QÜESTIIÓ, vol 21, 1 i 2, p. 153-175, 1997

HEURÍSTICO PARA LOS PROBLEMAS DE RUTAS CON CARGA Y DESCARGA EN SISTEMAS LIFO

JOAQUÍN A. PACHECO*
E.U.E. Empresariales de Burgos

En este trabajo se propone un algoritmo heurístico 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—. Este algoritmo es una extensión y adaptación del método de Or para el Problema del Viajante (TSP) que sirve también para matrices asimétricas. Con este heurístico se consiguen resolver problemas de gran tamaño en un tiempo de computación razonable en ordenadores personales, con una desviación del óptimo muy pequeña.

Heuristic for the pick-up and delivery routing problems in lifo unloading systems

Keywords: Carga y descarga con sistema LIFO; intercambios de Or; chequeo de factibilidad.

* Joaquín A. Pacheco. Dpto. Matemáticas. E.U.E. Empresariales de Burgos. Fco. Vitoria s/n. 09006 Burgos.

– Article rebut el març de 1995.

– Acceptat el juliol de 1996.

1. INTRODUCCIÓN

El Problema de Carga y Descarga (PDP) puede ser descrito de la forma siguiente: Sea un conjunto de clientes $N = \{2,3,\ldots,n\}$; cada cliente r requiere que le sea transportada una mercancía desde un origen r^+ a un destino r^-. 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 $N^+ = \{r^+/r = 2,\ldots,n\}$, el conjunto de puntos de carga y $N^- = \{r^-/r = 2,\ldots,n\}$, el conjunto de puntos de descarga, y $q(r)$, $r = 2,\ldots,n$, la mercancía de cada cliente r. 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.

El PDP y el PDP con Ventanas de Tiempo (PDPTW) son problemas de rutas muy estudiados, especialmente a partir de la segunda mitad de la década de los ochenta. Un repaso a los principales métodos de solución a estos problemas, tanto exactos como heurísticos, se puede encontrar en el trabajo de Pacheco, (1994).

Entre los algoritmos exactos destacan inicialmente el de Kalantari y otros, (1985), para el PDP con un solo vehículo sin restricciones de capacidad, que proponen una serie de modificaciones en el algoritmo de Little y otros, (1963), para el TSP, con el fin de impedir la selección de arcos incompatibles con las relaciones de precedencia *origen-destino* (r^+ anterior a r^-); Psaraftis, (1983), usa programación dinámica para resolver el DARP con Ventanas de Tiempo (DARPTW) con un solo vehículo. Los estados son de la forma $(j,k_2,\ldots,k_n)$, 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: -1, no ha sido cargada; 0, ha sido cargada pero no descargada; y $+1$, ha sido descargada; Desrosiers y otros, (1986), adaptan este algoritmo de $2n$ estados, a PDPTW con un solo vehículo. Dumas, (1985), y Desrosiers y otros, (1987), presentan un algoritmo para el PDPTW planteándolo como un problema de Partición de Conjuntos y adaptando las técnicas de solución de éste.

En cuanto a los algoritmos heurísticos Jaw y otros, (1986), 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 otras técnicas heurísticas para problemas con varios vehículos, como en el caso de las del tipo *Cluster* $1^o - $*Ruta* 2^a. Ejemplos de este tipo de aproximaciones los tenemos en los trabajos de Desrochers y otros, (1991).

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 y se contempla la situación de la mercancía en el vehículo.

Existen referencias de trabajos que tratan modelos que también tienen en cuenta la situación de la carga en el vehículo, principalmente el VRPB, (Vehicle Routing Problem with Backhauls), como Casco y otros, (1988), Deif y otros, (1984), Goetschalkx y otros, (1986), y Golden y otros, (1985). Sin embargo, el problema tal como se ha definido en el párrafo anterior, sólo tiene referencias en los trabajos de Pacheco, (1994), y Pacheco y otros, (1994). En éstos se propone una estrategia para desarrollar algoritmos exactos a este problema, consistente en, a partir de un algoritmo Branch & Bound para el TSP, introducir una serie de modificaciones que den lugar a un algoritmo exacto para el problema anterior. Sin embargo, se ha observado que para problemas de mayor tamaño a los utilizados en estos trabajos (10 o más clientes), el tiempo de computación en ordenadores personales deja de ser razonable, cuando se utilizan estos algoritmos u otros algoritmos exactos (como los basados en métodos de programación dinámica).

En este trabajo, se propone un heurístico capaz de resolver problemas de gran tamaño en un tiempo de computación moderado, desviándose del óptimo sólo en un pequeño número de casos y en una cantidad mínima. Esta técnica es una extensión y adaptación a este problema del algoritmo de Or para el TSP, (1976). El algoritmo de Or es una variante de los conocidos algoritmos $r-$óptimos desarrollados por Lin, (1965), para el TSP simétrico. En el caso del algoritmo de Or se toma $r = 3$ y se reduce la búsqueda de los 3-intercambios a aquellos que supongan una recolocación de cadenas de 1, 2 o 3 elementos entre otros dos consecutivos en la ruta actual. La ventaja de este método es que el número de operaciones que se realiza es de $\theta(m^2)$, frente al algoritmo 3-óptimo original que utiliza $\theta(m^3)$, (siendo m el número de puntos que intervienen en el problema). Además, como se verá mas adelante, el algoritmo de Or se puede adaptar fácilmente a problemas asimétricos. La eficacia de este algoritmo ha sido contrastada recientemente en el trabajo de Nurmi, (1991). Adaptaciones de este algoritmo al TSP con ventanas de tiempo (TSPTW) se pueden encontrar en el trabajo de Salomon y otros, (1988).

