Estudios de Economía Aplicada
Nº 3, 1995. Págs. 69 a 86

Problemas de rutas con carga y descarga en sistemas lifo: soluciones exactas

Joaquín Antonio Pacheco Bonrostro
Profesor Titular de E.U.E. Empresariales de Burgos

RESUMEN

En el trabajo se propone una estrategia para desarrollar algoritmos exactos para el Problema de Carga y Descarga (PDP) con un solo vehículo sin restricciones de capacidad en sistemas de descarga LIFO, -es decir, en cada momento sólo se puede descargar la última mercancía que ha entrado en el vehículo de entre todas las que se encuentran en él-, a partir de algoritmos Branch & Bound para el Problema del Viajante Asimétrico (ATSP). Para ello se introducen en estos últimos una serie de modificaciones: Inicialmente se incorporan un conjunto de procedimientos de *filtrado* para evitar, en cada vértice del árbol de búsqueda, la selección de arcos que vayan a dar lugar a soluciones no factibles; posteriormente se restringe dicho criterio de selección. Con estas modificaciones se consigue rebajar enormemente el tiempo de computación empleado para solucionar este problema en ordenadores personales. Este trabajo supone una ampliación de los realizados por Pacheco, (1994), y Pacheco y otros, (1.994 pp. 184-201), recientemente sobre este problema.

SUMMARY

In this paper a strategy is proposed for the development of exact algorithms for the Pick-up and Delivery Problem (PDP), with a single vehicle having no capacity limits in Last-In-First-out delivery systems, i.e., at any given moment only the last goods loaded onto the vehicle may be unloaded-, starting from Branch and Bound algorithms for the asymetrical Travelling Salesman Problem (ATSP). With this aim a series of modifications was introduced to these algorithms. Initially a set of filter processes was incorporated to avoid, in each vertex of the tree search, the selection of arcs which will give rise to unfeasible solutions; later this selection criterion was restricted. With these modifications the calculation time necessary for the solution of this problem on a personal computers was greatly reduced. This paper is an extension of a recent work done by Pacheco, (1994), and Pacheco et al., (1.994 pp. 184-201), on this problem.

1. Introducción

El Problema de Carga y Descarga (PDP) puede ser descrito de la forma siguiente: Sea un conjunto de clientes N = {2,3,..,n}; cada cliente i requiere que le sea transportada una mercancía desde un origen i^+ a un destino i^-. Para ello se dispone de m vehículos que parten de una ciudad inicial 1, a la que regresan al final del trayecto. Se trata de diseñar rutas de vehículos con distancia total a recorrer mínima. Se va a denotar $N^+ = \{i^+ / i = 2,...n\}$, el conjunto de puntos de carga y $N^- = \{i^- / i = 2,...n\}$, el conjunto de puntos de descarga, y q(i), i = 2,....n, la mercancía de cada cliente i. Al caso especial en el que las demandas requeridas son todas iguales se le denomina el *Dial-A-Ride-Problem* (DARP), que tiene lugar en el transporte de viajeros, transporte escolar, etc..

Aunque quizás con menos tradición que el TSP o el VRP (Problema de Rutas de Vehículos), el PDP es un problemas de rutas muy conocido y estudiado en los últimos años, y la Programación Matemática cuenta varios trabajos dedicados desarrollar técnicas de solución para este problema. En la mayoría de los casos estas técnicas de solución ya consideran en el problema las *Ventanas de Tiempo* (*Time Windows*).

Entre los algoritmos exactos destacan inicialmente el de Kalantari y otros, (1.985 pp.377-386), para el PDP con un solo vehículo sin restricciones de capacidad, que proponen una serie de modificaciones en el algoritmo de Little y otros, (1.963 pp.972-989), para el TSP, con el fin de impedir la selección de arcos incompatibles con las relaciones de precedencia *origen-destino* (i^+ anterior a i^-); Psaraftis, (1.983 pp.351-360), usa programación dinámica para resolver el DARPTW con un sólo vehículo. Los estados son de la forma (j, k_2, ..., k_n), donde j es el vértice actualmente visitado, y cada k_i puede tener tres valores diferentes, que indican el *estatus* de la mercancía de cada cliente: no ha sido cargada, ha sido cargada pero no descargada, y ha sido descargada; Desrosiers y otros, (1.986 pp.301-326), adaptan este algoritmo de 2n estados, a PDPTW con un sólo vehículo.

En cuanto a los algoritmos heurísticos Jaw y otros, (1.986 pp.243-257), describen un algoritmo de inserción para una variante del DARPTW. El uso de las técnicas descritas en el párrafo anterior también pueden usarse como subrutinas en otros técnicas heurísticas para problemas con varios vehículos, como en el caso de las del tipo *Cluster 1º-Ruta 2º*. Ejemplos de este tipo de aproximaciones los tenemos en los trabajos de Desrochers y otros, (1.991 pp.291-310).

Ahora bien, considérese en el planteamiento del PDP con un vehículo la siguiente restricción: En cada momento sólo puede ser descargado la última

mercancía que ha sido cargada, de entre las que se encuentren en el vehículo. Este problema, que se va a denominar como PDP con un vehículo con sistema de descarga LIFO (último en entrar primero en salir), aparece en muchas situaciones en el mundo del Transporte y la Industria: transporte de determinados gases industriales, distribución de pales en la industria del automóvil, o, en general, en aquellas actividades de reparto en las que se quiere un tiempo controlado para la descarga.

