptimamente distintas instancias, la mayor de las cuales tiene 50 clientes.

Gendrau *et al* [10] presentaron en 2002 un algoritmo exacto para el *SDVRPTW* (*SDVRP* donde los clientes tienen ventanas de tiempo durante las cuales se pueden realizar las entregas) basado en una formulación de tipo *set covering* y un enfoque de *column generation*. El esquema de *column generation* es incluido en un árbol *branch and bound* para obtener un algoritmo *branch and price* exacto. Este algoritmo es usado para encontrar la solución óptima a instancias de tamaño medio (de entre 50 y 100 clientes).

Lee *et al* [11] formulan el *SDVRP* como un problema de programación dinámica y sus pruebas computacionales muestran que este método de solución es capaz de resolver instancias sumamente pequeñas en tiempos razonables.

Liu [12] en 2005 propone un algoritmo de dos etapas para resolver exactamente el *SDVRP* considerando que la cantidad de vehículos es fija.

En la primera etapa se resuelve un problema de asignación para determinar *clusters* de clientes a ser atendidos por el mismo vehículo. Esta etapa permite obtener una cota inferior de la solución del problema.

En la segunda etapa se resuelve un *TSP* (***T****ravelling* ***S****alesman* ***P****roblem*) para cada *cluster*, determinando una cota superior. Esta cota superior es utilizada para generar desigualdades válidas que son usadas en la en la siguiente iteración para la resolución de la primer etapa.

El algoritmo continúa iterando hasta que la cota inferior y superior coincidan.

En el caso que la cantidad de vehículos sea variable propone un algoritmo basado en el enfoque *branch and price*.

4.2 Soluciones aproximadas

En esta sección se presentan los principales algoritmos de solución del problema *SDVRP* basados en heurísticas y metaheurísticas.

4.2.1 Two Stages Algorithm

Dror y Trudeau [13] propusieron un algoritmo para resolver el *SDVRP*, denominado *Two Stages Algorithm* que consiste básicamente en un algoritmo de búsqueda local y que contempla solamente el caso en el cual la demanda de cada cliente es menor que las capacidades de los vehículos.

1. Generar una solución inicial
2. Node interchange
3. Routes improvement
4. Setear *split_improvement*=false y *add_improvement*=false
5. k-split Interchanges. Ejecutar todos los k-split interchanges. Si hay por lo menos una mejora entonces *split_improvement*=true
6. Route Addition. Ejecutar todas las mejoras de Route Addition. Si hay por lo menos una mejora entonces *add_improvement*=true
7. Si *add_improvement*=true ir al paso 5, en caso contrario si *split_improvement*=true ir al paso 2, sino parar

Figura 5 *Two Stages Algorithm*

El algoritmo de la Figura 5 utiliza los siguientes algoritmos:

- Generación de solución inicial
- Node interchange
- Routes improvement (2-opt)
- k-split interchanges
- Route Addition

Estos algoritmos están agrupados en dos fases. En la primer fase se construye una solución inicial resolviendo un *VRP* y sobre la misma se aplican algoritmos de mejoras (Node interchange y Routes improvement). La segunda fase genera y/o elimina entregas partidas analizando si se mejora el costo de la solución.

En algoritmo para generar una solución inicial construye una solución factible resolviendo un *VRP* mediante una variante del algoritmo de Clarke y Wright [14].

El algoritmo Node interchange realiza intercambios de cadenas de 1 y 2 clientes entre las rutas.

El algoritmo Routes improvement aplica a cada ruta el algoritmo 2-opt [15].

Para mostrar en que consiste el algoritmo k-split Interchanges se explica, a continuación, el *2-split interchange* y luego se lo generaliza para $k \geq 3$.

Sean las rutas r_1 y r_2 tales que la capacidad remanente (s_1 y s_2 respectivamente) de los vehículos de cada una de ellas es positiva.

Sea un cliente *p* de la ruta r_3 al cual se le entrega (en esta ruta) una demanda d_{p3} y tal que

$$s_1 + s_2 \geq d_{p3}. \qquad (\mathbf{1})$$

Sean i_1 y j_1 dos clientes consecutivos de la ruta r_1 y i_2 y j_2 dos clientes consecutivos de la ruta r_2. Sea b_p el cliente inmediatamente anterior al cliente *p* en r_3 y a_p el cliente inmediatamente posterior al cliente *p* en r_3.

El ahorro que se obtiene si la demanda d_{p3} es dividida en las rutas r_1 y r_2 (es decir si el cliente *p* es eliminado de la ruta r_3 e insertado entre los clientes i_1 y j_1 de la ruta r_1 y entre los clientes i_2 y j_2 de la ruta r_2) es:

$$\text{SAV}(p) = Ci_1,j_1 + Ci_2,j_2 + Cb_p,p + Cp,a_p - Ci_1,p - Cp,j_1 - Ci_2,p - Cp,j_2 - Cb_p,a_p \qquad (\mathbf{2})$$

Si para algún cliente *p* se satisface (**1**) y (**2**) es positivo, entonces el costo total puede ser reducido (en SAV unidades) dividiendo la demanda d_{p3} entre las rutas r_1 y r_2.

Notar que no sirve dividir la demanda d_{p3} entre la ruta r_3 y una de las rutas r_1 o r_2, siempre y cuando la matriz de costos $C_{i,j}$ verifique la desigualdad triangular.

En definitiva lo que se intenta es dividir la demanda de un cliente entre *k* rutas, manteniendo las restricciones de capacidad y si se obtiene una mejora en el costo. Por ejemplo dada la situación de la Figura 6:

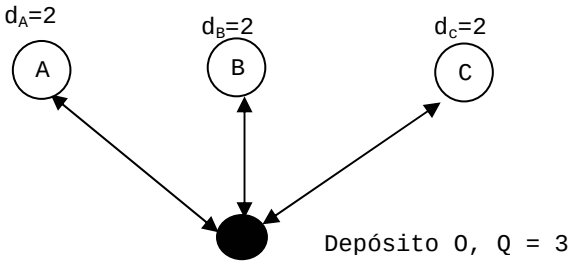


Figura 6 Ruteo usando 3 vehículos

un posible *2-split interchange* es el que se muestra en la Figura 7.

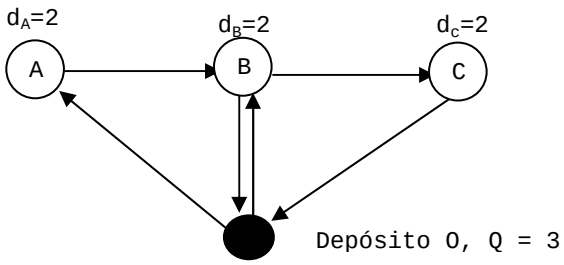


Figura 7 *2-split interchange*

en el cual la demanda del cliente *2* es satisfecha por dos vehículos, cada uno de los cuales entrega una unidad.

La generalización del *2-split interchange* a *k-split interchange*, para $k \geq 3$ es la siguiente:

$SAV(p) = \sum_{t=1}^{k} (Ci_t,j_t - Ci_t,p - Cp,j_t) + Cb_p,p + Cp,a_p - Cb_p,a_p$ (**3**)

si $s_1 + s_2 + ... + s_k \geq d_{p(k+1)}$ y SAV(*p*) es positivo, entonces el cliente *p* puede ser eliminado de la ruta *k+1* y el costo total ser reducido.

El algoritmo *k-split interchange* se muestra en Figura 8.

```
Para cada cliente p con demanda total d_i
      Remover p de todas las rutas en las que es visitado
      Considerar todos los subconjuntos R de rutas cuya suma de
      capacidades remanentes es mayor o igual que d_i
      Seleccionar el subconjunto R que tenga el menor costo de inserción
      de p aplicando la ecuación (3).
      Insertar p en todas las rutas de R, considerando las rutas de
      forma ordenada por capacidad remanente de menor a mayor. Si la
      demanda no satisfecha del cliente es mayor que la capacidad
      remanente de la ruta, se entrega la capacidad remanente de la
      ruta.
```

Figura 8 Algoritmo *k-split interchanges*

En algunos casos el agregado de una ruta (algoritmo `Route Addition`) que elimina una entrega partida puede reducir el costo total.

Sea un cliente *h* que aparece en 2 rutas r_1 y r_2 por lo menos. Se elimina el cliente *h* de las rutas y se analizan las configuraciones formadas por las rutas que cumplen las siguientes propiedades:

- preservan los segmentos de ruta (desde el depósito al cliente anterior al cliente *h* en las rutas r_1 y r_2 y desde el cliente siguiente al cliente *h* al depósito en las rutas r_1 y r_2)
- para cada ruta que no visita al cliente *h* se unen los segmentos correspondientes.

por ejemplo para el caso que se muestra en la Figura 9.

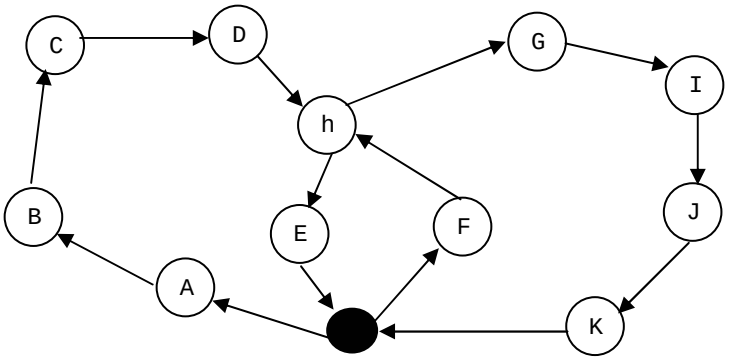


Figura 9 Ruteo con entregas partidas

tres de las posibles configuraciones que cumplen las propiedades anteriores se muestran en la Figura 10, Figura 11 y Figura 12