El trabajo se estructura de la siguiente forma: en la sección 2 se explica la idea básica del algoritmo que se quiere desarrollar; en la sección 3 se estudian las condiciones para que un 3-intercambio, en este caso recolocación, sea factible; en la sección 4 se estudian las situaciones posibles dependiendo de los elementos que componen la cadena que se quiere recolocar; en la sección 5 se describe el algoritmo —desarrollado a partir de los resultados descritos en los apartados anteriores—; en la sección 6 se chequea la eficacia de este algoritmo a partir de la resolución de una

serie de problemas simulados; en la sección 7 se expondrán las conclusiones que se extraen de los resultados obtenidos en la sección anterior.

Igual que en trabajos anteriores, —Pacheco, (1994), y Pacheco y otros, (1994),— se supondrá que la capacidad del vehículo es mayor que la carga total del conjunto de clientes; el caso general se estudiará posteriormente.

2. IDEA BÁSICA

Or, (1976), propone restringir la búsqueda de intercambios a los 3-intercambios en los que cadenas de uno, dos o tres clientes consecutivos son recolocadas entre otros dos clientes (ver figura 1). Esto hace reducir el número de 3-intercambios a considerar a una cantidad de orden $\theta(n^2)$. Nótese que con estos intercambios no se cambia el sentido de los diferentes tramos.

En nuestro caso seguiremos la misma idea, pero no el límite del tamaño de la cadena a recolocar; además debemos chequear la factibilidad de cada posible recolocación respecto a las restricciones del problema.

Figura 1. Posible recolocación del elemento i hacia adelante y hacia atrás entre j y $j+1$.

En el caso general de recolocación de cadenas de k elementos consecutivos, comenzando en la posición i, entre dos puntos que ocupan las posiciones j y $j+1$, como

muestra la figura 2, supone suprimir los arcos $(i-1,i)$, $(i+k-1,i+k)$, $(j,j+1)$ e incorporar los arcos $(i-1,i+k)$, (j,i) y $(i+k-1,j+1)$.

Figura 2. Recolocación de una cadena de k elementos.

En nuestro caso la ruta contiene $2 \cdot n$ puntos: el origen 1, $n-1$ puntos de carga, $n-1$ puntos de descarga, y además, por comodidad, se añade al final el punto 1 que se interpreta como la vuelta al origen. Se definen las siguientes variables: k, el tamaño de la cadena a recolocar; i el punto inicial de la cadena; y j el punto tras el que recolocamos la cadena.

Para cada número de elementos k de la cadena, los posibles valores que puede tomar i varían desde 2, hasta $2 \cdot n-k$. Para cada valor de i y cada valor de k, los posibles valores que puede tomar j, punto a partir del cual insertamos la cadena, son $j = 1,\ldots,i-2$; (recolocación hacia atrás), y $j = i+k,\ldots,2 \cdot n-1$, (recolocación hacia adelante). Por comodidad, se añade a las rutas el punto $2 \cdot n$ que se interpreta como la vuelta al origen 1.

El algoritmo actúa básicamente de la forma siguiente: a partir de una ruta inicial examina todos los intercambios o recolocaciones posibles que sean factibles. Para cada uno de estos intercambios factibles calcula la variable *ganancia* definida como la suma de las distancias de los arcos que se suprimen menos la suma de las distancias de los arcos que se añaden. Se realiza el intercambio correspondiente al mayor valor de *ganancia*, y se vuelve a repetir el proceso con la nueva ruta. El algoritmo termina cuando el mayor valor de *ganancia* no sea positivo.

El chequeo de la factibilidad de cada intercambio, es decir de la nueva ruta a la que éste da lugar, puede suponer un tiempo de computación excesivo. Si para cada recolocación posible (definida por i,k y j) se chequea su factibilidad, el número de operaciones en cada iteración sería de $\Theta(n^4)$. A continuación vamos a proponer un método que rápidamente examine las recolocaciones factibles para cada cadena, es decir, para cada valor de i y de k, determinar directamente los valores de j factibles; de esta forma que el número de operaciones sea de $\theta(n^3)$.

3. INTERCAMBIOS FACTIBLES

Para examinar la factibilidad de un intercambio se deben considerar las dos restricciones, o precedencias obligatorias, siguientes del problema:

1. Para cada cliente r cada punto de carga r^+ debe ser anterior a cada punto de descarga r^-.
2. Sólo se puede descargar la última mercancía que ha sido recogida, es decir, cuando haya dos mercancías en el vehículo, se ha de descargar antes la última en entrar. En otras palabras: no se pueden dar situaciones de este tipo en una ruta
$$1 - \cdots - r^+ - \cdots - s^+ - \cdots - s^- - \cdots - r^- - \cdots - 1.$$

Cuando se estudia la recolocación de una cadena concreta, (determinada por los valores de i y k definidas anteriormente), se ha de examinar en que lugares se puede insertar, es decir qué valores de j van a dar lugar a rutas que cumplan las dos restricciones anteriores. Para estudiar los valores de j que cumplan la primera restricción, basta con fijarse por una parte en las posiciones de los puntos de carga correspondientes a los puntos de descarga que están en la cadena; y por otra parte en las posiciones de los puntos de descarga correspondientes a los puntos de carga que están en la cadena. Como se verá de forma explícita más adelante, dichas posiciones marcarán el mínimo y el máximo valor que puede tomar j.