Existen algunas referencias a los sistemas de descarga LIFO, como Assad, (1.988 pp.7-46), que menciona este problema sin proponer soluciones concretas.

En este trabajo se propone una forma de desarrollar algoritmos exactos para el PDP con un vehículo sin restricciones de capacidad con sistema de descarga LIFO mediante una serie de modificaciones en los algoritmos Branch & Bound existentes para el ATSP. Estas modificaciones consisten básicamente en añadir *filtros*, es decir, procedimientos destinados a impedir la elección de arcos que vayan a dar lugar a soluciones no factibles de forma análoga a como lo hacen Kalantari y otros, (1.985 pp.377-386), para el PDP con 1 sólo vehículo sin restricciones de capacidad. Además, para aumentar el efecto de estos filtros se va a variar el proceso de elección del arco o arcos que entran en la solución respecto a los algoritmos originales.

En los dos siguientes apartado se explican las modificaciones que se introducen, -filtros y elección de arcos-, en el tercer apartado se muestran los efectos de las modificaciones desarrolladas en el tiempo de computación mediante la resolución de una serie de problemas simulados, dejando para el quinto apartado las conclusiones. Se ha añadido un apéndice para describir con detalle todos los procedimientos utilizados.

Para hacer las pruebas el algoritmo inicial utilizado es el de Little y otros, (1.963 pp.972-989), pero se pueden hacer las mismas modificaciones a otros algoritmos exactos para el ATSP. Se ha añadido un apéndice en el que se describen los aspectos teóricos del algoritmo de Little, así como el seudocódigo de las modificaciones propuestas y el algoritmo resultante.

Como se ha dicho anteriormente Se supondrá que la capacidad del vehículo es superior a la cantidad total de mercancía transportada; para el caso general se proponen en el quinto apartado una serie de modificaciones fáciles de incorporar.

2. Descripción de los filtros

En este apartado y el siguiente se describen las modificaciones que se introducen en los algoritmos Branch & Bound para el ATSP para dar lugar a una

técnica de solución para el PDP con un sólo vehículo con sistema de descarga LIFO. Estas modificaciones consisten básicamente en el desarrollo de *filtros* para impedir la elección de arcos no factibles y cambiar la forma de elección de nuevos arcos.

Para diseñar filtros eficaces, se han desarrollado una serie de resultados teóricos

2.1. Aspectos teóricos

Definición

Sea T una ruta factible del PDP con un sólo vehículo con sistema de descarga LIFO, se define la siguiente relación de precedencia:

$\forall\ a,b \in N^{-} \cup N$, a *ant* b si y sólo si a es visitado anteriormente que b en la ruta T.

Propiedad 1

Para toda ruta factible T del PDP con un sólo vehículo con sistemas de descarga LIFO, la relación *ant* definida en el apartado anterior verifica ser una relación de orden:

Demostración: Trivial

Definición y Notaciones

Se define una cadena C como una secuencia de arcos de la forma: (a,b), (b,c), (c,d), ...,(m,n).

Considérese en cada vértice del árbol de búsqueda del algoritmo original el conjunto de arcos seleccionados para entrar en la solución del problema anterior. Se denota por C(r) a la cadena formada por ese conjunto de arcos que contiene al nodo r. A la concatenación de las cadenas C_1 y C_2 lo denotamos por (C_1, C_2). Sea T una ruta cerrada o tour, se denota T ∋ C si la cadena C forma parte de ella.

Propiedad 2

Sea T una ruta factible y *ant* la relación anteriormente definida y a,b dos nodos cualesquiera, con C(a) y C(b) T entonces, a *ant* b si se verifican una de las siguientes condiciones:

- C(a) = C(b); C(a) C(1) y a aparece antes que b en C(a); (*i*)
- C(a) = C(1); C(a) C(b) y a aparece después que 1 en C(a); (*ii*)
- C(b) = C(1); C(a) C(b) y b aparece antes que 1 en C(b); (*iii*)
- C(a) = C(b) = C(1) y:

o bien a aparece después que 1 y b antes,
o bien a y b aparecen antes que 1 y a antes que b;
o bien a y b aparecen después que 1 y a antes que b. (*iv*)

Demostración.- Trivial

Los dos siguientes teoremas generalizan, extienden y corrigen los teoremas 2 y 3 propuestos por Kalantari en (1.985 pp.377-386):

Teorema 1

Sea T una ruta factible, y *ant* la relación anteriormente definida, entonces se verifican las siguientes propiedades:

a) $\forall\ a,b \in N^{+} \cup N^{-}$ a *ant* b entonces, (1,b), (b,a), (a,1) $\notin$ T.

b) $\forall\ a,b,c,d \in N^{+} \cup N^{-}$ a *ant* b, c *ant* d, con a $\neq$ c, d, b $\neq$ c, d:

- Si (a,d) $\in$ T entonces (c,b), (b,c), (1,a) (d,1) $\notin$ T; (*b1*)
- Si (b,c) $\in$ T entonces (a,d), (d,a), (c,a) (d,b) $\notin$ T; (*b2*)
- Si (a,c) $\in$ T entonces (d,a) $\notin$ T; (*b3*)
- Si (b,d) $\in$ T entonces (d,a) $\notin$ T. (*b4*)

c) $\forall\ a,b,c \in N^{+} \cup N^{-}$ / a *ant* b, c *ant* b, y a $\neq$ c:

- Si (a,b) $\in$ T entonces (1,a) $\notin$ T.