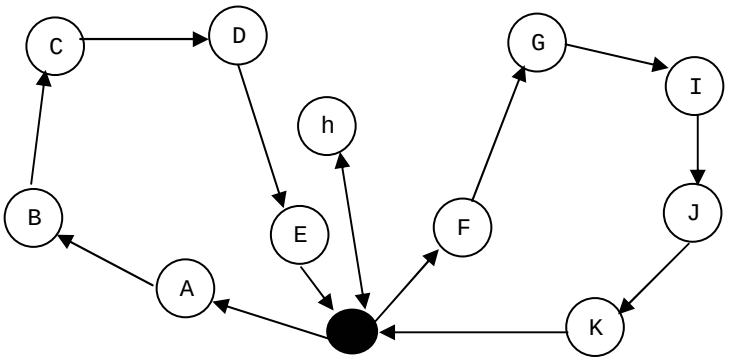


Figura 10 Primer configuración que elimina la entrega partida

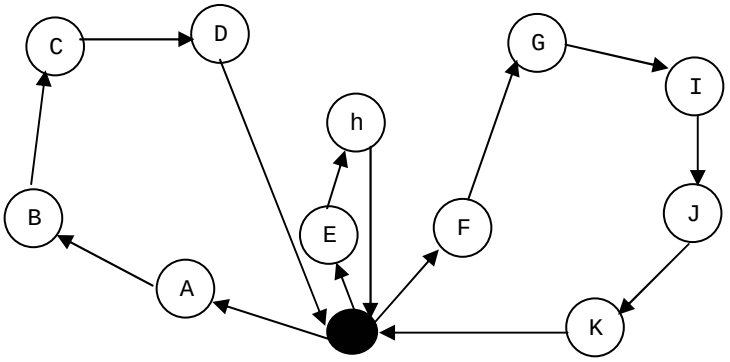


Figura 11 Segunda configuración que elimina la entrega partida

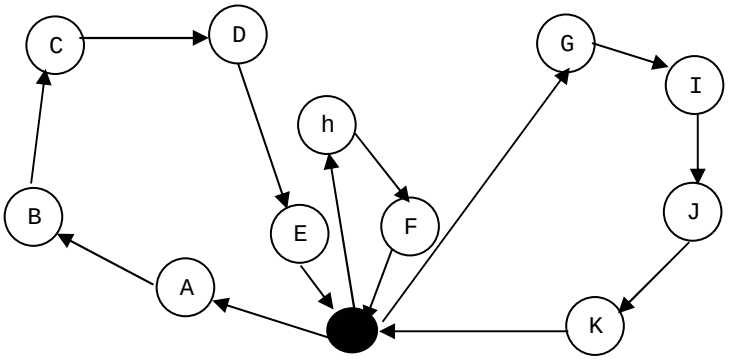


Figura 12 Tercer configuración que elimina la entrega partida

Puede observarse que en el caso que un cliente se encuentre en *k* rutas la cantidad de configuraciones diferentes agregando una ruta para eliminar la entrega partida es $2k^2 + 1$.

Cuando $k=2$, la cantidad de configuraciones posibles es 9, mientras que cuando un cliente se encuentra en 3 rutas la cantidad de configuraciones a analizar es 19.

Debido a que la cantidad de configuraciones crece rápidamente a medida que un cliente se encuentra en una mayor cantidad de rutas, los autores restringen el algoritmo de `Route`

Adition a analizar solamente las configuraciones cuando un cliente se encuentra en 2 o 3 rutas.

En el caso que un cliente se encuentre en más de 3 rutas entonces se analizan las configuraciones resultantes de cada subconjunto de 2 y 3 rutas respectivamente.

El algoritmo presentado fue la primera propuesta para resolver el *SDVRP* y sobre el mismo se pueden realizar los siguientes comentarios:

- Para hallar la solución inicial se plantea resolver un *VRP* usando una variante del algoritmo de Clarke & Wright. Esto es posible porque se asume que la demanda de cada cliente es menor que la capacidad de los vehículos. Sin embargo no realizan un análisis de cual es el impacto de esta solución inicial en la calidad de la solución final.

- Si bien el algoritmo es una heurística y por lo tanto no necesariamente debe encontrar la solución óptima, hay casos sumamente simples en los cuales falla y no encuentra la solución óptima, un ejemplo de esto es el que se muestra en la
- Figura 13 [5], en el cual la capacidad de los vehículos es $Q=4$, las distancias son simétricas y satisfacen la desigualdad triangular:

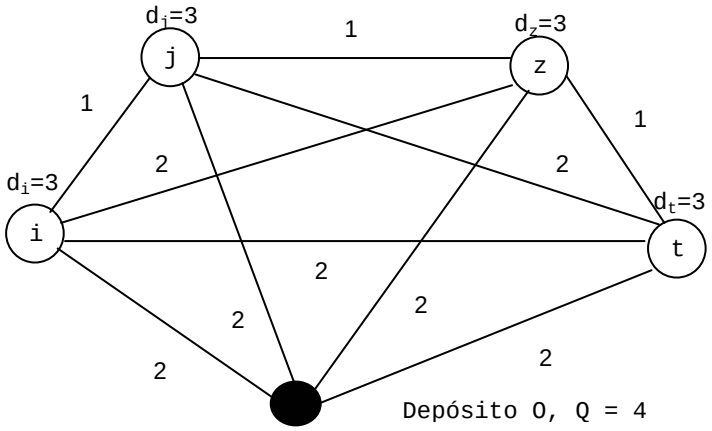


Figura 13 Caso donde *Two Stages Algorithm* no encuentra la solución óptima

La única solución factible, sin entregas partidas, es decir resolviendo un *VRP* se muestra en la Figura 14 y tiene un costo de 16.

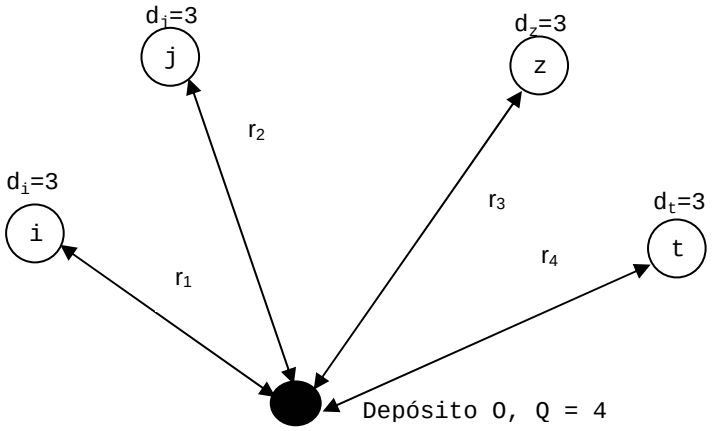


Figura 14 Solución resolviendo un *VRP*

Las rutas de la Figura 14 constituyen la solución inicial para el algoritmo de Dror y Trudeau. Notar que los algoritmos `Node Interchange` y `Routes Improvement` no pueden mejorar esta solución.

Como la demanda de cada cliente es 3 y la capacidad remanente de cada vehículo es 1, el único *k-split interchange* factible consiste en dividir la demanda de un cliente entre las 3 rutas restantes, como se muestra en la Figura 15.

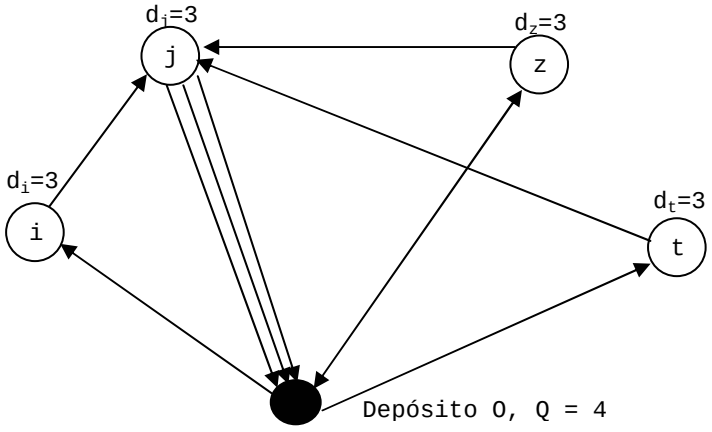


Figura 15 Demanda partida de un cliente

En la Figura 15 la demanda del cliente *j* es dividida en las rutas r_1, r_3 y r_4. Esta división produce una solución de costo 16 (tanto si la demanda del cliente *j* ó *z* es dividida) o de 17 (si la demanda de los clientes *i* ó *t* es dividida).

Por lo tanto no se encuentra ninguna mejora mediante el algoritmo `k-split interchanges` y por lo tanto se ejecuta el procedimiento `Route Addition` con la solución inicial (*VRP*).

Como no hay ningún cliente con entrega en más de una ruta este algoritmo no puede realizar ningún cambio y por lo tanto finaliza sin encontrar la solución óptima que es la que se muestra en la Figura 16 y que tiene un costo de 15.

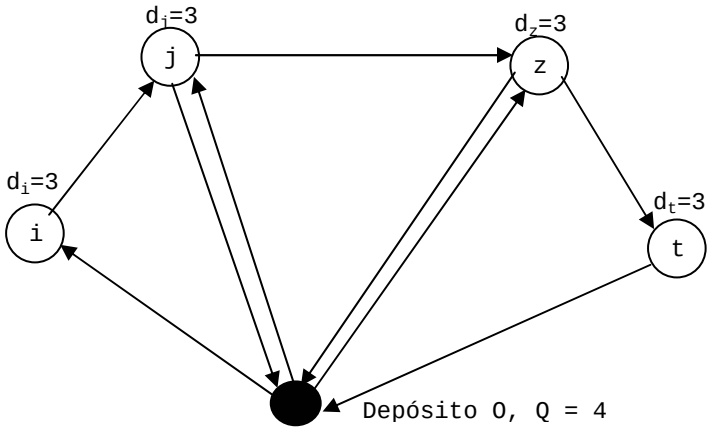


Figura 16 Solución óptima

El algoritmo *Two Stages Algorithm* fue testeado en problemas de 75, 115 y 150 clientes, con una flota homogénea de vehículos de capacidad 160 unidades y generando la demanda de cada cliente de forma randómica entre varios escenarios, en cada uno de los cuales la demanda máxima y mínima es una fracción de la capacidad de los vehículos. Se compararon los resultados con la resolución de un *VRP* y de los resultados obtenidos los autores observaron que al resolver el problema *SDVRP*:

- Se disminuye la cantidad de vehículos utilizados y por lo tanto el costo de la misma.
- El tiempo requerido para resolver cada problema es mayor que al resolver un *VRP*.
- Cuando la demanda de los clientes es muy pequeña, comparada con la capacidad de los vehículos, no hay (o si la hay es muy pequeño) ahorros frente al *VRP*.

4.2.2 Splitabú

Archetti *et al* [6] propusieron un algoritmo basado en la metaheurística *Tabú Search*, propuesta por Glover *et al* [16] para resolver el *SDVRP* el cual se denomina *Splitabú* .Este algoritmo consta de tres etapas y se muestra en la Figura 17

```
1. Construcción de una solución inicial factible
2. Tabu Search
3. Mejora de la solución
```

Figura 17 Algoritmo *Splitabú*

En la etapa de construcción de la solución inicial (Figura 18) se crea una instancia reducida del problema en la cual la demanda de cada cliente es menor que la capacidad *Q* de los vehículos. Sobre esta instancia reducida se resuelve un *TSP* y finalmente se “corta” la ruta generada para satisfacer las restricciones de capacidad.
Para resolver el *TSP* se utiliza el algoritmo *Genius* [17] el cual está compuesto por dos procedimientos: el primero es un procedimiento de inserción generalizado y el segundo es un algoritmo de post optimización.