El estudio de los valores que cumplan la segunda restricción es más complicado. Para ello considérese C una cadena de esta ruta. Definimos el siguiente conjunto:

$M =$ Conjunto de clientes cuyos puntos de carga y descarga no pertenecen ninguno a C;

para todo cliente r, cumpliendo que o bien r^+ o bien r^-, pero solo uno, pertenece a C se definen:

$A(r) =$ Subconjunto de M de clientes s tales que s^- es anterior a r^+ en la ruta actual;

$$s^+ - \cdots - s^- - \cdots - r^+ - \cdots - r^-$$

$B(r) =$ Subconjunto de M de clientes s tales que s^+ es anterior a r^+ y s^- es posterior a r^- en la ruta actual;

$$s^+ - \cdots - r^+ - \cdots - r^- - \cdots - s^-$$

$C(r) =$ Subconjunto de M de clientes s tales que s^+ es posterior a r^+ y s^- es anterior a r^- en la ruta actual;

$$r^+ - \cdots - s^+ - \cdots - s^- - \cdots - r^-$$

$D(r) =$ Subconjunto de M de clientes s tales que s^+ es posterior a r^- en la ruta actual

$$r^+ - \cdots - r^- - \cdots - s^+ - \cdots - s^-.$$

$\forall t \in N^+ \cup N^-$ se define orden(t) como el número de orden o posición que ocupa el elemento t en la ruta actual. Se tiene el siguiente

Lema 1 *Sea j la variable que indica la posición donde se recoloca C, según se ha definido anteriormente:*

a) $\forall s \in A(r)$, *C no puede ser recolocado entre s^+ y s^-, es decir, j no puede tomar valores comprendidos entre orden(s^+) y orden(s^-)* -1, *ambos incluidos;*

b) Sean $m1 = \max\{orden(s^+)/s \in B(r)\}$ *y* $m2 = \min\{orden(s^-)/s \in B(r)\}$, *$j$ no puede tomar valores comprendidos entre 1 y* $m1-1$ *ambos incluidos, ni entre* $m2$ *y* $2 \cdot n - 1$ *ambos inclusive.*

c) $\forall s \in C(r)$, *j no puede tomar valores comprendidos entre orden(s^+) y orden(s^-)* $-$ 1, *ambos incluidos;*

d) $\forall s \in D(r)$, *j no puede tomar valores comprendidos entre orden(s^+) y orden(s^-)* $-$ 1, *ambos incluidos.*

Demostración: Si j tomara alguno de los valores anteriormente señalados, se tendría, para algún cliente s, una de las dos posiciones relativas siguientes entre los puntos de carga y descarga de r y s en la nueva ruta obtenida:

$$s^+ - \cdots - r^+ - \cdots - s^- - \cdots - r^-$$

o bien

$$r^+ - \cdots - s^+ - \cdots - r^- - \cdots - s^-.$$

■

4. SITUACIONES POSIBLES DEL CASO GENERAL

Se define $r(t)$ el elemento de la ruta actual que ocupa la posición t; por consiguiente, según lo comentado anteriormente, que $r(1) = r(2n) = 1$. Sea la cadena C determinada por los valores de i y k:

$$r(i) - r(i+1) - \cdots - r(i+k-1).$$

Definimos los siguientes conjuntos de clientes:

$$A^+ = \{r/r^+ \in C,\ r^- \notin C\} \quad \text{y} \quad A^- = \{r/r^- \in C,\ r^+ \notin C\};$$

se definen también:

$$\textit{límite\_inferior} = \max\{\text{orden}(j^+)/\, j \in A^-\}, \quad \text{si } A^- = \emptyset \quad \Rightarrow \quad \textit{límite\_inferior} = 1;$$

$$\textit{límite\_superior} = \min\{\text{orden}(j^-)/\, j \in A^+\}, \quad \text{si } A^+ = \emptyset \quad \Rightarrow \quad \textit{límite\_superior} = 2n.$$

Sean además $j1$ el elemento de A^+ verificando orden($j1^-$) = *límite_superior* si $A^+ \neq \emptyset$; y $j2$ el elemento de A^- verificando orden($j2^+$) = *límite_inferior* si $A^- \neq \emptyset$; se tiene el siguiente

Lema 2 *Sea j la variable que indica la posición donde se recoloca C, según se ha definido anteriormente, se tiene que j sólo puede tomar los valores comprendidos entre límite_inferior y límite_superior-1, ambos inclusive.*

Demostración: Trivial, ya que si j tomara otros valores en la ruta resultante se violaría, para al menos un cliente, la restricción 1 del apartado 3, es decir, algún punto de descarga sería anterior a su correspondiente de carga.

Por otra parte, según los elementos de A^- y A^+ se tienen los siguientes 4 casos:

1. $A^+ = \emptyset, \quad A^- = \emptyset$:

La cadena está compuesta por pares completos de puntos de carga descarga, como por ejemplo, $r^+r^-s^+s^-$ o bien $r^+s^+s^-r^- \cdots$; en este caso ninguna recolocación puede romper ninguna precedencia obligatoria señalada en la sección anterior.