d) $\forall\ a,b,d \in N^{+} \cup N^{-}$/ a ant b, a ant d, y b $\neq$ d:

- Si (a,d) $\in$ T entonces (d,1) $\notin$ T

e) $\forall\ a,b,d \in N^{+} \cup N^{-}$/ a *ant* b, b *ant* d :

- (a,d) $\notin$ T; (*e1*)
- Si (a,b) $\in$ T entonces (d,a) $\notin$ T; (*e2*)
- Si (b,d) $\in$ T entonces (d,a) $\notin$ T. (*e3*)

Demostración

La demostración del apartado a), como se puede observar, es trivial ya que la presencia del arco (1,b) hace que b sea el primer nodo visitado, después del origen lo que contradice que a *ant* b, análogamente se puede demostrar la no existencia de los arcos (b,a) y (a,1). En el caso del apartado (*b1*) vale observar que la existencia simultánea del arco (a,d) con cualquiera de los siguientes: (c,b), (b,c), (1,a) (d,1), contradice al menos una de las dos relaciones a *ant* b o c *ant* d. De igual forma se demuestran los demás apartados.

Teorema 2

Sea T una ruta factible, y *ant* la relación anteriormente definida, entonces se verifican las siguientes propiedades:

a) $\forall\ a,b \in N^+ \cup N^-/$ a *ant* b:

- Si C(a), C(b) ≠ C(1) entonces (C(b),C(a)) ∉ T; (a1)
- Si C(1), C(b) ≠ C(a) entonces (C(1),C(b)) ∉ T; (a2)
- Si C(1), C(a) ≠ C(b) entonces (C(a),C(1)) ∉ T. (a3)

b) $\forall\ a,b,c,d \in N^- \cup N^-/$ a *ant* b, c *ant* d, con a ≠ c, d, b ≠ c, d :

- Si C(a) = C(d); C(a), C(b), C(c) ≠ C(1); C(a) ≠ C(b) y C(c) ≠ C(d), entonces (C(b),C(c)), (C(c),C(b)) ∉ T.

c) $\forall\ a,b,d \in N^+ \cup N^-/$ a *ant* b y b *ant* d:

- Si C(a), C(b) ≠ C(d) y C(a) C(b) entonces (C(a), C(d)) ∉ T.

Demostración.- Trivial a partir de la definición de la relación *ant* y del teorema 1.

Propiedad 3

Sea T una ruta factible entonces:

a) i^+ *ant* i^-, $i = 2,...,n$;
b) $i, j = 2,...,n$, si i^+ *ant* j^+ y j^+ *ant* i^- entonces j^- *ant* i^- ;
c) $i, j = 2,...,n$; si i^- *ant* j^- y j^+ *ant* i^- entonces j^+ *ant* i^+.

Demostración

El apartado a) es trivial: para cada cliente la carga debe preceder siempre a la descarga. Para demostrar los apartados b) y c) vale comprobar que la siguiente situación no es factible:

$$1 - .... - i^+ - .... - j^+ - .... - i^- - .... - j^- - .... - 1;$$

ya que en el momento en que se encuentren cargadas en el vehículo dos o más mercancías se ha de descargar antes la que se ha cargado más recientemente.-

Propiedad 4

Sea T una ruta factible entonces $(i^+,j^-) \notin T$, $\forall\ i, j = 2,..., n; i \neq j$.

Demostración

No se puede descargar la mercancía de un cliente inmediatamente después de cargar la de otro.

2.2. Descripción básica de los filtros

Estos teoremas y propiedades anteriores junto con la definición de la relación *ant* son la base del desarrollo de los filtros que se exponen a continuación: El primer filtro, que se va a insertar en el procedimiento principal, actua como sigue:

1.- Impide la elección de los arcos (i^+,j^-), (Propiedad 4), redefiniendo la matriz de distancias

2.- Registran las relaciones de precedencia determinadas por el apartado a) de la Propiedad 3.

El segundo filtro, que se inserta en la exploración de cada vértice, actúa de la forma siguiente:

1.- Determina las posiciones relativas de los diferentes nodos según los arcos seleccionados, para ello utiliza la Propiedad 2.
2.- A partir de estas posiciones relativas *Detecta* y registra todas las precedencias surgidas al aplicar los apartados b) y c) de la Propiedad 3 y la propiedad transitiva de la relación *ant*.
3.- Examina si el conjunto de precedencias obtenido verifica las propiedades antisimétrica y transitiva. Si no es así se interrumpe la exploración de ese vértice.
4.- Finalmente, con las precedencias registradas, se utilizan los Teorema 2 y 3 para impedir la elección de arcos que no pueden estar incluidos en una solución factible.

3. Variación en la elección de nuevos arcos

Además de la incorporación de los procedimientos de filtrado se propone una pequeña modificación en la elección de los arcos que se añaden a la solución, en cada vértice del árbol de búsqueda, con el objeto de aumentar la eficacia de los filtros en exploraciones posteriores. La idea básica es sencilla: se restringe la búsqueda de este conjunto de arcos únicamente a los que finalicen en el primer nodo de la cadena donde esta el nodo 1 y a los arcos que comienzan en el último nodo de dicha cadena.

Obsérvese que de esta manera solamente se forma una cadena en las diferentes exploraciones, con el consiguiente ahorro de variables y de código de programa.