1. Crear $\lfloor d_i/Q \rfloor$ viajes directos para cada cliente i y considerar la instancia reducida del problema I_r en la cual cada cliente tiene una demanda igual a d_i-Q* $\lfloor d_i/Q \rfloor$. Remover los clientes sin demanda de I_r
2. Construir una ruta T, resolviendo un *TSP* mediante el algoritmo Genius, para I_r
3. Elegir una orientación para la ruta T y etiquetar los clientes $T = 0, v_1, v_2, \ldots, v_{b-1}, 0$ donde 0 es el depósito.
4. Si la demanda total de T es menor o igual que Q entonces terminar. En caso contrario determinar el menor índice i tal que la demanda en T, hasta i, es mayor que Q. Construir una ruta $0, v_1, .., v_{i-1}, 0$ y considerar $T = 0, v_i, \ldots, v_{b-1}, 0$ e ir al paso 3

Figura 18 Construcción de la solución inicial de *Splitabú*

En la etapa de *Tabú Search* se utilizan dos algoritmos `Order Routes` y `Best Neighbour` que son invocados por el algoritmo principal.

El algoritmo `Order Routes` (Figura 19) recibe un cliente y construye una lista ordenada de rutas que visitan al cliente. El orden de esta lista está dado por el ahorro que se obtendría al eliminar el cliente.

1. Determinar el conjunto U_i de rutas que visitan al cliente i
2. Para cada ruta $u \in U_i$ calcular $s_u = c_{pi} + c_{iq} - c_{pq}$, donde p y q son el predecesor y sucesor del cliente i en u respectivamente
3. Ordenar las rutas de U_i de forma descendente por el valor s_u y setear O_i como esta lista ordenada

Figura 19 Algoritmo `Order Routes`

En esta etapa se consideran movidas de una solución s a una vecindad s´ aquellas que insertan un cliente *i* en una ruta *r* y eliminan el cliente de un subconjunto $U \subseteq O_i - \{r\}$ de rutas que lo visitan, donde U se determina según la lista ordenada que se obtiene a partir de `Order Routes`.

Cuando un cliente *i* es insertado en una ruta *r* se considera tabú eliminar al cliente *i* de la ruta *r* por θ iteraciones y también se considera que la ruta *r* es tabú para el cliente *i*.

Cuando un cliente *i* es eliminado de una ruta *u* se considera tabú volver a insertar al cliente *i* en la ruta *u* por θ iteraciones y también se considera que la ruta *u* es tabú para el cliente *i*.

Como las restricciones tabú pueden llegar a ser muy estrictas y por lo tanto no permitir obtener una buena vecindad, se considera la posibilidad de eliminar e insertar un cliente *i* de rutas que son tabú para dicho cliente. Las vecindades que se obtienen con estas movidas solo son aceptadas si conducen a una mejor solución que la mejor solución encontrada hasta el momento (criterio de aspiración).

Para explicar el algoritmo Best Neighbour (Figura 20) se usa la siguiente notación:

- d_{ir} la cantidad entregada al cliente *i* en la ruta *r*
- p_r la capacidad remanente de la ruta *r*
- f(*s*) el costo de una solución *s*
- R conjunto de todas las rutas de una solución más una nueva ruta (que no visita a ningún cliente)
- U_{iT}= {u ∈ O_i | u es tabú para *i*}

Best Neighbour (i)
Entrada:
Solución *s*
Mejor solución encontrada *s**
Salida:
Una vecindad s_i para el cliente *i*
Una ruta *r**
Un subconjunto de rutas U* ⊆ O_i-{r*}

1. BestValue = ∞, O_i = Order Routes(*i*)
2. Para cada ruta *r* en *R*
   2.1 Eliminar de rutas no tabú e insertar en rutas no tabu
   1. Si *r* no es tabu para *i* entonces p = p_r, U = ∅ sino ir a 2.2
   2. Considerar todas las rutas *u* ∈ O_i -{*r*} según el orden definido en O_i. Si *u* no es tabu para *i* y d_{iu} < p entonces U = U ∪ {*u*}, p = p - d_{iu}
   3. Si U = ∅ entonces F = ∞, sino sea *s´* la solución que se obtiene a partir de *s* eliminando *i* de todas las rutas de U e insertando *i* en *r*, d_{ir} = d_{ir} + Σ(*u* ∈ U) d_{iu}, F = f(*s´*)
   4. Si U = ∅, sea *u* la primer ruta no tabu para *i* en O_i - {*r*}, U = {*u*} y considerar la solución *s´* que se obtiene insertando *i* en *r*, d_{iu} = d_{iu} - p, d_{ir} = d_{ir} + p, F = f(*s´*)
   5. Si F < BesValue entonces U* = U, *r** = *r*, BestValue = F, s_i = *s´*

   2.2 Eliminar de rutas tabu y/o insertar en rutas tabu
   1. Si *r* es tabu para *i* ó U_{iT} contiene una ruta *r* con d_{iu} ≤ p_r entonces p = p_r y U = ∅ sino volver a 2.1 con la próxima ruta *r* ∈ R. Si *r* es tabu para *i* entonces U = ∅ sino determinar la primer ruta *u* en U_{iT} (según el ordenamiento en O_i) tal que d_{iu} ≤ p_r y U = U ∪ {*u*}, p = p_r - d_{iu}
   2. Considerar todas las rutas *u* ∈ O_i - (U ∪ {*r*}) según el orden definido en O_i. Si d_{iu} < p entonces U = U ∪ {*u*}, p = p - d_{iu}
   3. Si U = ∅ entonces F = ∞, sino sea *s´* la solución que se obtiene a partir de *s* eliminando *i* de todas las rutas en U e insertando *i* en *r*, d_{ir} = d_{ir} + Σ(*u* ∈ U) d_{iu}, F = f(*s´*)
   4. Si F < BesValue y F < f(*s**) entonces U* = U, *r** = *r*, BestValue = F, s_i = *s´*
   5. Volver a 2.1 con la próxima ruta *r* ∈ R

Figura 20 Algoritmo Best Neighbour

El algoritmo Best Neighbour es invocado por el algoritmo principal de esta etapa para cada cliente y determina para cada uno de ellos una vecindad candidata. Se realiza una movida de la solución actual a la mejor vecindad entre las candidatas.

El algoritmo principal de esta etapa es un algoritmo estándar de *Tabú Search* (Figura 21) que finaliza cuando se alcanzan n_{max} iteraciones sin mejorar la mejor solución encontrada hasta el momento. Según las pruebas realizadas por los autores el valor recomendado para n_{max} es 400*n*, siendo *n* la cantidad de clientes.

1. Sea *s* la solución inicial generada (generada por la primer etapa)
2. *s** = *s*, count = 0, Best = ∞
   Para i=1 hasta n
   Best Neighbour(i)
   Si f(s_i) < Best entonces
   BestI = i, BestU = U*, Bestr = *r**, BestS = s_i,
   BestF = f(s_i)
3. *s* = BestS
   Considerar Bestr y todas las rutas de BestU como tabú para BestI durante θ iteraciones
   Si BestF < f(*s**) entonces *s** = *s*, count = 0 sino count = cont + 1
   Si count < n_{max} ir a 2 sino parar

Figura 21 Algoritmo *Tabú Search*

Según las pruebas realizadas por los autores los valores de θ que dependen de la cantidad *n* de clientes y de la cantidad *g* de rutas en la solución actual producen mejores soluciones. Por lo tanto recomiendan elegir θ como un valor randómico en el intervalo $[\sqrt{10}[,]\sqrt{(10+p)}]$, siendo $p = n + g$ si $n + g < 100$ y en caso contrario $p = 3/2(n + g)$.

En la última etapa se intenta mejorar la solución determinada en la etapa de *Tabú Search* eliminando los ciclos *k-split* (Figura 22).

De acuerdo a lo visto en la sección Propiedades, si el costo satisface la propiedad de desigualdad triangular entonces existe una solución óptima que no contiene ciclos *k-split*. En base a esto se intenta remover los ciclos *k-split* de la siguiente forma:

Suponiendo que existe un ciclo *k-split* que involucra a las rutas $r_1, r_2, ..., r_k$, tal que la ruta r_w contiene los clientes i_w e i_{w+1}, $w = 1..k-1$ y la ruta r_k contiene los clientes i_k y i_1, sea w* un índice tal que $d_{iw*rw*} \leq d_{iwrw}$, $w = 1..k$: se puede transferir d_{iw*rw*} unidades de la demanda de cada cliente i_w, $w = 1..k-1$ de la ruta r_w a la ruta r_{w+1}, así como la misma cantidad para el cliente i_k de la ruta r_k a la ruta r_1.
El cliente i_{w*} puede ser eliminado de la ruta r_{w*}. Si se satisface la propiedad de la desigualdad triangular esta nueva solución posiblemente sea mejor que la que contiene el ciclo.

Finalmente se aplica el algoritmo *Genius* a cada ruta.

1. Sea *s* la solución resultante de la etapa *Tabú Search*. Si los costos satisfacen la propiedad de la desigualdad triangular y hay ciclos *k-split* eliminarlos de *s*.
2. Aplicar a cada ruta el algoritmo *Genius*.

Figura 22 Algoritmo de mejora de la solución

Los autores realizaron distintos testeos de *Splitabú* usando los problemas *benchmark* de *VRP* descriptos en [18] generando la demanda de cada cliente de forma randómica entre varios

escenarios, en cada uno de los cuales la demanda máxima y mínima es una fracción de la capacidad de los vehículos.

Generaron en primer lugar instancias pequeñas de hasta 15 clientes (considerando los primeros clientes de los problemas) que fueron resueltos exactamente y con *Splitabú*, comprobándose que *Splitabú* hallaba la misma solución óptima y en muchísimo menos tiempo.

Para las instancias más grandes se eligieron los problemas 1-5, 11 y 12 de [18] con un rango de 50 hasta 199 clientes y se comparó los resultados de *Splitabú* con el algoritmo de Dror y Trudeau tanto en el costo total de la solución como en el tiempo de ejecución.

Los resultados obtenidos indicaron que se obtuvo una mejora en el costo total de la solución en casi todas las instancias. En cuanto al tiempo de ejecución, el algoritmo de Dror y Trudeau es mucho más rápido que *Splitabú* dado que el máximo tiempo requerido por el algoritmo de Dror y Trudeau fue de 7 segundos, mientras que *Splitabú* necesitó un promedio de 10 minutos y un máximo de más de 30 minutos.

De la observación de los resultados experimentales los autores deducen que algunas soluciones pueden ser mejoradas si se aplican intercambios de clientes entre rutas como parte de las mejoras que se realizan a las soluciones (en el algoritmo *Splitabú* no se aplican). Para esto se propone una variante denominada *Splitabú-DT* que aplica los procedimientos `Node interchange` y `2-opt` cada vez que una mejor solución es encontrada en el paso 3 de la etapa *Tabu Search* (Figura 21).

Al ser testeada esta variante de *Splitabú*, con el mismo conjunto de instancias, los autores verificaron una mejora en el costo total de la solución con respecto al costo encontrado por el algoritmo de Dror y Trudeau, en todos los casos.

Una de las críticas que pueden realizarse tanto a los algoritmos *Splitabú* y *Splitabú-DT* es que si bien tienen una mejor *performance* que el algoritmo de Dror y Trudeau en cuanto al costo total de la solución encontrada, cuando la demanda de los clientes es grande se inducen vecindades también grandes lo cual tiene un efecto directo en el tiempo de ejecución del algoritmo, esto explica las diferencias con los tiempos de ejecución del algoritmo de Dror y Trudeau para las mismas instancias.