2. $A^+ \neq \emptyset, \quad A^- = \emptyset$:

$$1 - \cdots - r(i) - r(i+1) - \cdots - r(i+k-1) - \cdots - j1^- - \cdots - 1$$

Los clientes que se van a considerar, según las posiciones relativas de sus correspondientes puntos de carga y descarga, para estudiar las posibles recolocaciones son las siguientes:

a) $1-\cdots-s^+-\cdots-s^--\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-j1^--\cdots-1$

Conjunto de clientes s verificando que orden$(s^-) < i$; que va a coincidir con $A(j1)$, (ver apartado anterior);

b) $1-\cdots-s^+-\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-j1^--\cdots-s^--\cdots-1$

Conjunto de clientes s, verificando que orden$(s^+) <$ orden$(j1^-)$; es decir $B(j1)$;

c) $1-\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-s^+-\cdots-s^--\cdots-j1^--\cdots-1$

Conjunto de clientes s verificando que orden$(s^+) >$ orden$(j1^-)$, es decir $C(j1)$.

A efectos del estudio de recolocaciones factibles no se considera el conjunto de clientes s verificando que orden$(s^+) >$ orden$(j1^-)$, (es decir $D(j1)$), pues la cadena C no puede ser insertada en una posición posterior a la de $j1^-$ (Lema 2).

3. $A^- \neq \emptyset, \quad A^+ = \emptyset$:

$$1-\cdots-j2^+-\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-1$$

Los conjuntos de clientes a considerar son los siguientes:

a) $1-\cdots-s^+-\cdots-j2^+-\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-s^--\cdots-1$

Conjunto de clientes s verificando que orden$(s^+) <$ orden$(j2^+)$ y orden$(s^-) > i+k-1$, es decir $B(j2)$;

b) $1-\cdots-j2^+-\cdots-s^+-\cdots-s^--\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-1$

Conjunto de clientes s verificando que orden$(s^+) >$ orden$(j2^+)$ y orden$(s^-) < i$, es decir $C(j2)$;

c) $1-\cdots-j2^+-\cdots-r(i)-r(i+1)-\cdots-r(i+k-1)-\cdots-s^+-\cdots-s^--\cdots-1$

Conjunto de clientes s verificando que orden$(s^+) > i+k-1$, es decir $D(j2)$.

Al igual que en el caso anterior no se considera el conjunto $A(j2)$.

4. $A^+, A^- \neq \emptyset$:

$$1 - \cdots - j2^+ - \cdots - r(i) - r(i+1) - \cdots - r(i+k-1) - \cdots - j1^- - \cdots - 1$$

Sólo se consideran los dos siguientes conjuntos:

a) $1 - \cdots - j2^+ - \cdots - s^+ - \cdots - s^- - \cdots - r(i) - r(i+1) - \cdots - r(i+k-1) - \cdots - j1^- - \cdots - 1$
Conjunto de clientes s verificando que orden$(s^+) >$ orden$(j2^+)$ y orden$(s^-) <$ i, es decir $C(j2)$;

b) $1 - \cdots - j2^+ - \cdots - r(i) - r(i+1) - \cdots - r(i+k-1) - \cdots - s^+ - \cdots - s^- - \cdots - j1^- - \cdots - 1$
Conjunto de clientes s verificando que orden$(s^+) > i+k-1$ y orden$(s^-) <$ orden$(j1^+)$, es decir $C(j2)$;

Obviamente, no se van a considerar otros conjuntos de clientes.

■

5. DESCRIPCIÓN DEL ALGORITMO EN SEUDOCÓDIGO

A continuación se describe el algoritmo que sigue la idea básica descrita en el apartado 2, y con la determinación de intercambios factibles según se estudia en los apartados 4 y 5. Se tienen los siguientes parámetros de entrada:

n : valor que determina los conjuntos N^+ y N^- según la definición del apartado;

t : matriz de distancias;

y los siguientes parámetros por variable:

ruta: vector que registra la solución en cada momento, es decir, ruta[i] indica el punto que ocupa la posición i en la ruta actual;

costetotal: distancia total de la solución;

además se hacen uso de las siguientes variables:

gmax: Indica el valor máximo de *ganancia* (definida en el apartado 2) en cada iteración;

imax: Indica el lugar donde comienza la cadena correspondiente a *gmax*;

nele: indica el número elementos de dicha cadena;

jmax: indica el lugar donde se recoloca la cadena correspondiente a *gmax*;

orden: vector que indica la posición de cada punto en la ruta actual;

valido: vector lógico que indica, para cada cadena, en qué posiciones puede ser recolocada de forma factible.

Asímismo, siguiendo las definiciones tomadas hasta ahora, i es la variable auxiliar que indica la posición donde comienza la cadena C, k el número de elementos que la componen, y j la posición donde va a ser recolocada; el algoritmo actúa de la forma siguiente:

Hacer ruta como la ruta inicial y coste total como la distancia de ésta.

Repetir

hacer *gmax*:=0;

Para k:=1 hasta $n-1$ hacer:

Para i:=2 hasta $2 \cdot n - k - 1$ hacer

inicio

Para j:=1 hasta $2n-1$ hacer valido[j]:=TRUE;

Determinar los conjuntos A^+ y A^- (según la definición del apartado 4).