4. Resultados computacionales

En este apartado se van a mostrar los efectos que produce en el tiempo de computación la incorporación de los procedimientos de filtrado y de elección de los nuevos arcos en la resolución del PDP con un sólo cliente en sistemas de descarga LIFO. Para ello se han simulado una serie de matrices de distancias correspondientes a problemas con 5, 6, 7 y 8, clientes (20 para cada número de clientes). Los valores de las matrices de distancias se han generando aleatoriamente de forma uniforme entre 0 y 100. Cada problema es resuelto de tres formas o variantes diferentes según la incorporación de los diferentes procedimientos descritos en el apartado anterior. Estas tres variantes son las siguientes:

VARIANTE A: En esta variante se incorpora el primer filtro al algoritmo Branch & Bound original. Cuando se llega a una solución se comprueba si es factible en cuyo caso se registra.
VARIANTE B: En esta variante se incorporan los dos filtros.
VARIANTE C: En esta variante se incorporan todos las procedimientos descritos en el apartado anterior: los dos filtros y la modificación en la elección de arco o arcos que entran en la solución.

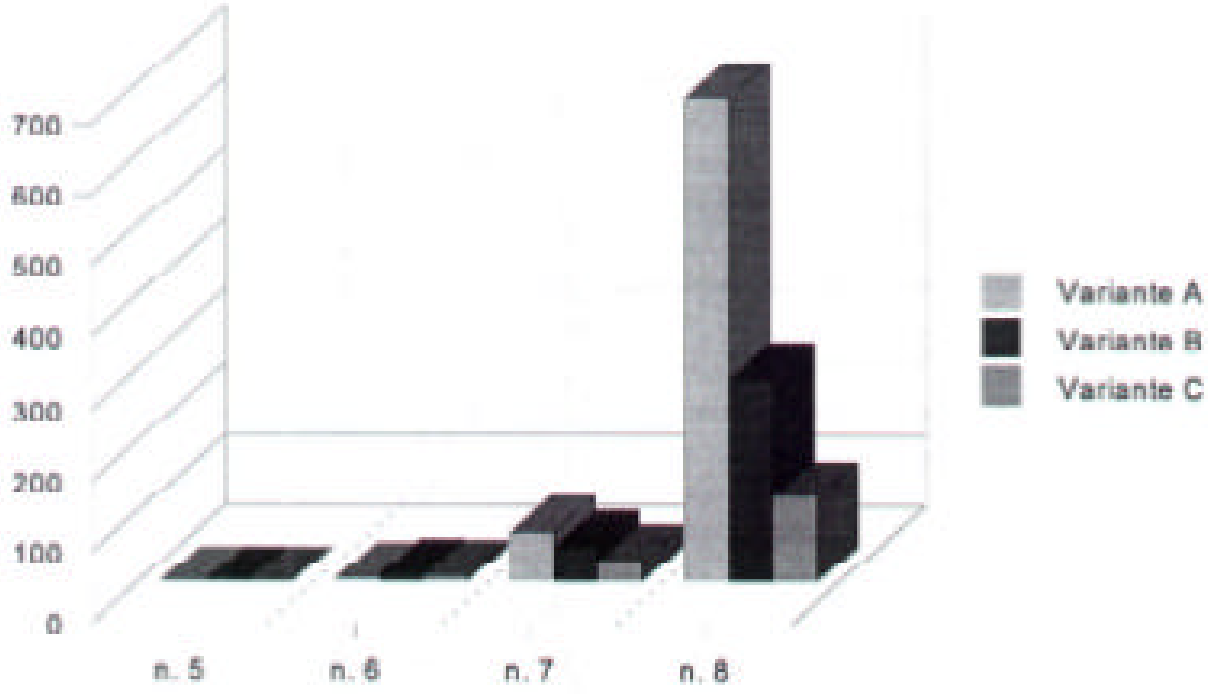
Para la implementación y ejecución de estos problemas con estos algoritmos se ha utilizado el compilador BORLAND PASCAL 7.0. El equipo utilizado es un ordenador personal de tipo PC AT i486-dx2 a 50 Mhz. A continuación se muestran los tiempos medios de computación expresados en segundos empleados por cada variante, así como los tiempos máximo y mínimo, según el tamaño del problema:

Número de clientes		5(11 nod)	6(13 n.)	7(15 n.)	8 (17 n.)
Variante A	Mínimo	0,06	0,05	0,49	5,60
	Medio	0,28	10,2265	67,7315	682,8215
	Máximo	1,05	118,09	638,45	4668,23
Variante B	Mínimo	0,05	0,17	0,22	3,19
	Medio	0,357	7,3005	44,716	282,507
	Máximo	1,04	59,59	360,92	2898,64
Variante C	Mínimo	0,06	0,28	0,77	4,34
	Medio	0,5625	2,752	19,817	116,3115
	Máximo	2,26	10,93	135,17	692,17