Por último, como se vio anteriormente, hay instancias relativamente simples para las cuales el algoritmo de Dror y Trudeau no encuentra la solución óptima (Figura 13) lo cual es solucionado con la vecindad que se define en este algoritmo.

4.2.3 Tabú Search para SDVRPTW

En 2004 Haugland *et al* [19] propusieron una algoritmo basado en *Tabú Search* para resolver el problema **SDVRPTW** el cual consta de tres etapas (Figura 23) y es similar al algoritmo *Splitabú* en su estructura aunque presenta varias diferencias en cada una de las etapas.

1. Construcción de una solución inicial factible
2. Tabu Search
3. Mejora de la solución

Figura 23 *Tabú Search* para *SDVRPTW*

La construcción de la solución inicial se realiza agregando el cliente no ruteado más cercano *j* al último cliente ruteado *i*, teniendo en cuenta el tiempo de viaje y el tiempo de espera. Este proceso se repite hasta que todos los clientes hayan sido ruteados y se muestra en la Figura 24 donde se utiliza la siguiente notación:

- C = {1,2,...,n} es el conjunto de clientes a rutear.
- 0 es el depósito
- $N = C \cup \{0\}$
- Todo par $(i,j) \in N$, $i \neq j$, tiene asociado un valor d_{ij} que representa el costo de viajar de *i* a *j* y un valor t_{ij} que representa el tiempo de viaje de *i* a *j*.
- V es el conjunto de vehículos, cada uno de los cuales tiene una capacidad *m*.
- Todo $i \in N$ tiene una ventana de tiempo $[a_i,b_i]$ que indica cual es el intervalo de tiempo en el cual puede empezar el servicio del cliente. Un vehículo puede llegar al cliente antes del inicio de su ventana de tiempo pero no puede empezar el servicio hasta el inicio de dicha ventana. En el caso del depósito, el inicio de la ventana de tiempo indica cuando un vehículo sale del mismo; y el fin de la ventana de tiempo indica hasta cuando puede volver al mismo.
- θ_i es el tiempo en el cual se inicia el servicio para el cliente *i*.
- C* es el conjunto de clientes no ruteados en un determinado momento
- f_{ik} es la fracción de la demanda del cliente *i* que es entregada por el vehículo *k*.

```
k=1
Repetir
   Crear una ruta para el vehículo k  Rk = (0,0)
   j*=0, θ0= a0
   Repetir
         i= j*
         Buscar un cliente no ruteado j* (Σ(h∈V) fj*h < 1) más cercano
         al cliente i de acuerdo al criterio
         j* ∈ arg minj ∈ C* {tij + max(aj - θi - tij,0)} teniendo en
         cuenta la factibilidad en términos de ventanas de tiempo.
         Insertar el cliente j* en la ruta Rk y
         θi* = θi + tij* + max(aj* - θi - tij*,0)
         Calcular la capacidad remanente uk del vehículo k
   hasta que uk=0 ó no haya más inserciones válidas
   k=k+1
hasta que todos los clientes hayan sido ruteados
```

Figura 24 Construcción de la solución inicial

Cuando la demanda del cliente seleccionado *j** excede la capacidad remanente del vehículo el resto de la demanda se deja para que sea considerada por otros vehículos. Este hecho implica una diferencia con la forma en la cual el algoritmo *Splitabú* construye la solución inicial, dado que en este caso la solución inicial puede incluir entregas partidas aunque no se tiene en cuenta si son redituables o no.

El algoritmo que construye la solución inicial puede llegar a construir una solución con más rutas que la cantidad de vehículos disponibles. Esto se intenta corregir en la siguiente etapa y si no es posible entonces se devuelve que no hay solución factible.

La segunda etapa está basada en *Tabú Search* y para su explicación se utiliza la siguiente notación:

- $\sigma = (R_1,..., R_v)$ es una solución factible que está formada por un conjunto de rutas
- *S* es el conjunto de conjuntos de rutas factibles
- Cada solución $\sigma \in S$ tiene asociado un conjunto de vecindades (soluciones) $N(\sigma) \subset S$, tal que cada $\sigma^* \in N(\sigma)$ se obtiene aplicando una movida a σ.

Las posibles movidas que se utilizan en este algoritmo quedan definidas por cuatro operadores.

Considerando que:

- R_h es una ruta
- *i-1* es el cliente predecesor del cliente *i* en una ruta
- *i+1* es el cliente sucesor de un cliente *i* en una ruta

los cuatro operadores son los siguientes:

Operador ***relocate***: Para los clientes $i \in R_k$ y $j_\alpha \in R_l$, esta movida consiste en colocar *i* después de j_α en R_l, siendo α=1,...,β y $\beta \geq 1$ es la cantidad de vehículos que sirven al cliente *i*. Las nuevas rutas son: R_k = (0,...,*i-1*,*i+1*,...,0) y R_l = (0,..., j_α,*i*,j_α*+1*,…,0). Cuando $i \in R_k \cap R_l$ una entrega partida puede ser eliminada y la posición de *i* en R_l permanece incambiada. En la Figura 25 se muestra un ejemplo.

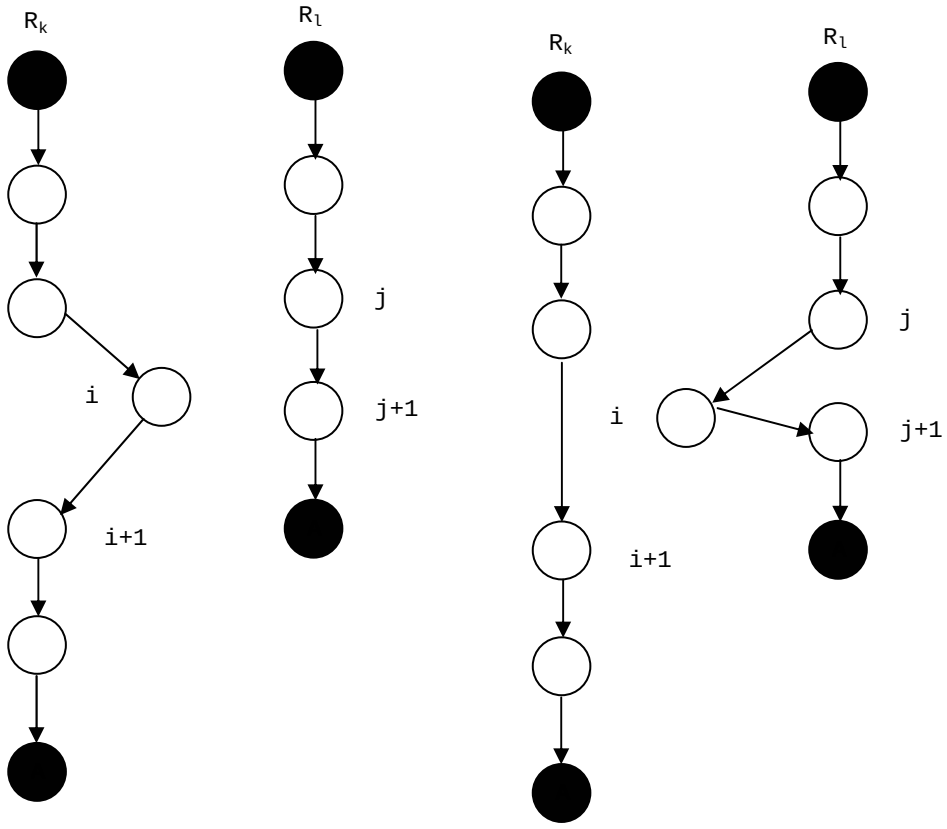


Figura 25 Operador *Relocate*

Operador ***relocate split***: Para los clientes $i \in R_k \cap R_l$ y $j \in R_l \setminus R_k$ esta movida consiste en remover *i* de R_k insertándolo en R_l e insertar *j* en R_k, sin eliminarlo de R_l. La cantidad entregada al cliente *i* en R_l es la cantidad que era entregada a *i* en R_k y la cantidad entregada al cliente *j* en R_k es esta misma cantidad. En la Figura 26 se muestra un ejemplo.

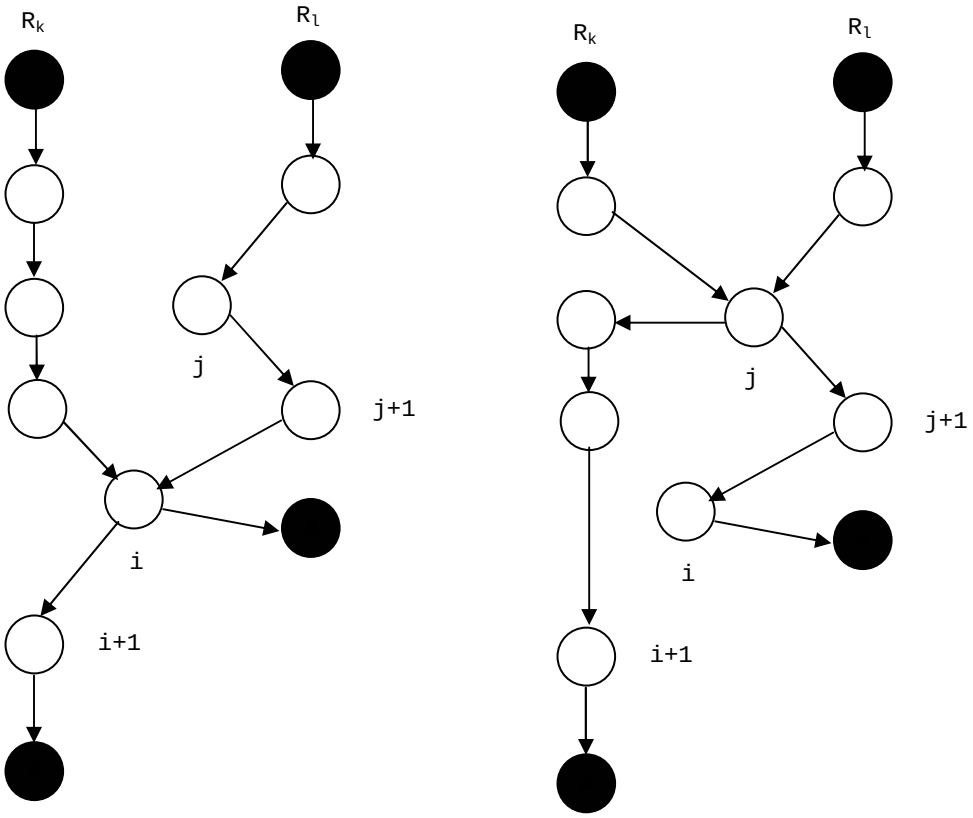


Figura 26 Operador *Relocate split*

Operador ***exchange***: Esta movida consiste en intercambiar los clientes $i \in R_k$ y $j \in R_l$ entre las rutas R_k y R_l. El cliente *i* es insertado en la ruta $R_l \setminus \{j\}$ pero no necesariamente en la posición en la cual estaba el cliente *j*. Análogamente, el cliente *j* es insertado en alguna posición en la ruta $R_k \setminus \{k\}$. Cuando $i \in R_k \cap R_l$ e $i \neq j$ una entrega partida puede ser eliminada. En la Figura 27 se muestra un ejemplo.