Si $A^- \neq \emptyset$ entonces

inicio

Hacer *límite_inferior* = $\max\{\text{orden}[j^+] / j \in A^-\}$;

Determinar $j2 \in A^- / orden[j2^+]$ = *límite_inferior*

fin

si no: Hacer *límite_inferior* = 1;

Si $A^+ \neq \emptyset$ entonces

inicio

Hacer *límite_superior* = $\min\{orden[j^-] / j \in A^+\}$;

Determinar $j1 \in A^+ / orden[j1^-]$ = *límite_superior*

fin

si no: Hacer *límite_superior* = $2n$;

Para j:=1 hasta *límite_inferior*-1 hacer *valido*$[j]$:=FALSE;

Para j:=*límite_superior* hasta $2n-1$ hacer *valido*$[j]$:=FALSE;

Si $A^+ \neq \emptyset, \quad A^- = \emptyset$ entonces:

inicio

Determinar conjuntos $A(j1), B(j1), C(j1)$; (según la definición del apartado 3)

$\forall s \in A(j1) \cup C(j1)$ hacer:

Para j:=*orden*$[s^+]$ hasta *orden*$[s^-]-1$ hacer *valido*$[j]$:=FALSE;

Si $B(j1) \neq \emptyset$ hacer

inicio

Determinar $m1 = \max\{orden[s^+] / s \in B(j1)\}$;

Determinar $m2 = \min\{orden[s^-] / s \in B(j1)\}$;

Para j:=1 hasta $m1-2$ y de $m2$ hasta $2 \cdot n-1$ hacer *valido*$[j]$:=FALSE

fin;

fin;

Si $A^+ = \emptyset, \quad A^- \neq \emptyset$ entonces:

inicio

Determinar conjuntos $B(j2), C(j2), D(j2)$; hacer:

$\forall s \in C(j2) \cup D(j2)$ hacer:

Para j:=*orden*$[s^+]$ hasta *orden*$[s^-]-1$ hacer *valido*$[j]$:=FALSE;

Si $B(j2) \neq \emptyset$ hacer

inicio
Determinar $m1 = \max\{orden[s^+]/s \in B(j2)\}$;
Determinar $m2 = \min\{orden[s^-]/s \in B(j2)\}$;
Para j:=1 hasta $m1-2$ y de $m2$ hasta $2 \cdot n-1$ hacer $valido[j]$:=FALSE
fin;

fin;

Si $A^+, A^- \neq \emptyset$ entonces:

inicio
Determinar conjuntos $C(j1), C(j2)$;
$\forall x \in C(j1) \cup C(j2)$ hacer:

Para j:=$orden[s^+]$ hasta $orden[s^-]-1$ hacer $valido[j]$:=FALSE;

fin;

Para j:=1 hasta $i-2$, y j:=$i+k$ hasta $2 \cdot n-1$ hacer:

Si $valido[j]$ entonces:

inicio
Calcular el valor de *ganancia* (según se definió el apartado 2);
Si *ganancia* $>$ *gmax* entonces:

inicio
gmax:=*ganacia*;
imax:=*i*;

kmax:=*k*;

jmax:=*j*;

fin

fin;

fin;

Si $gmax > 0$ entonces:

inicio

Sustituir los arcos $(imax-1, imax)$, $(imax+kmax-1, imax+kmax)$, $(jmax, jmax+1)$ por los arcos $(imax-1, imax+kmax)$, $(jmax, imax)$ y $(imax+kmax-1, jmax+1)$ y registrar el resultado en *ruta*;

Hacer *costetotal*:=*costetotal* – *gmax*

fin

hasta que $gmax = 0$.

Existen copias del algoritmo programado en PASCAL a disposición de las personas interesadas.

6. RESULTADOS COMPUTACIONALES

Para examinar la eficacia del algoritmo anteriormente descrito se han realizado cuatro tipos de pruebas: el primero tiene como objeto reflejar la eficacia del método desarrollado para determinar el ahorro en el tiempo de computación de los intercambios factibles, el segundo tiene como objeto comprobar la desviación del óptimo de la solución obtenida; el tercero determinar en qué medida la incorporación de este heurístico en los algoritmos exactos descritos por Pacheco, (1994), y Pacheco y otros, (1994), para este problema hace reducir el tiempo de computación de éstos; y el cuarto comprobar la evolución de los tiempos de computación de este algoritmo en función del tamaño del problema.

Tanto la implementación de los algoritmos que se han utilizado, como la programación de los diferentes problemas que han servido de prueba, se han hecho utilizando el compilador Borland Pascal (ver. 7.0). El equipo técnico usado es un ordenador

personal tipo PC AT i 486dx2 a 50 Mhz para las tres últimas pruebas, y un ordenador Pentium a 100 Mhz. para la primera. (Existen copias de estos programas a disposición de las personas interesadas.)

6.1. Chequeo de intercambios factibles. Reducción de tiempo

Como se ha comentado en el apartado 2, el chequeo de la factibilidad de cada intercambio uno a uno puede emplear mucho tiempo de computación. El objeto de los apartados 3 y 4 es desarrollar un método para determinar de forma más *rápida* qué intercambios son factibles. Para establecer la eficacia del método propuesto, se han simulado 20 matrices de distancia, para cada tamaño de problema: 5, 10, 15, 20, 25 y 30 clientes. Los problemas así generados se han resuelto utilizando el algoritmo con este método (tal como se describe en el apartado 5) y chequeando los intercambios uno a uno. La forma de generar distancias es asignar a cada punto del problema dos coordenadas *x* e *y*, cuyos valores son generados aleatoriamente con distribución uniforme entre 0 y 100. La distancia entre cada par de puntos se define como la distancia euclídea correspondiente.