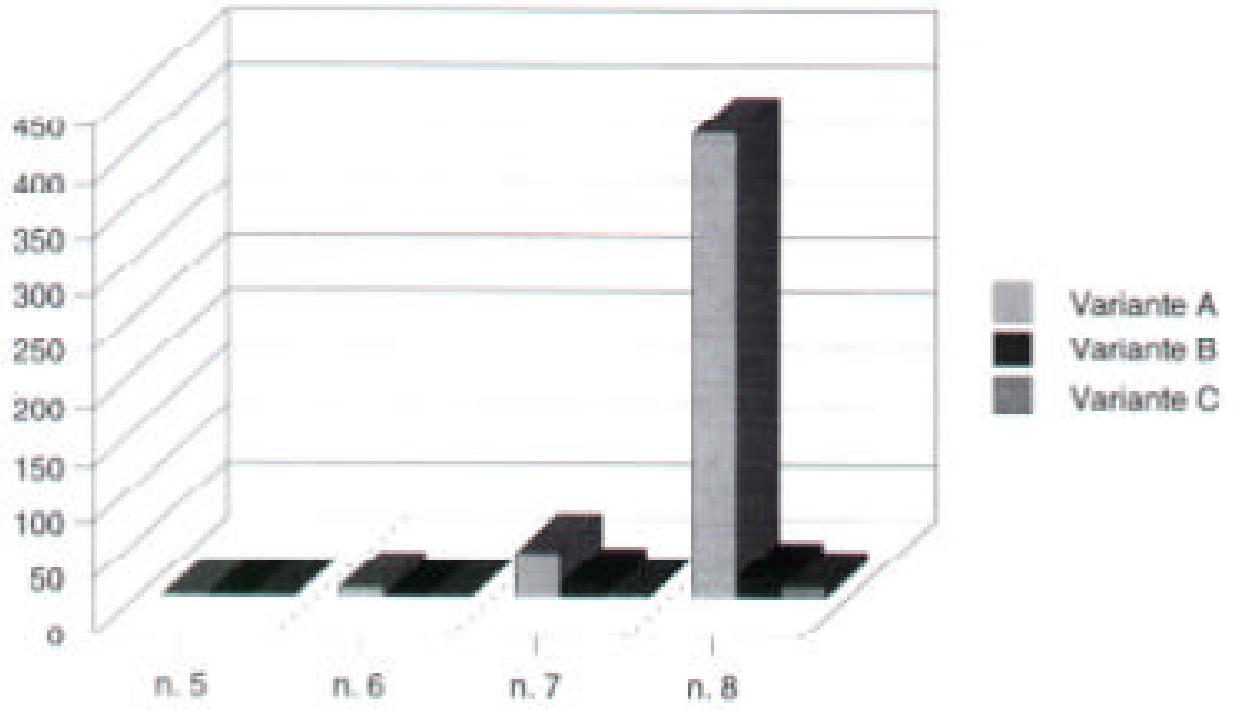
De igual forma, en el siguiente cuadro se muestra la evolución del número de exploraciones empleadas por cada variante, (media, mínima y máxima), según el tamaño del problema:

Número de clientes		5(11 nod)	6(13 n.)	7(15 n.)	8 (17 n.)
Variante A	Mínimo	0,06	0,05	0,49	5,60
	Medio	0,28	10,2265	67,7315	682,8215
	Máximo	1,05	118,09	638,45	4668,23
Variante B	Mínimo	0,05	0,17	0,22	3,19
	Medio	0,357	7,3005	44,716	282,507
	Máximo	1,04	59,59	360,92	2898,64
Variante C	Mínimo	0,06	0,28	0,77	4,34
	Medio	0,5625	2,752	19,817	116,3115
	Máximo	2,26	10,93	135,17	692,17

Los siguientes gráficos muestran más claramente la evolución de los tiempos y número de exploraciones medios



Tiempo de Computación (segundos) según el número de clientes



Número de exploraciones (en miles) en el árbol de búsqueda según el número de clientes

5. Conclusiones, extensiones y mejoras

A la vista de estos resultados las conclusiones que se pueden hacer son las siguientes:

- Comparando los tiempos de computación de las variantes A y B se observa que se reduce tiempo de computación con esta segunda; sin embargo esta reducción aunque importante no es suficiente y en algunos casos el tiempo de computación empleado por esta variante B sigue siendo excesivo (obsérvese la evolución de los máximos de los tiempos y exploraciones empleados); por consiguiente, aparentemente, el efecto del segundo filtro es pobre o poco eficaz, en el sentido de no que siempre evita la selección de arcos que vayan a dar lugar a soluciones infactibles, y por tanto se realizan muchas exploraciones innecesarias. La razón es la siguiente: dicho procedimiento detecta y registra relaciones de precedencia a partir de los apartados b) y c) de la Propiedad 3. Para aplicar estos apartados hace falta conocer las posiciones relativas de los diferentes pares de puntos; sin embargo cuando se forman varias cadenas no siempre es posible determinar dichas posiciones relativas, (ver Propiedad 2), con lo cual sólo se pueden registrar pocas relaciones de precedencia y por consiguiente se detectan pocos arcos que no pueden ser seleccionados en pasos anteriores.
- Este defecto se compensa con la modificación del proceso de elección de los nuevos arcos, variante C, que hace que solo haya una cadena en las diferentes exploraciones. por tanto, siempre se puede conocer la posición relativa de cada

par de puntos sólo con que alguno de los dos pertenezca a uno de los arcos seleccionados en pasos anteriores. De esta forma se puede aplicar en muchos más casos los apartados b) y c) de la Propiedad 3, con lo que ya se hace *rentable* la utilización del segundo filtro.

- Como consecuencia de lo anterior la variante C resulta en conjunto la más eficaz: el tiempo de computación es mayor que en la variante B para pocos clientes, sin embargo a medida que aumenta el tamaño del problema el tiempo de computación de la variante C aumenta mucho más lentamente que en las otras. Si se observa además, el número de exploraciones realizadas por esta variante realizadas es aún mucho menor que el empleado por las otras dos: se evitan realizar muchas más exploraciones innecesarias que con las otras dos variantes.