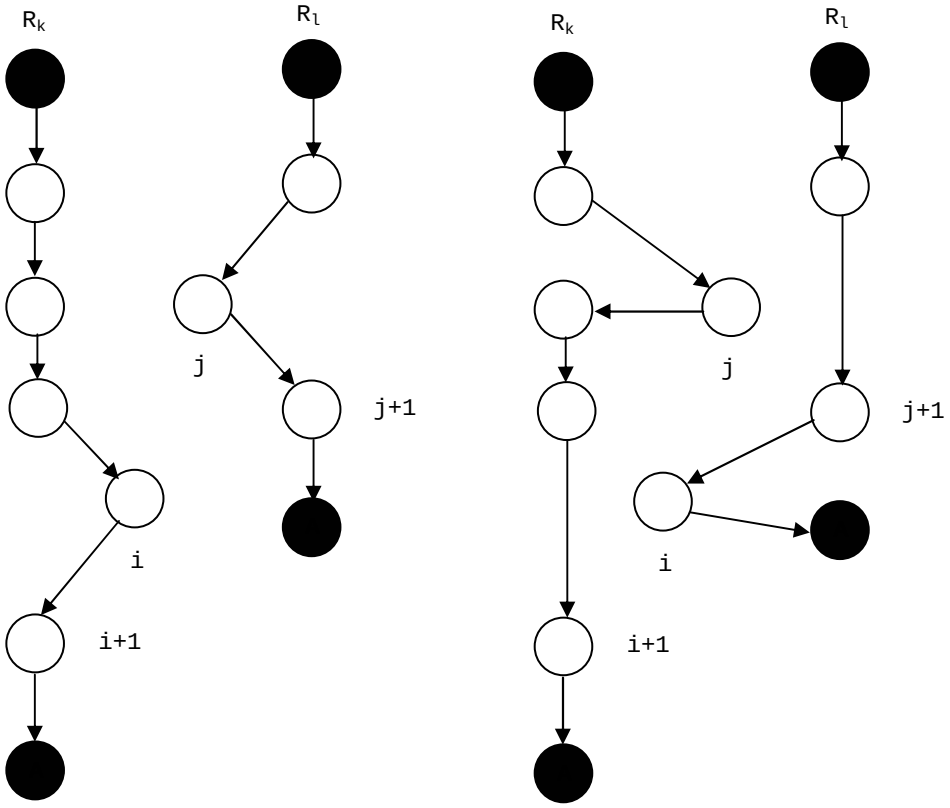


Figura 27 Operador *exchange*

Operador ***2-opt****: Dados los clientes $i \in R_k$ y $j \in R_l$ esta movida consiste en generar las nuevas rutas: R_k=(0,...,*i*,*j+1*,*j+2*,...,0) y R_l=(0,..., *j*,*i+1*,*i+2*,…,0). Cuando $i \in R_k \cap R_l$ e *i* coincide con *j+1* una entrega partida puede ser eliminada entre R_k y R_l. En la Figura 28 se muestra un ejemplo.

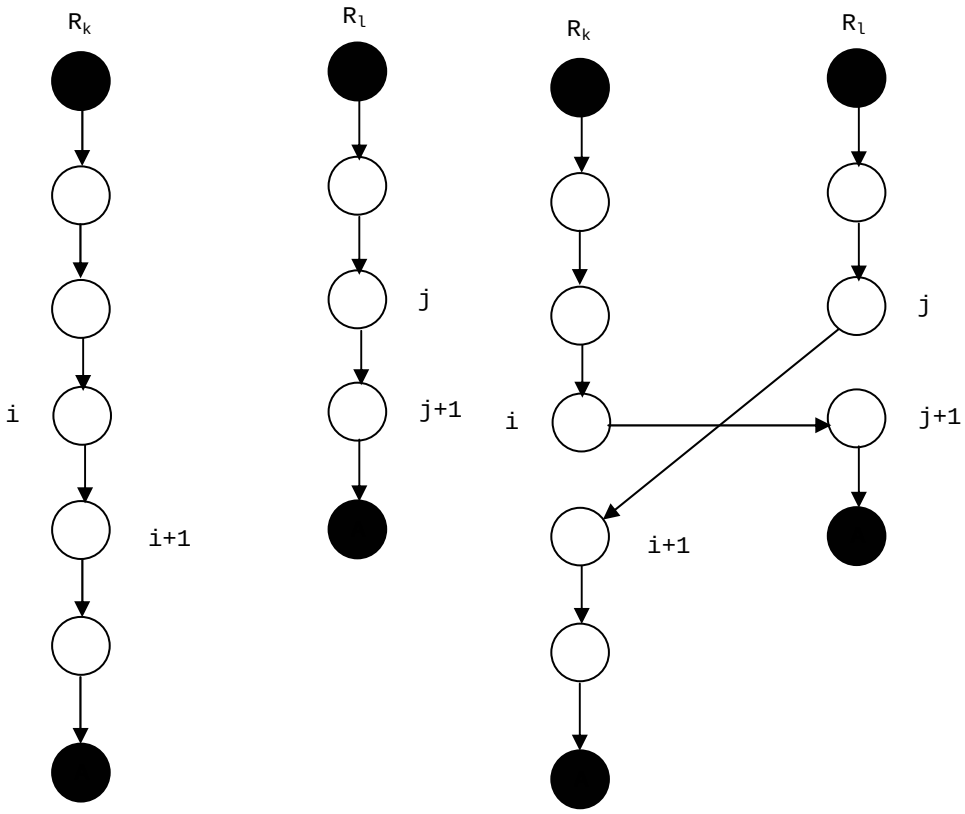


Figura 28 Operador *2-opt**

En cada iteración de esta etapa se evalúan los cuatro operadores sobre la solución actual y se elige la mejor solución generada factible. Esta solución generada puede ser tabú o no. Es tabú si la movida que la genera es tabú, aunque esto puede ser anulado por el criterio de aspiración (mejor costo total hasta el momento). La movida inversa se considera tabú por las siguientes *p* iteraciones a menos que esta movida sea anulada por el criterio de aspiración.

Para explicar las movidas tabú se considerará una solución σ cuyo costo es $z(\sigma)$. A los operadores de movidas *relocate*, *relocate split*, *exchange* y *2*-opt* se le asocia un índice *i* variando de 1 a 4 respectivamente y se define para cada operador *i*:

- N_i es la vecindad definida por el operador.
- $T_i(\sigma)$ es el conjunto de soluciones para las cuales las movidas definidas por el operador son tabú.
- $A_i(\sigma) \subseteq T_i(\sigma)$ es el conjunto de soluciones con movidas tabú tales que estas movidas son anuladas por el criterio de aspiración.

Sean las rutas R_k y R_l e $i \in R_k$ y $j \in R_l$, al aplicar los distintos operadores de movidas se obtienen las siguientes movidas tabu:

Operador ***relocate***: Se setea TABU(i, R_k)=p, significado que volver a colocar i en R_k es tabú por las siguientes p iteraciones, y se considera $T_1(\sigma) \subseteq N_1(\sigma)$ como el conjunto de soluciones en la vecindad $N_1(\sigma)$ tales que $i \in R_k$ y TABU(i, R_k) > 0 para algún i y k.

Operador ***relocate split***: Se setea TABU(i, R_k)=p y se considera $T_2(\sigma) \subseteq N_2(\sigma)$ definido de manera análoga que $T_1(\sigma)$.

Operador ***exchange***: Se seta TABU(i, R_k)=p y TABU(j, R_l)=p, se considera $T_3(\sigma) \subseteq N_3(\sigma)$ definido de manera análoga que $T_1(\sigma)$.

Operador ***2-opt****: Se setea TABU(i, $i+1$)=p y TABU(j, $j+1$)=p, significando que volver a insertar los arco (i, $i+1$) en R_k y (j, $j+1$) en R_l es tabú por las siguientes p iteraciones, se considera $T_4(\sigma) \subseteq N_4(\sigma)$ como el conjunto de soluciones en la vecindad $N_4(\sigma)$ en las cuales h es el sucesor de i en alguna ruta y TABU(i,h) > 0.

Además de los operadores, en esta etapa se utilizan los siguientes algoritmos:

Route saving: Dada una ruta R_s con hasta η clientes (siendo η un valor dado) se intenta insertar todos los clientes de R_s en las otras rutas de la solución usando el operador *relocate*. Si es posible ubicar a todos los clientes en otras rutas, la ruta R_s es eliminada de la solución y en caso contrario se mantiene en la solución.

Intra relocating: Dado un cliente $i \in R_k$ se intenta colocarlo en otra posición de la ruta R_k que implique un menor costo.

El primer algoritmo se aplica cada q iteraciones y el segundo se aplica cuando no se ha mejorado la mejor solución hallada por v iteraciones consecutivas.

La etapa (Figura 29) finaliza cuando se han realizado y iteraciones sin mejorar la mejor solución.

```
Construir la solución inicial y aplicar a la misma el algoritmo US
[17] y el procedimiento Route Saving.
Sea σ la solución hallada, σ*=σ
Inicializar las listas tabu
Repetir
      Para cada operador i
            σ'i = arg min {z(γ)| γ ∈ (Ni(σ) – Ti(σ)) ∪ Ai(σ)}
      i' = arg min i=1..4 {z(σ'i)}
      σ = σ'i
      Setear la movida inversa a σ como tabú por las siguientes p
      iteraciones, actualizar las lista tabú decrementando las
      entradas en 1
      Si z(σ) < z(σ*) entonces σ = σ*
      Aplicar Route Savings cada q iteraciones
      Aplicar Intra Relocating luego de que v iteraciones
      consecutivas no mejoran la mejor solución hallada
Hasta que y iteraciones consecutivas no hayan mejorado la mejor
solución conocida
```

Figura 29 *Tabú Search*

En la última etapa se realiza una post optimización de la solución encontrada en la etapa de *Tabú Search* que consiste en aplicar el algoritmo *US* (parte del algoritmo *Genius* [17]) y operaciones de relocalización de clientes en su propia ruta (*intra relocating*)

Para testear el algoritmo propuesto los autores usaron los siguientes valores para los parámetros del algoritmo:

- Largo de las listas tabú: $p = 15$
- Criterio de finalización: $y = 100$
- *Route savings*: $\eta = 7$ y $q = 5$
- Aplicación de *Intra Relocating*: $v = 15$
- Parámetros usados por el algoritmo *US*: $p_1 = 4$ y $p_2 = 5$

y como datos se utilizaron los problemas *VRPTW* de Solomon [20] de clase C1, C2, R1, R2, RC1 y RC2 de 100 clientes con distancia euclidiana y considerando distintos escenarios de demanda de los clientes.