A continuación se muestra un cuadro que resume los resultados obtenidos: tiempos de computación para ambas formas y reducción porcentual:

N. clientes	**5**	**10**	**15**	**20**	**25**	**30**
Uno_a_Uno	$0'023$	$0'247$	$2'199$	$9'617$	$34'594$	$85'415$
Método	$< 0'001$	$0'021$	$0'083$	$0'199$	$0'464$	$0'823$
Reducción	$95'65$	$91'49$	$96'22$	$97'93$	$98'37$	$99'03$

6.2. Desviación del óptimo

Para estudiar la proximidad al óptimo de las soluciones obtenidas por este heurístico se han simulado matrices de distancias de dos formas diferentes. En la primera de ellas las distancias son generadas aleatoriamente tomando valores enteros con la misma probabilidad entre 0 y 99. La segunda forma de generar distancias coincide con la utilizada en el apartado 6.1. En ambos casos se han generado 20 matrices para cada tamaño del problema: 5, 6, 7, 8 y 9 clientes, (correspondientes a 11, 13, 15, 17 y 19 puntos).

A continuación, para cada tipo de matriz, se muestra un cuadro que resume los resultados obtenidos: En estos cuadros se muestra, para cada número de clientes, la desviación porcentual del óptimo, (media, máxima y mínima), así como el número de casos en los que se alcanzó.

6.2.1. Primer tipo de matriz

Para obtener la ruta inicial, (Paso 1 descrito en el apartado 5) se han generado aleatoriamente 50 rutas factibles y se ha tomado aquella con menos distancia total. Los resultados son los siguientes:

Número de clientes	Desviación porcentual del óptimo			Casos en los que se alcanza
	Mínima	Media	Máxima	
5	0	**0**	0	20
6	0	**0′011**	0′104	16
7	0	**0′015**	0′240	16
8	0	**0′028**	0′350	18
9	0	**0′014**	0′255	17

6.2.2. Segundo tipo de matriz

En este caso, para la ruta inicial se ha elegido aquella con menos distancias entre 10 rutas factibles generadas aleatoriamente y las obtenidas utilizando la adaptación a este problema de los algoritmos de inserción más cercana y más lejana para el TSP, (en el trabajo de Golden y otros, (1980), se hace una amplia recopilación y descripción de los principales algoritmos de inserción para el TSP, así como de otros algoritmos constructivos que pueden ser adaptados fácilmente a este problema). Los resultados son los siguientes:

Número de clientes	Desviación porcentual del óptimo			Casos en los que se alcanza
	Mínima	Media	Máxima	
5	0	**0**	0	20
6	0	**0**	0	20
7	0	**0**	0	20
8	0	**0′015**	0′302	19
9	0	**0**	0	20

6.3. Reducción en el tiempo de computación de los exactos

Como se ha mencionado anteriormente, en los trabajos de Pacheco, (1994), y Pacheco y otros, (1994), se desarrolla una estrategia para el diseño de algoritmos

exactos para este problema, a partir de métodos Branch & Bound para el TSP. En los algoritmos exactos para problemas de rutas se puede conseguir, en muchos casos, reducir el tiempo de computación mediante determinadas estrategias. Entre éstas destacamos: generar cotas superiores en cada vértice del árbol de búsqueda si el algoritmo es de tipo Branch & Bound, (ver Volgenant y Jonker, (1982), o Nurmi, (1991), más recientemente); obtener cotas inferiores más ajustadas al óptimo mediante métodos de penalización lagrangiana (ver Held & Karp, (1970) y (1971)), o métodos de Programación Dinámica, (ver Christodides y otros, (1981), etc. En nuestro caso, vamos a reducir el tiempo de computación realizando dos modificaciones en los algoritmos exactos anteriormente señalados; éstas son las siguientes:

1. Partir como solución inicial la obtenida por el heurístico descrito en este trabajo.

2. Cada vez que se llegue a una solución, utilizar dicho heurístico para mejorarla.

El objeto de estas modificaciones es obtener rápidamente cotas superiores ajustadas al óptimo para evitar exploraciones innecesarias, y reducir el tiempo de computación. Para verificar estos ahorros de tiempo se han generado 20 matrices de distancias, como en el caso del apartado 6.1., para cada tamaño del problema: 5, 6, 7, 8 y 9 clientes. Para obtener una solución inicial se empleó como ruta inicial aquella con menos distancia total entre 50 rutas factibles generadas aleatoriamente. Los resultados son los siguientes:

Número de clientes	**Tiempo de Computación en segundos**		**Reducción Porcentual**
	Algoritmo Original	**Algoritmo Modificado**	
5	0′288	0′215	25′347
6	1′396	0′630	54′871
7	17′205	11′362	33′961
8	59′644	44′558	25′293
9	474′427	238′822	49′661

6.4. Evolución de los tiempos de computación

Finalmente se ha realizado un tercer tipo de pruebas destinado a establecer la evolución de los tiempos de computación en ordenadores personales, del heurístico desarrollado en este trabajo, para problemas de mayor tamaño. Para ello se han

simulado 20 matrices de distancia como en el apartado 6.1.2., (distancias euclídeas), para cada tamaño del problema: 10 clientes, 20,..., hasta 120 clientes (241 puntos). Como ruta inicial se ha elegido aquella con menos distancias entre 10 rutas factibles generadas aleatoriamente y las obtenidas por los algoritmos de inserción (ver apartado 6.1.2). Los resultados son los siguientes:

Número de clientes	**Tiempo medio de Computación en segundos**		
	Solución Inicial	**Heurístico Propuesto**	**Total**
10	0′120	0′035	0′155
20	1′282	0′321	1′603
30	5′029	1′516	6′545
40	14′891	3′212	18′103
50	35′403	8′295	43′698
60	71′609	18′775	90′384
70	125′081	24′044	149′125
80	210′159	39′886	250′045
90	339′721	57′293	397′014
100	520′928	101′775	622′703
110	761′427	130′193	891′620
120	1074′992	174′439	1249′431

7. CONCLUSIONES

A la vista de estos resultados se llegan a las siguientes conclusiones:

- El heurístico descrito ofrece soluciones muy poco desviadas del óptimo: en caso de las matrices de distancias del apartado 6.1.1. un máximo del 0'35%; en caso de matrices de distancias euclídeas la solución coincide con la óptima en 99 de los 100 casos; el único caso en el que no coincide se desvía el 0'302%. Por consiguiente, especialmente para este último tipo de matrices, la aproximación es realmente grande.
- El ofrecer soluciones cercanas al óptimo, (cuando no coinciden con él), permite reducir el tiempo de computación de los algoritmos exactos para este problema cuando se combinan con este heurístico; —(tal como se hace en el apartado

6.3)—. Esto se debe a que se parte de una cota superior en el vértice inicial muy ajustada al óptimo, con lo que se rechazan más exploraciones innecesarias.

- El tiempo de computación que emplea es insignificante si se le compara con el que emplean los exactos existentes. Esto hace que se puedan resolver problemas de mucho mayor tamaño. En el apartado 6.4 se resuelven problemas de hasta 241 puntos, pero la evolución de los tiempos de computación permite adivinar que se podrían resolver problemas de mayor tamaño, en un tiempo aceptable, con ordenadores o compiladores que soportaran matrices de distancia de mayor dimensión.

- El empleo de un tiempo de computación pequeño, se debe fundamentalmente al método utilizado para establecer los intercambios factibles en cada iteración, basado en los resultados teóricos desarrollados obtenidos en los apartados 3 y 4. Este método permite una reducción porcentual del tiempo de computación de más del 90% y que aumenta con el tamaño de los problemas.

Concluir finalmente que el algoritmo heurístico propuesto resulta extremadamente eficaz al ofrecer soluciones muy cercanas al óptimo, —sobre todo con distancias euclídeas—, en un tiempo de computación mucho menor que los exactos, lo que permite poder resolver problemas de mayor tamaño.

8. REFERENCIAS Y BIBLIOGRAFÍA

[1] **Christofides, N., Mingozzi, A.** & **Toht, P.** (1981). «State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems». *Networks*, **11, 2**, 145–164.

[2] **Casco, D.O., Golden, B.L.** & **Wasil, E.A.** (1988). «Vehicle Routing with Bachauls: Models, Algorithms, and Case Studies». In*Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., Nort-Holland, 7–46.

[3] **Deif, I.** & **Boding, L.** (1984). «Extension of the Clark and Wrigth Algorithm for Solving the Vehicle Routing Problem with Backhauling». *Proceedings of the Babson Conference on Software Uses in Transportation and Logistics Management* (A.E.KIDDER, edit.), 75–96. Babson Park, Massachussets.

[4] **Desrochers, M.** & **Soumis, F.** (1991). «Implantation et Complexité des Techniques de Programmation Dynamique dans les Méthodes de Confection des Tournées et d'Horaires». *Recherche Operationnelle/Operations Researsch*, **25, 3**, 291–310.

[5] **Desrosiers, J., Dumas, Y.** & **Soumis, F.** (1987). «Vehicle Routing Problem with Pickup, Delevery and Time Windows». *Cahiers du GERARD*, Ecole Des Hautes Estudes Commerciales de Montreal.

[6] **Dumas, Y.** (1985). «Confection d'iteneraires de vehicules en vue du transport de plusiers origines à plusiers destinations». *Centre de Recherche sur le Transports*. Université de Montreal.

[7] **Goetschalcks, M.** & **Horsley, C.** (1986). «The Vehicle Routing Problem with Backhauls». *Material Handling Researcsh Center*, Dept. of Industrial and Systems Engineering, Georgia Institute of Technology.

[8] **Golden, B., Baker,E. , Alfaro, J.** & **Schaffer, J.** (1985). «The Vehicle Routing Problem with Backhauling: Two Approaches». *Twenty-First Annul Meeting of S.E. TIMS* (R.D.HAMMESFAHR, edit.), Myrtle Beach, SC, 90–92.

[9] **Golden, B., Bodin, L. Doyle,T.** & **Stewart,W.Jr.** (1980). «Approximate Traveling Salesman Algorithms». *Operations Researsch*, **28, 3**, parte II, 694–711.

[10] **Held, M.** & **Karp, R.M.** (1970). «The Traveling Salesman Problem and Minimum Spanning Trees». *Ops. Res.*, **18**, 1138–1162.

[11] **Held, M.** & **Karp, R.M.** (1971). «The Traveling Salesman Problem and Minimum Spanning Trees: Part II». *Mathematical Programing I*, 6–25.