Estas variantes, como se comentó al principio de este trabajo, no consideran las restricciones de carga en el problema, sin embargo la formación de una sola cadena entorno al nodo inicial 1, hace que esta restricción se pueda incorporar fácilmente a la variante C. En la exploración de cada vértice vale añadir un pequeño procedimiento que, comenzando en el nodo 1, calcule en cada paso el espacio libre que queda en el vehículo, -tanto hacia adelante como hacia atrás-, de la siguiente forma: se comienza en el nodo 1 con el valor de la capacidad del vehículo; en caso del cálculo hacia adelante q(i) se resta si se visita i^+ y se suma si se visita i^- ; en caso del cálculo hacia atrás la operación se hace al revés. A continuación, dependiendo del espacio libre calculado en cada caso, se impide la elección de los arcos que añadan nodos a la cadena que vayan a dar lugar a espacios libres negativos en pasos posteriores.

En cualquier caso es necesario remarcar que lo propuesto en este trabajo no es un algoritmo exacto concreto para el problema planteado, sino una estrategia o técnica para desarrollar estos a partir de cualquiera de los algoritmos exactos para el ATSP, no solamente el algoritmo de Little y otros, (1.963 pp.972-989). El tiempo de computación empleado dependerá también del algoritmo para el ATSP de partida, y de las mejoras que este tenga, como por ejemplo: partir de la solución inicial propuesta por algún heurístico, o generar cotas superiores en cada vértice del árbol de búsqueda.

6. Apendice

6.1. El algoritmo de Little

El algoritmo de Little es una técnica Branch & Bound basado en los dos siguientes teoremas:

Teorema A.

Sea T cualquier ruta factible del TSP conteniendo el conjunto de arcos R(α) y que no contiene ninguno de los arcos del conjunto F(α); sea D1 = ($d1_{ij}$) la matriz resultante de hacer infinito los elementos d_{ij} con (i,j) - F(α);

se definen además:

filas y *colum* los conjunto de nodos que resultan de eliminar en el conjunto de nodos original respectivamente los nodos que son comienzo y final de los arcos de R(α);

$$e_i=\min_{j\in colum} dl_{ij}, i \in filas; f_i=\min_{i\in filas} (dl_{ij}-e_i), j \in colum$$

entonces:

$$Z(T)=\sum_{(i,j)\in R(\alpha)} d_{ij} + \sum_{i\in filas} e_i + \sum_{j\in colum} f_j = h_0$$

siendo Z(T) la distancia total de la ruta T.

De esta forma h_0 es una cota inferior del conjunto de soluciones conteniendo los arcos R(a) y excluyendo los arcos F(a). A la matriz D* = ($d1_{ij}$ - e - f), con , se la denomina matriz reducida.

Teorema B.

Sea T cualquier ruta factible del TSP conteniendo los arcos de R(α) y no incluyendo ninguno del conjunto F(α) ni el (i,j), donde i y j son respectivamente fila y columna de la matriz D*, entonces:

$$Z(T)\geq h_0 + p_{ij},$$

donde $p_{ij}=\min_{k\neq j} d*_{ik} + \min_{l\neq i} d*_{lj}.$

A p_{ij} se le denomina *penalización* por no incluir el arco (i,j) en pasos posteriores.

El algoritmo actúa básicamente como sigue: en cada vértice del árbol de búsqueda se reduce la matriz D1 obteniéndose una cota inferior para este vértice. Para los arcos correspondientes a los 0 de matriz reducida D* se calculan sus respectivas penalizaciones, y se elige aquel con mayor penalización. Sea (i,j) el arco elegido, el conjunto de soluciones del vértice se ramifica o divide en dos partes: las que contienen el arco (i,j) y las que no lo contienen.Una vez hecha la elección del arco (i,j) se ha de prevenir la elección de arcos que en pasos posteriores den lugar a ciclos. Se vuelve a repetir el proceso hasta la obtención de una solución y se exploran las ramas que quedan sin examinar.

6.2. Descripción en seudocódigo de todos los procedimientos utilizados

Para la descripción de los filtros se define inicialmente el siguiente tipo de variables:

PREFERENCIA: Tipo de variables con dos componentes, Inicio y Final, de tipo entero;

a continuación se definen las siguientes variables:
vectorpref: vector donde se van a ir guardando las preferencias que se van detectando;
numorden: vector donde se guarda el orden de visita que ocupa cada nodo en la ruta parcial;
prefpres y presente: matrices lógicas auxiliares; prefpres(i,j) toma el valor TRUE si se detecta la preferencia i *ant* j en cada rama y FALSE en caso contrario; presente(i,j) toma el valor TRUE si el arco (i,j) se encuentra en la ruta parcial y FALSE en caso contrario;
kpref: contador que señala el número de preferencias detectadas;
primero y ultimo: vectores enteros auxiliares; primero(i) y ultimo (i) indican respectivamente el primer y ultimo elemento de la cadena C(i); en la exploración de cada vértice α se consideran además los siguientes parámetros de entrada:

$R(\alpha)$: Conjunto de arcos seleccionados para formar parte de lo solución;
$F(\alpha)$: Conjunto de arcos que no forman pueden formar parte de ella;

finalmente se describen ambos procedimientos que denominaremos FILTROGENERAL y FILTROENRAMA:

Procedimiento FILTROGENERAL.