Se compararon los resultados que se obtienen aplicando el algoritmo propuesto con una variante del mismo que no aplica entregas partidas (y que por lo tanto resuelve un *VRPTW*) Se observó que la opción de permitir entregas partidas es favorable tanto en términos de la distancia total recorrida como en la cantidad de vehículos usados en casi todos los casos a excepción de dos casos correspondientes a las clases RC1 y RC2.

También se observó que cuando se realizan entregas partidas el tiempo de ejecución se incrementa sustancialmente.

Finalmente compararon los resultados del algoritmo con los mejores resultados publicados para los problemas *VRPTW* de Solomon utilizados en las pruebas. En general el costo total de la solución y la cantidad de vehículos usados por el algoritmo propuesto por los autores son similares a los mejores resultados publicados, dado que en algunos casos se mejora el costo total pero se utilizan más vehículos, en algunos se obtiene el mismo costo total y la misma cantidad de vehículos y en otros se obtiene un costo total mayor pero la misma cantidad de vehículos.

4.2.4 Scatter search algorithm

La metaheurística de búsqueda dispersa (*scatter search*) [21] pertenece a la familia de algoritmos evolutivos, los cuales se distinguen por estar basados en la combinación de un conjunto de soluciones para crear nuevas soluciones. A diferencia de los algoritmos genéticos, que utilizan estrategias aleatorias sobre un conjunto grande de soluciones, en esta metaheurística se utilizan estrategias sistemáticas sobre conjuntos pequeños de soluciones.

El esquema general de la búsqueda dispersa se muestra en la Figura 30. El mismo puede ser adaptado para resolver distintos problemas de optimización. La búsqueda dispersa se basa en combinar las soluciones de un conjunto de referencia, el cual almacena soluciones de buena calidad. El concepto de buena calidad no se refiere solamente al costo sino también a la diversidad que la solución aporta al conjunto de referencia.

```
P = Ø
Mientras |P| ≠ sizeP
      Construir una solución con el método de generación diversa
      Aplicar el método de mejora para tratar de mejorarla. Sea x
      la solución obtenida. Si x ∉ P entonces P = P ∪ {x}
Construir el conjunto de referencia RefSet = {x1,...,xb} con las
b/2 mejores soluciones de P y las b/2 soluciones de P más diversas
a las ya incluidas.
Evaluar las soluciones en RefSet y ordenarlas según el valor de
función objetivo f de menor a mayor
NuevaSolución = TRUE
Mientras (NuevaSolución)
      NuevaSolucion = FALSE
      Generar los subconjuntos de RefSet en los que haya al menos
      una nueva solución.
      Mientras (Queden subconjuntos sin examinar)
            Seleccionar un subconjunto y etiquetarlo como
            examinado
            Aplicar el método de combinación a las soluciones del
            subconjunto
            Aplicar el método de mejora a cada solución obtenida
            por combinación. Sea x la solución mejorada, si
            f(x) < f(xb) ∧ x ∉ RefSet entonces actualizar Refset
            eliminando xb y agregando x. Reordenar RefSet
            NuevaSolucion = TRUE
```

Figura 30 Búsqueda dispersa

Campos *et al* [22] presentaron en 2007 un algoritmo basado en búsqueda dispersa que genera una solución factible usando la mínima cantidad de vehículos. Esto último se basa en el hecho de que siempre existe una solución factible al problema *SDVRP* usando una cantidad mínima *k* de vehículos, siendo *k* el menor entero que es mayor o igual que $\sum_i di \,/\, Q$. Observar que esto no se cumple si la demanda de los clientes no puede ser partida.

El hecho de generar una solución que utiliza una cantidad mínima de vehículos puede ser interesante desde un punto de vista práctico considerando los costos fijos de los vehículos

A continuación se presenta una explicación de cada fase del método tanto desde un punto de vista conceptual y también las adaptaciones de las mismas para resolver el *SDVRP* que proponen los autores.

El *método de generación diversa* es un generador de soluciones que se utiliza para crear un conjunto *P* (de tamaño *sizeP*) de soluciones diversas entre sí.

En [22] los autores utilizan dos heurísticas de resolución del problema *VRP* adaptadas para entregas partidas.

La primer heurística se basa en la heurística de Lin *et al* [23] para construir una única ruta con todos los clientes y el depósito.

A partir de esta ruta se obtiene una solución factible para el *SDVRP* con *k* rutas. Suponiendo que la ruta es $0,v_1,v_2,...,v_n,\ 0$ la primer ruta comienza en el depósito y visita sucesivamente los clientes $v_1,v_2,...$ hasta el cliente v_i, para el cual la demanda acumulada de la ruta a los sumo igual a la capacidad del vehículo. Si la demanda del cliente v_i es satisfecha completamente se finaliza la ruta y se comienza una nueva ruta visitando el cliente v_{i+1} en caso contrario se finaliza la ruta y se comienza una nueva ruta empezando por el cliente v_i. Esta forma de construir la solución satisface las propiedades estructurales 1 y 2 mencionadas en la sección Propiedades.
Por último se realizan cambios en las rutas buscando balancear la carga de los vehículos asignados a las distintas rutas.

La ruta inicial es usada para generar otras soluciones siguiendo el mismo procedimiento pero empezando en otros clientes de la misma.

La segunda heurística es una versión modificada del algoritmo de ahorros en paralelo de Clarke y Wright [24]. Según esta heurística, a partir de una solución inicial que consiste en rutas que visitan un solo cliente se utiliza la noción de ahorro ($s_{ij} = c_{0i} + c_{0j} - \lambda c_{ij}$) para unir rutas (*0*,*i*,*0*) y (*0*,*j*,*0*) en la nueva ruta (*0*,*i*,*j*,*0*). Este procedimiento se repite hasta que no haya más uniones factibles debido a la capacidad de los vehículos ó bien no haya más ahorros.
Los ahorros se calculan solamente para la vecindad formada por los clientes más cercanos a cada cliente.

Se admite la entrega partida a un cliente *l* solamente cuando el mejor ahorro corresponde a la unión de una ruta *r* con una ruta que solo visita el cliente *l* y la demanda excede la capacidad del vehículo. En este caso parte de la demanda del cliente *l* es satisfecha por la ruta *r* y se mantiene una ruta que solo visita al cliente *l* por la demanda restante.

Para limitar la complejidad de este procedimiento, cualquier cliente es visitado por a lo sumo 2 rutas.

A pesar que este procedimiento no garantiza una solución factible utilizando la mínima cantidad de vehículos, esta situación no se fue detectada en los testeos computacionales realizados por los autores.

Para generar más de una solución, se prohíben la mitad de los ahorros usados en una solución cuando se calcula otra solución. Esta prohibición se realiza con una probabilidad que es directamente proporcional a la frecuencia de uso del ahorro en las soluciones previamente generadas.

La mitad de las soluciones factibles que forman el conjunto *P* son generadas usando la primer heurística y la otra mitad usando la segunda heurística.

El *método de mejora* consiste en aplicar algoritmos para mejorar soluciones. En general son algoritmos de búsqueda local.

En [22] los autores aplican las siguientes movidas a todos los clientes que no tengan demanda partida: intercambios 1-0 (moviendo un cliente de una ruta a otra), intercambios 1-1 (intercambiando un cliente de una ruta con otro cliente de otra ruta).
Los clientes con demanda partida se sacan de las rutas que los visitan y se intenta colocarlos en 2 rutas que puedan satisfacer su demanda (este es un caso particular de *k*-`split interchanges` con *k* =2 usado por el algoritmo *Two Stages* de Dror y Trudeau).
Por último se aplican intercambios 2-2 (se intercambia una pareja de clientes sucesivos de una ruta con una pareja de clientes sucesivos de otra ruta).
Cuando no se obtienen más mejoras en la solución, se aplica a cada ruta de la misma el algoritmo *2-opt*.

La *construcción del conjunto de referencia* se realiza con las *b*/2 mejores soluciones (en términos de una función objetivo) contenidas en el conjunto *P* y las restantes *b*/2 soluciones se seleccionan del conjunto *P* con el criterio de maximizar la mínima distancia con aquellas ya contenidas en el conjunto de referencia. El criterio de distancia es particular de cada aplicación. Las soluciones de este conjunto se ordenan según el valor de la función objetivo de menor a mayor.

En [22] se seleccionan las *b*/2 soluciones de *P* de menor costo y las restantes *b*/2 soluciones corresponden a aquellas soluciones de *P* que difieren más al ser comparadas con las soluciones de menor costo. La diferencia se mide como la cantidad de aristas (parejas de clientes sucesivos) que están en una solución pero no en la otra.

Las soluciones del conjunto de referencia son combinadas entre si para generar nuevas soluciones. Estas nuevas soluciones entran al conjunto de referencia (*actualización del conjunto de referencia*) reemplazando soluciones del mismo en caso de que sean mejores en términos de la función objetivo. De esta manera el conjunto de referencia mantiene un tamaño *b* constante pero el valor de las soluciones que contiene va mejorando a lo largo de la búsqueda.

La *generación de subconjuntos* es un método para generar subconjuntos de soluciones contenidas en el conjunto de referencia a los cuales se les aplicará un cierto método de combinación. Una posibilidad, utilizada con frecuencia, es considerar que cada subconjunto es una de la posible pareja de soluciones del conjunto de referencia. Esta forma de generar los subconjuntos es la que se utiliza en [22].

Cuando en la Figura 30 se dice que los subconjuntos que se generen contengan una solución nueva significa que por lo menos una de las soluciones del subconjunto sea una solución que haya entrado al conjunto de referencia después de realizar la última combinación de todo el conjunto de referencia.

El *método de combinación* es un aparte fundamental de la búsqueda dispersa y consiste combinar todas las soluciones del conjunto de referencia, para lo cual se utiliza la generación de subconjuntos descripta anteriormente. Las soluciones de cada subconjunto son combinadas entre si para generar una nueva solución.

En [22] se emplea como método de combinación un procedimiento que comienza identificando para cada solución un conjunto de clientes críticos según las siguientes condiciones:

1. todos los clientes con demanda partida
2. todos los clientes de aquellas rutas que visitan 1 o 2 clientes
3. los clientes que al ser eliminados de una ruta generan el mayor ahorro , para cada ruta con al menos 3 clientes
4. todo cliente tal que por lo menos uno de sus tres vecinos más cercanos pertenece a una ruta diferente.

luego para combinar una solución *A* con una solución *B*, se considera cada uno de los cliente críticos de la solución *A* según las condiciones 1..3 y se modifica la solución *A* moviendo cada cliente según la situación del mismo en la solución *B*:

- si el cliente tiene la demanda partida en la solución *B* entonces no se mueve en la solución *A*.
- si no tiene demanda partida en la solución *B*, se considera el cliente anterior (α) y el cliente posterior (β) en la ruta de la solución *B* y se localizan los mismos en la solución *A*. Si los clientes α y β son clientes consecutivos en la misma ruta de la solución A entonces se inserta el cliente crítico entre ellos y en caso contrario el cliente crítico es insertado después del cliente α o antes del cliente β dependiendo de cual opción es mejor en costo.