[12] **Kalantari, B., Hill, A.V.** & **Arora, S.R.** (1985). «An Algorithm for the Traveling Salesman Problem with Pickup and Delevery Customers». *European Journal of Operational Research*, **22**, 377–386.

[13] **Little, J., Murty, K., Sweeney, D.** & **Karel, C.** (1963). «An Algorithm for the Traveling Salesman Problem». *Operations Research*, **11 (5)**, 972–989.

[14] **Lin, S.** (1965). «Computer Solutions to the Traveling Salesman Problem». *Bell Syst. Tech. Jou.*, **44**, 2245–2269.

[15] **Nurmi, K.** (1991). «Traveling Salesman Problem Tools for Microcomputers». *Computers & Ops. Res.* **18, 8**, 741–749.

[16] **O'Brien, S.K.** & **Nameroff, S.** (1993). *Turbo-Pascal 7: Manual de Referencia*. Osborne McGraw-Hill. Madrid.

[17] **Or, I.** (1976). *Traveling Salesman Type Combinatorial Problems and their Relations to the Logistics of Blood Banking*. Ph. Thesis, Dpt. of Industrial Enginnering and Management Sciences, Northwestern Univ.

[18] **Pacheco, J.** (1994). *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 1994.

[19] **Pacheco, J., García, A.** & **Aragón, A.** (1994). *Problemas de Ruta con Carga y Descarga en Sistemas LIFO*. XXI Congreso de Estadística e I. Optva. celebrado en Calella en Abril de 1994.

[20] **Psaraftis, H.** (1983). «An Exact Algorithm for the Single Vehicle Many-to-Many Dial-a-Ride Problem with Time Windows». *Transportation Sci.*, **17**, 351–360.

[21] **Solomon, M., Baker, E.** & **Schaffer, J.** (1988a). «Vehicle Routing and Scheduling Problems with Time Window Constraints». In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN, B.L. and ASSAD, A.A., Nort-Holland, 85–106.

[22] **Volgenant, T.** & **Jonker, R.** (1982). «A Branch and Bound Algorithm for the Symmetric Traveling Salesman Problem based on the 1-tree Relaxation». *Eur. J. Ops. Res.*, **9**, 83–89.

ENGLISH SUMMARY

HEURISTIC FOR THE PICK-UP AND DELIVERY ROUTING PROBLEMS IN LIFO UNLOADING SYSTEMS

JOAQUÍN A. PACHECO*
E.U.E. Empresariales de Burgos

In this paper a heuristic is proposed to solve the «Pick-up and Delivery Problem (PDP) using only one vehicle with limitless capacity in LIFO unloading systems», i.e., only the last goods to be loaded onto the vehicle may be unloaded at one time. This algorithm is an extension an adaptation of the Or method for the Travelling Salesman Problem (TSP) which can also be used for aymetrical instances. With this heuristic one can solve large scale problems in reasonable calculation times on PC with only a small deviation from the optimum.

Keywords: Pick-up & Delivery with LIFO systems, Or-exchanges, feasibility, checking.

*Joaquín A. Pacheco. Dpto. Matemáticas. E.U.E. Empresariales de Burgos. Fco. Vitoria s/n. 09006 Burgos.

– Received march 1995.
– Accepted july 1996.

The Pick-up and Delivery Problem (PDP) may be described in the following way: One has a set of clients $N = \{2,3,\ldots,n\}$; each client i requires that goods be transported from an origin i^+ to a destination i^-. With this objective in mind one has m vehicles wich leave an initial citi 1, which they then return to when their route is completed. The object is to design feasible vehicle routes with the shortest total distances. The set of pick-up points will be denoted by $N^+ = \{r^+/r = 2,3,\ldots,n\}$; the set of delivery by $N^- = \{r^-/r = 2,3,\ldots,n\}$; and the loads of each client r by $q(r)$, $r = 2,3,\ldots,n$. The special case in which the orders required are all the same is know as Dial-A-Ride-Problem (DARP) which occurs in passenger transport, school transport, etc...

Now let us consider the PDP with one vehicle with the fallowing restriction: At any one moment only the last goods loaded may be unloaded. This problem, wich will be denoted PDP with one vehicle with a LIFO (last-in-first-out) un loading system, appears in many situations in the world of Transport and Industry: transport of certain industrial gases, distribution of pallets in the auto industry, or in general, in those delivery activities in which a controlled time is required for unloading.

In this paper a heuristic is proposed capable of solving large scale problems in moderable calculation times, deviating from the optimum in only a few cases and by a small amount. This technique is an extension and adaptation to this problem of the Or algorithm for the TSP, (1976).

The paper is organized in the falowing way: section 2 explains the basic idea of the algorithm to be developed; section 3 studies the conditions necessary for a 3-interchange, in this case repositioning, to be feasible; section 4 studies the possible situations depending on the elements that make up the chain which is to the repositioned; section 5 describes the algoritm with the results from the previous sections; section 6 cheks the efficiency of this algorithm starting from the solving of a series of the simulated problems; section 7 explains the conclusions drawn from the results obtained in the previous section. Furthermore, an appendix is added which describes the algorithm in more detail in PASCAL.

As in previous papers, —Pacheco, (1994), and Pacheco *et al.*, (1994),— it is assumed that the total load from all the clients; the general case will be studied later in other works.