Paso 1.: Hacer $d_{1j} = \infty$, $d_{i1} = \infty$, para i = 2,3,..,n;
Hacer $d_{ij} = \infty$, para i, j = 2,3,...,n, i ≠ j;

Paso 2.: Hacer *vectorpref*(*i*-1) = (i^+, i^-), para i = 2, 3,...,n.

Procedimiento FILTROENRAMA.

Paso 0.: (Inicializar variables)
Hacer *presente*(k,k1) = FALSE, *prefpres* k,k1) =
= FALSE, $\forall k, kl \in N^+ \cup N^-$, $k \neq kl$;
Hacer primero(k) como el primer elemento de C(k), $\forall k, \in N^+ \cup N^- \cup \{1\}$;
Hacer ultimo(k) como el último elemento de C(k), $\forall k, \in N^+ \cup N^- \cup \{1\}$;
Hacer *presente*(i,j) = TRUE, $\forall (i,j) \in R(\alpha)$.

Paso 1.: (Preferencias iniciales)
Hacer *kpref* = n-1;
Para i, j = 2, 3,..., n; i ≠ j:
Si i^+ *ant* j^+ y j^+ *ant* i^- entonces:
hacer *kpref* = *kpref*+1,
hacer *vectorpref*(*kpref*) = (j^-, i^-);
Si i^- *ant* j^- y j^+ *ant* i^- entonces:
hacer *kpref* = *kpref*+1,
hacer *vectorpref*(*kpref*) = (j^+, i^+);
Hacer *prefpres*(*vectorpref*(k).*inicio*, *vectorpref*(k).*final*) = TRUE, para k = 1,..., kpref.

Paso 2.: (Propiedad transitiva)
Hacer *kpref1* = *kpref*;
Para k, k1 = 1, 2,..., kpref, k ≠ k1, hacer:
a = *vectorpref*(k).*inicio*, b = *vectorpref*(k).*final*,
c = *vectorpref*(k1).*inicio*, d = *vectorpref*(k1).*final*,
Si (b = c) y (no prefpres(a,d)) entonces:
kpref = *kpref*+1,
vectorpref(*kpref*) = (a,d),
prefpres(a,d) = TRUE;
Si *kpref* > *kpref1* volver al comienzo de este paso,
sino ir al paso siguiente.

Paso 3.: (Compatibilidad de preferencias)
prefcompat = TRUE;
Hacer *prefcompat* = *prefcompat* and (not *prefpres*(*i*,*i*)), para $i \in N^+ \cup N^-$,
Si *prefcompat* entonces ir al paso siguiente;
sino finalizar.

Paso 4.: (Filtrado - Teorema 1)
Para k = 1,2,...,*kpref*, hacer:
a = *vectorpref*(k).*inicio*, b = *vectorpref*(k).*final*,
$dl_{ib} = \infty$, $dl_{al}{}^{-} \infty$, $dl_{ba} = \infty$,
Para k1 = 1,2,...,*kpref*, k ≠ k1, hacer:
c = *vectorpref*(k).*inicio*, d = *vectorpref*(k1).*final*,
Si *presente*(a,d) entonces
$dl_{cb} = \infty$, $dl_{bc} = \infty$,
si a ≠ c entonces $dl_{ta} = \infty$,
si b ≠ d entonces $dl_{dl} = \infty$,

si b = c entonces *prefcompat* = FALSE,

Si *presente*(b,c) entonces

$dl_{cb} = \infty,\ dl_{da} = \infty,\ dl_{ca} = \infty, dl_{db} = \infty$,

Si *presente*(a,c) entonces $dl_{da} = \infty$

Si *presente*(b,d) entonces $dl_{da} = \infty$.

Paso 5.: (Filtrado - Teorema 2)

Para k = 1,2,...,*kpref*, hacer:

a = *vectorpref*(k).*inicio*, b = *vectorpref*(k).*final*,

Si C(a), C(b) ≠ C(1), entonces $dl_{ultimo(b),primero(a)} = \infty$;

Si C(1), C(b) ≠ C(a), entonces $dl_{ultimo(1),primero(b)} = \infty$;

Si C(1), C(a) ≠ C(b), entonces $dl_{ultimo(a),primero(1)} = \infty$;

Para k1 = 1,2,...,*kpref*, k ≠ k1, hacer:

c = *vectorpref*(k).*inicio*, d = *vectorpref*(k1).*final*,

Si a ≠ c,d, b ≠ c,d, C(a) = C(d), C(a),C(b),C(c) ≠ C(1),

C(a) ≠ C(b) y C(c) ≠ C(d), entonces

$dl_{ultimo(b),primero(c)} = \infty$, $dl_{ultimo(c),primero(b)} = \infty$;

Si b = c, C(a),C(b) ≠ C(d), C(a) ≠ C(b), entonces:

$dl_{ultimo(a),primero(d)} = \infty$.

Paso 6.: Finalizar.

Para la descripción de la exploración en cada vértice y del algoritmo principal, una vez insertados los procedimientos filtrados, se definen las siguientes variables:

Distminima: Distancia de la mejor solución obtenida hasta el momento;
Soluc: Conjuntos de arcos que forman dicha solución;
Disttotal: Distancia de la solución que se halla en cada momento.