Cuando se han considerado todos los clientes críticos de la solución A, el método de combinación usado puede haber generado una solución no factible por la carga de cada ruta. En este caso se aplica un algoritmo que considera ciertas movidas para hacer la solución factible.

Cada vez que se obtiene una nueva solución factible como resultado de la combinación se le aplica el método de mejora descrito anteriormente.

Una vez que se han realizado todas las combinaciones de subconjuntos de soluciones y ninguna solución nueva entra al conjunto de referencia se aumenta el tamaño del conjunto de clientes críticos incluyendo los clientes de la condición 4 y se realizan todas las combinaciones de subconjuntos nuevamente.

La búsqueda dispersa finaliza cuando se han analizado todas las combinaciones y no se genera una nueva solución factible para entrar al conjunto de referencia. En [22] se agrega además como condición de finalización haber realizado una cantidad preestablecida de iteraciones actualizando el conjunto de referencia.

Para la los testeos se utilizaron los mismos casos de prueba que en [6], el tamaño del conjunto *P* fue fijado en 150 y el tamaño del conjunto de referencia en 25 y se comparan los resultados con los de *Splitabú*.

La comparación muestra que se obtienen soluciones que utilizan una menor cantidad de vehículos que *Splitabú*.
En cuanto al costo de la solución la comparación no es concluyente, dado que dependiendo de los escenarios se obtiene un costo mayor o menor. Esto se observa fundamentalmente en aquellos escenarios en los cuales la demanda de cada cliente es grande (superior a la mitad de la capacidad de los vehículos) y es atribuido por los autores a que el algoritmo no está diseñado para estas situaciones dado que se intenta utilizar la mínima cantidad de vehículos.
La comparación de los tiempos de ejecución muestra que son del mismo orden.

4.2.5 Otros algoritmos

En sus artículos de 1992 [25] y 1995 [26] Frizzell et al presentaron heurísticas para resolver casos particulares de los problemas ***SDVRP*** y el ***SDVRPTW*** respectivamente. En ambos casos supusieron que los clientes se ubican en una red geográfica de tipo grilla y presentaron heurísticas y algoritmos de post procesamiento para realizar mejoras que fueron específicamente diseñadas teniendo en cuenta dicha forma de ubicación de los clientes.

Mullaseril et al [27], estudiaron en 1997 un caso real de distribución de alimento para ganado, modelando el problema como un *Capacitated Rural Postman Problem* y proponiendo una heurística que es una adaptación del algoritmo de Dror y Trudeau (usando fundamentalmente `k-split Interchanges` y `Route Addition`)

4.3 Soluciones híbridas

Esta alternativa consiste en combinar algoritmos de búsqueda de soluciones aproximadas, basados en heurísticas o metaheurística, con métodos de resolución exactos.

4.3.1 EMIP

Chen et al [28] presentaron en 2006 un algoritmo denominado *EMIP* (***E****ndpoint* ***M****ixed* ***I****nteger* ***P****rogramming*) que comienza generando una solución inicial al problema *SDVRP* resolviendo un VRP mediante el algoritmo de Clarke y Wright [14].

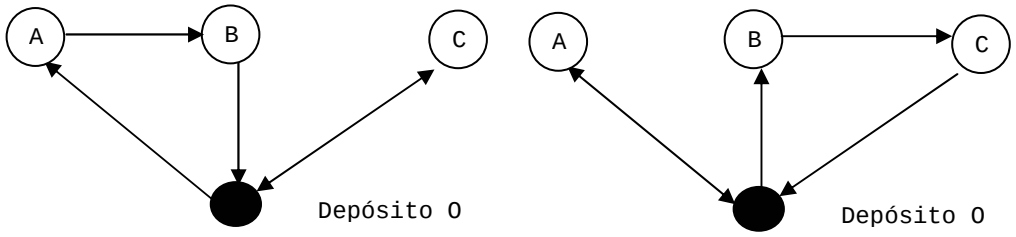
Para cada ruta de la solución inicial se consideran los clientes extremos de la ruta (los clientes que ocupan la primer y última posición en la ruta) y los *c* clientes vecinos más cercanos a cada uno de ellos (*c* es un parámetro del algoritmo y los clientes vecinos son también clientes extremos).

La idea propuesta es encontrar las redistribuciones de la demanda de cada cliente extremo entre sus vecinos que maximicen el ahorro total (entendiéndose por ahorro las diferencias entre las distancias recorridas).

Después de esta redistribución hay 3 posibilidades para cada cliente extremo:

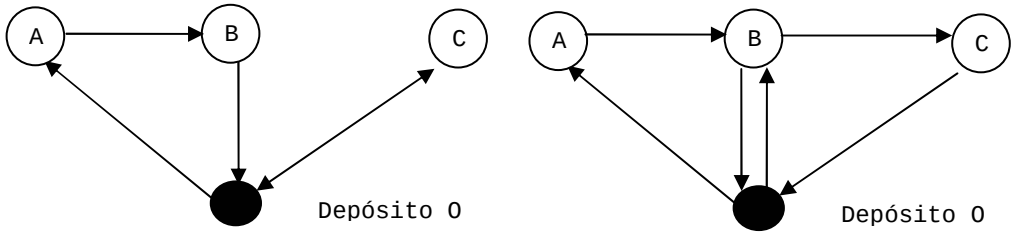
1. no hay cambios en su ubicación
2. el cliente extremo es eliminado de su ruta y toda su demanda es pasada a otra ruta u otras rutas
3. La demanda del cliente extremo es parcialmente redistribuida desde su ruta y parte de la misma es pasada a otra ruta u otras rutas

En la Figura 31 se muestra un ejemplo de la posibilidad 2 y en la Figura 32 un ejemplo de la posibilidad 3, además del ahorro que se genera en cada una de ellas.



$ahorro = (d_{AC} + d_{B0} - d_{A0}) - (d_{0B} + d_{BC} - d_{0C})$

Figura 31 Toda la demanda de un cliente extremo pasa a otra ruta



$ahorro = -(d_{0B} + d_{BC} - d_{0C})$

Figura 32 Parte de la demanda de un cliente extremo pasa a otra ruta

Para encontrar las redistribuciones que maximizan el ahorro se resuelve un *MIP* (***M**ixed **I**nteger **P**rogramming*) según una formulación dada por los autores que tiene en cuenta el conjunto *N* de clientes extremos y los conjuntos de vecinos *c* para cada cliente extremo.

Se ejecuta el *MIP* hasta encontrar la solución óptima o durante un tiempo máximo de *T* segundos. Cuando termina la ejecución se guarda la mejor solución factible encontrada.

La cardinalidad de los conjuntos *c* y el valor e *T* son determinados previamente a la ejecución del algoritmo dependiendo de la cardinalidad de *N* y pueden ser determinados de distintas maneras.

La solución encontrada al resolver el *MIP* se utiliza como solución inicial para resolver un segundo *MIP* en el cual se incrementa en un 50% el tamaño de los conjuntos de vecinos *c* y es ejecutado hasta encontrar la solución óptima o durante un tiempo máximo que es un 60% del tiempo *T* (estos factores pueden ser modificados)
Cuando termina la ejecución, la mejor solución factible encontrada es post procesada aplicándole el algoritmo desarrollado por Li et al [29]

El algoritmo propuesto tiene limitaciones dado que para algunos problemas no todas las soluciones factibles son encontradas por el mismo, tal como muestran los autores mediante ejemplos.

Para los testeos fueron seleccionados los problemas 1, 2, 4, 5, 11 y 12 propuestos en [30] y [1] con 50, 75, 100, 120, 150 y 199 clientes, con distancias euclidianas y considerando distintos escenarios de demanda de los clientes.

Los resultados obtenidos muestran que cuando la demanda de los clientes es pequeña con respecto a la capacidad de los vehículos, la resolución del problema de programación entera no tiene mucha incidencia en el ahorro que se logra (el cual es debido fundamentalmente a la post optimización que se aplica al final). En cambio cuando la demanda es grande se da la situación inversa, por lo cual este algoritmo parece ser efectivo cuando hay clientes con demandas grandes con respecto a la capacidad de los vehículos.

Los resultados son comparados con los obtenidos por el algoritmo *Splitabú* en situaciones similares (pero no sobre los mismos problemas) llegando, los autores, a la conclusión que en general se logra una mejora en el costo de la solución del algoritmo *EMIP*. También comparan los resultados obtenidos por ambos algoritmos sobre los problemas con las demandas originales observándose que *EMIP* produce mejores soluciones que *Splitabú*.

Para los testeos fueron utilizados también problemas tomados de [9] para los cuales está publicada una cota inferior de la solución optima (en *http://www.uv.es/belengue/sdvrp.html*), observándose que los resultados generados por *EMIP* son un 5.85% mayores que la cota en promedio.

4.3.2 Optimization-based heuristic

Archetti *et al* [31] presentaron en 2006 un algoritmo, denominado *Optimization-based heuristic*, que utiliza la información generada por el algoritmo *Splitabú* de Archetti et al [6] para identificar partes del espacio de soluciones que pueden contener soluciones de alta calidad y luego explorar en profundidad estas partes del espacio de soluciones mediante un optimizador *MIP* (***M****ixed* ***I****nteger* ***P****rogramming*) que resuelve un modelo formulado por los autores.

La primera fase consiste analizar el conjunto *S* de todas las soluciones encontradas durante la aplicación del algoritmo *Splitabú* para buscar un conjunto *C* de clientes que probablemente sean atendidos por un único vehículo en soluciones de alta calidad y construir un conjunto *R* de rutas de alta calidad.

Para hallar el conjunto *C* se calcula la cantidad de veces que cada cliente participa en una entrega partida en *S*. Si la cantidad de entregas partidas de un cliente es menor que el 10% de la máxima cantidad de entregas partidas y el cliente no participa en una entrega partida en la solución final de *Splitabú* entonces el cliente forma parte del conjunto *C*.

Para hallar el conjunto *R* se calcula la cantidad de veces que una arista (*i*,*j*) (lo cual indica que luego del cliente *i* se visita al cliente *j*) aparece en alguna de las rutas de *S* y se asocia dicho contador a cada una de ellas Se considera un subconjunto de aristas base que son aquellas cuyo contador es mayor que un cierto parámetro.

Para cada arista base se construye una ruta, comenzando por la propia arista y extendiéndola por ambos extremos agregando otras aristas (que no están en la ruta en construcción) y cuyo contador sea mayor que parámetro. Se sigue de esta manera hasta alcanzar el depósito, verificando las restricciones de capacidad de los vehículos.

Los estudios realizados por los autores indican que para obtener soluciones de alta calidad el conjunto *R* debe contener una gran cantidad de rutas (para esto los autores utilizan un rango para definir una cota inferior y superior de la cardinalidad de *R*).