Procedimiento EXPLORACION(R(α),F(α))

Si los arcos de R(a) forman una ruta:

Poner *Disttotal* como la suma de las distancias de los arcos que componen R(α);

Si *Disttotal* < *Distminima* y los arcos de R(α) forman una ruta factible, entonces hacer:

Distminima = *Disttotal*;

Soluc = R(α);

en caso contrario, hacer:

Reconocer y Registrar las cadenas de R(α); (1)

Ejecutar FILTROENRAMA; (2)

Poner $dl_{ii} = \infty, \forall (i,j) \in F(\alpha)$;

Reducir la matriz D1

Determinar h_0 (Teorema A)

Si h_0 < *Distminima* y *preferenciascompatibles* entonces hacer:

Seleccionar (i*,j*) el arco con mayor penalización $p_{i^*j^*}$ (Teorema B); (3)

Poner R(α*) = P(α) ∪ (ι*,φ*);

Si R(α) ∪ (i*,j*) no forman una ruta entonces:

Poner F(α*) = F(α) ∪ (ultimo(j*),primero(i*)))

en caso contrario Poner F(α*) = F(α);

Ejecutar EXPLORACION(R(α*), F(α*));

$h_0 + p_{i^*j^*}$ < *Distminima* entonces, hacer

R(α**) = R(α);

F(α**) = F(α) ∪ (ι*,φ*);

Ejecutar EXPLORACION(R(α**), F(α**)).

Finalizar.

Procedimiento PRINCIPAL

Leer datos iniciales: n y D

Hacer *Distminima* =∞.

Ejecutar FILTROGENERAL (1)

Ejecutar EXPLORACION(∅,∅)

Para describir el nuevo procedimiento de elección del arco (i*,j*) se define la variable auxiliar

penal : variable donde se guarda el mayor valor de las penalizaciones de los ceros encontrados;

los pasos que lo componen son los siguientes:

PROCEDIMIENTO NUEVAELECCION;

Paso 1.: *penal* =-∞.

Paso 2.: $\forall i \in$ filas, si $d^*_{i,primero(1)} = 0$ entonces hacer:

si $p_{i,primero(1)}$ > *penal* entonces hacer:

$i^* = i$
$j^* = primero(i)$.

Paso 3.: $\forall j \in$ colum si $d^*_{ultimo(1),j}$, entonces hacer:

si $p_{ultimo(1),j} > penal$ entonces hacer:
$penal = p_{ultimo(1),j}$,
$j^* = j$
$i^* = ultimo(i)$.

Paso 4.: Fin.

Por consiguiente las variantes A, B, y C utilizadas actuan de la forma siguiente:

VARIANTE A: En esta variante se incorpora el procedimiento FILTROGENERAL, es decir en el procedimiento EXPLORACION(R(α),F(α)) no se ejecutan las líneas (*1*) y (*2*), pero en el procedimiento PRINCIPAL si se ejecuta la línea (*1*).

VARIANTE B: En esta variante se incorporan todos los filtros, es decir en el procedimiento EXPLORACION(R(α),F(α)) se ejecutan las líneas (*1*) y (*2*), y en el procedimiento PRINCIPAL se ejecuta la línea (*1*).

VARIANTE C: En esta variante se incorporan todos las procedimientos descritos en el apartado 2 y 3; por consiguiente, es igual que la VARIANTE C salvo que se cambia la línea (*3*) del procedimiento EXPLORACION (R(α), F(α)) por: Ejecutar NUEVA ELECCION.

7. Bibliografía

1.-ASSAD,A.A.(1.988): "Modeling and Implemantation Issues in Vehicle Routing". In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., Nort-Holland.

2.-DESROCHERS,M. and SOUMIS,F., (1.991): "*Implantation el Complexité des Techniques de Programmation Dynamique dans les Méthodes de Confection des Tournées et d'Horaires*". Recherche Operationnelle/Operations Researsch, vol.25, nº 3, 291-310.

3.-DESROSIERS,J., DUMAS,Y., and SOUMIS,F., (1.986): "*A Dynamic Programming Solution of the Large-Scale Single-Vehicle Dial-a-Ride Problem with Time Windows*". Amer. J. Math. Management Sci. 6, 301-326.

4.-JAW,J.J., ODONI,A.R., PSARAFTIS,H.N. and WILSON,N.H.M., (1.986): "*A Heuristic Algorithm for the Multi-Vehicle Advance Request Dial-a-Ride Problem with Time Windows*". Transp.Res.B, vol.20b, nº 3, 243-257.

5.-KALANTARI,B., HILL,A.V. and ARORA,S.R.,(1.985): "*An Algorithm for the Traveling Salesman Problem with Pickup and Delevery Customers*". European Journal of Operational Research, 22, 377-386.

6.-LITTLE,J., MURTY,K, SWEENEY,D. and KAREL,C.), (1.963): "An Algorithm for the Traveling Salesman Problem". Operations Researsch 11 (5), 972-989.

7.- PACHECO,J., GARCIA,A. y ARAGON,A., (1.994): "*Problemas de Ruta con Carga y Descarga en Sistemas LIFO*". XXI Congreso de Estadística e I.Optva celebrado en Cálella en Abril de 1.994.

8.- PACHECO,J., (1.994): "*Problemas de Rutas con Ventanas de Tiempo*". Tesis Doctoral leída en el Dpto de Estadística e I.Optva de la Facultad de Matemáticas de la U.Complutense de Madrid. Mayo 1.994. pp. 184-201.

9.-PSARAFTIS,H., (1.983): "*An Exact Algorithm for the Single Vehicle Many-to-Many Dial-a-Ride Problem with Time Windows*". Transportation Sci.,17, 351-360.