Sin embargo cuando el conjunto es muy grande es imposible resolver de forma práctica el modelo de programación entera planteado. Por este motivo se seleccionan subconjuntos *R** de *R*, cuyo tamaño es un parámetro, que incluyen siempre las rutas de la mejor solución conocida,

las rutas con valores positivos en la solución de la relajación lineal del modelo de programación entera sobre todo conjunto *R* (es decir rutas de interés desde una perspectiva global al considerar todo el conjunto *R*) y rutas seleccionadas según un determinado criterio (los autores plantean varios criterios) hasta alcanzar el tamaño del subconjunto.

Usando los subconjuntos *C* y *R** generados se halla la solución del problema de programación entera optimizando durante un determinado tiempo que es un parámetro. Si el tiempo de resolución es menor que el parámetro se genera un nuevo subconjunto *R** sacando las rutas seleccionadas según el criterio y agregando otras rutas de *R* que verifican el criterio. Se sigue de esta manera resolviendo una cantidad máxima (parámetro) de problemas de programación entera, mientras haya tiempo, y quedándose con la mejor solución.

Para los testeos fueron usadas instancias derivadas de los mismos casos utilizados para testear el algoritmo propuesto en [6], comparando los resultados de ambos algoritmos.

Los resultados experimentales muestran una mejora en el costo total de las rutas de 0.5% en promedio con respecto al algoritmo *Splitabú*. Los autores atribuyen este porcentaje relativamente bajo a que las soluciones del algoritmo S*plitabú* son de buena calidad.

En cuanto al tiempo de ejecución se observa que depende fuertemente del tamaño de los subconjuntos de rutas usados en el modelo de programación entera. A medida que el tamaño de los mismos crece se incrementa rápidamente el tiempo necesario para su resolución.

5. CONCLUSIONES

Desde la introducción por parte de Dror y Trudeau en 1989 del problema *SDVRP* mostrando los eventuales beneficios de permitir entregas partidas como alternativa a la resolución de un *VRP*, se han presentado varios artículos sobre *SDVRP* que abarcan distintos temas.

En varios artículos se han realizado investigaciones sobre propiedades estructurales de las soluciones óptimas y sobre la complejidad de este problema. Si bien se trata de estudios teóricos que en general no son aplicables desde un punto de vista práctico por *SDVRP* ser un problema *NP-hard*, algunos de estos resultados pueden ser utilizados en la generación de métodos de resolución que sí son aplicables en la práctica.

Otros artículos corresponden a la descripción de la resolución de situaciones reales modelándolas como un *SDVRP* con buenos resultados.

Por otro lado, algunos autores han propuesto distintas metodologías de resolución del *SDVRP* buscando soluciones óptimas o bien soluciones aproximadas.

En el caso de búsqueda de soluciones óptimas se han aplicado distintas técnicas y formas de modelado. Se han alcanzado resultados satisfactorios, generalmente en problemas con poca cantidad de clientes. A medida que aumenta la cantidad de clientes estas técnicas no resultan aplicables por el tiempo de ejecución necesario para su resolución.

En el caso de búsqueda de soluciones aproximadas hay dos enfoques. El primer enfoque es el uso de heurísticas, como es el caso del algoritmo de Dror y Trudeau, que tienen la virtud de ser sumamente rápidas pero que pueden no generar algunas soluciones factibles y por lo tanto la solución que encuentran puede que esté muy alejada de la solución óptima en términos de costo.

El segundo enfoque propuesto se basa en el uso de metaheurísticas que tienen la virtud de obtener mejores soluciones que las heurísticas debido a que realizan una exploración más adecuada y profunda del espacio de soluciones y por lo tanto brindan mayores posibilidades de encontrar soluciones más próximas a la solución óptima en costo.
Sin embargo demandan más recursos computacionales y tiempo de ejecución (que, de todo modo, no son comparables con los métodos de búsqueda de soluciones exactas).

Las metaheurísticas que han sido utilizadas para resolver el *SDVRP* en la literatura son *Tabú Search* y búsqueda dispersa (*scatter search*).

También, como caso particular, para la búsqueda de soluciones aproximadas se han propuesto combinaciones de heurísticas y/o metaheurísticas con métodos de resolución exactos. Las heurísticas y metaheurísticas se han utilizado para crear soluciones iniciales que luego se intenta optimizar mediante la búsqueda de la solución exacta.
Las restricciones de tiempo de ejecución para hallar una solución en un tiempo razonable implican que cuando los problemas a resolver tienen una cierta cantidad de clientes, los métodos de búsqueda de soluciones exactos deben ser interrumpidos por tiempo.
Esto implica que en general no se encuentre la solución óptima sino una aproximación a la misma.

En resumen, desde un punto de vista de aplicación práctica, parece sumamente razonable el uso de metaheurísticas para resolver el *SDVRP*. La metaheurística con más propuestas es hasta el momento *Tabú Search* y le sigue la búsqueda dispersa. A pesar de que esta última se basa en combinar soluciones de un conjunto de referencia que contiene soluciones de calidad en cuanto a costo y diversidad, no se han encontrado algoritmos basados en la metaheurística ***AMP***

(***A****daptive* ***M****emory* ***P****rocedure*) [32] que es bastante similar a la búsqueda dispersa en cuanto a la combinación y mantenimiento de soluciones diversas.

AMP se basa en la combinación de componentes de soluciones (rutas) almacenados en una memoria. La estrategia para la combinación está fuertemente basada en la idea de que no es muy difícil construir una buena ruta, sino que lo complejo es hallar un conjunto de rutas que sea simultáneamente bueno para todos los clientes. En ese sentido, la combinación de rutas que estén en soluciones diversas de buena calidad, puede originar mejores soluciones.

6. BIBLIOGRAFÍA

[1] Christofides N., Mingozi A., Toth P. *The vehicle routing problem*. Combinatorial Optimization, Joh Wiley & Sons, Chichester. 1979.

[2] Dror M., Trudeau P. *Split Delivery routing*. Naval Research Logistics 37, pp. 283:402, 1990.

[3] Archetti C., Savelsbergh M., Speranza M. G. *Worst case analysis for split delivery vehicle routing problem*. Technical Report 241, Department of Quantitative Methods, University of Brescia, Italia, 2004.

[4] Archetti C, Savelsbergh M., Speranza M. G. *To Split or not to Split: That is the question*. Technical Report 261 Department of Quantitative Methods, University of Brescia, Italia, 2006.

[5] Archetti C., Mansini R., Speranza M. G. *The Split Delivery Vehicle Routing Problem with Small Capacity*. Technical Report n. 201, Department of Quantitative Methods, University of Brescia, Italia, 2001.

[6] Archetti C., Hertz A., Speranza M. G. *A Tabu Search algorithm for the split delivery vehicle routing problem*. Les Cahiers du Gerad ISSN: 0711-2440, 2003.

[7] Dror M., Laporte G., Trudeau P. *Vehicle Routing with Split Deliveries*. Discrete Applied Mathematics 50, pp. 239-254, 1994.

[8] Sierksma G., Tijssen G.A. *Routing Helicopters for Crew Exchanges on Off-shore Locations*. Annals of Operations Research 76, pp. 261-286, 1998.

[9] Belenguer J.M., Martinez M.C., Mota E. .*A Lower Bound for the Split Delivery Vehicle Routing Problem*. Operations Research 48 (5), pp. 801-810, 2000.

[10] Feillet D., Dejax P., Gendreau M., Gueguen C. *Vehicle routing with Time windows and split deliveries*. Working paper.

[11] Lee C.G., Epelman M.A., White III C.C., Bozer Y.A., *A shortest path aproach to the multiple vehicle routing problem with split pick ups*. Transportation Research B, 40, pp. 265-284, 2007.

[12] Liu K. *A study on the split delivery vehicle routing problem*. Ph.D thesis, Mississippi State University, 2005.

[13] Dror M., Trudeau P., *Savings by split delivery routing*. Transportation Science 23 pp. 141-145, 1989.

[14] Bodin L., Golden B., Assad A., Ball M. *The state of the art in the routing and scheduling of vehicles and crews*. Computer Operations Research Special issue, June 1983.

[15] Croes G., *A method for solving traveling salesman problems*. Operations Research 6, pp. 791-812, 1958.

[16] Glover F., Laguna M. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[17] Gendrau M., Hertz A., Laporte G. *New insertion and post optimization procedures for the traveling salesman problem*, Operations Research 40, pp.1086-1094, 1992.

[18] Gendrau M., Herta A., Laporte G. *A tabu search heuristic for the vehicle routing problem*. Management Science 40, pp. 1276-1290, 1994.

[19] Sin C. Ho, Dag Haugland. *A Tabu Search heuristic for the vehicle routing problem with time windows and split deliveries*. Computers and Operations Research 31, 12, pp. 1947-1964, 2004.

[20] Solomon M. *Algorithms for the vehicle routing and scheduling problems with time windows constraints*. Operations Research 35(2), pp. 254-265, 1987.

[21] Martí R., Laguna M. *Scatter Search: Diseño básico y estrategias avanzadas*. http://www.uv.es/~rmarti/paper/docs/ss3.pdf. Noviembre 2007.

[22] Campos V., Corberán A. Mota E. *A scatter search algorithm for the split delivery vehicle routing problem*. Technical Report 08-07 Departamento de Estadística y Investigación Operativa, Universidad de Valencia, España, 2007.

[23] Lin S., Kernighan B. *An effective heuristic algorithm for the travelling salesman problem*. Operations Research 21, pp. 498-516, 1973.

[24] Clarke G., Wright J.V. *Scheduling of vehicles from a central depot to a number of delivery points*. Operations Research 12, pp. 568-581, 1964.

[25] Frizzell P.W, Giffin J.W. *The Bounded Split Delivery Vehicle Routing Problem with Grid Networks Distances*. Asia Pacific Journal of Operational Research 9, pp. 101-116, 1992.

[26] Frizzell P.W, Giffin J.W. *The Split Delivery Vehicle Scheduling Problem with Time Windows and Grid Network Distance*. Computers and Operations Research 22, pp. 655-667, 1995.

[27] Mullaseril P.A., Dror M., Leung J. *Split-Delivery Routing Heuristics in Livestock Feed Distribution*. Journal of Operations Research Society. 48, pp. 107-116, 1997.

[28] Chen S., Golden B., Wasil E. *The Split delivery vehicle routing problem: applications, algorithms, test problems and computational results.* Networks 49(4), pp. 318-324, 2007.

[29] Li F., Golden B., Wasil E. *Very large-scale vehicle routing: New test problems, algorithms and results*. Computer Operations Research 32, p. 1197-1212, 2005.

[30] Christofides N., Eilon S. *An algorithm for the vehicle dispatching problem*. Operations Research Quart 20, pp. 309-318, 1969.

[31] Archetti C., Savelsbergh M., Speranza M. G. *An Optimization-Based Heuristic for the Split Delivery Vehicle Routing Problem*, Technical Report 262, Department of Quantitative Methods, University of Brescia, Italia, 2006.

[32] Rochat, Y., Taillard, E. *Probabilistic diversification and intensification in local search for vehicle routing*. Journal of Heuristics 1 pp. 147-167, 1995.