Universidad Complutense de Madrid

DPTO. Estadística e Investigación Operativa

Facultad de Ciencias Matemáticas

PROBLEMAS DE RUTAS CON VENTANAS DE TIEMPO

Autor: *Joaquín Antonio PACHECO BONROSTRO*

Directores: *Angel FELIPE ORTEGA*

Alfredo GARCIA GUEMES

Agradecimientos y Dedicatorias

Muchas personas me han ayudado en la realización de esta memoria, ya sea en el aspecto anímico como en el aspecto intelectual y científico. A todos ellos les dedico la realización de este trabajo.

Entre los que me ayudaron anímicamente destacar a mis amigos, mis padres y mi novia Cristina.

Entre los segundos quiero dar las gracias especialmente por su colaboración y paciencia a mis dos directores: Alfredo García Güemes, Catedrático de la E.U.E.Empresariales de Burgos y Angel Felipe Ortega, Profesor Titular de la Facultad de Matemáticas de la Universidad Complutense de Madrid, sin cuyos consejos y directrices hubiera sido imposible la realización de este trabajo.

En cualquier caso, cualquier error que pudiera aparecer es de mi exclusiva responsabilidad.

INTRODUCCION

El campo de los Problemas de Rutas de Vehículos ha sido catalogado por ASSAD, (1.988), como *"uno de los grandes éxitos de la Investigación Operativa en la última década"*. Según los observadores, este éxito debe atribuirse por una parte al diseño de modelos muy ajustados a la realidad para los que se han desarrollado algoritmos eficientes; por otra parte al desarrollo del *hardware*. En otras palabras, la llave del éxito ha consistido en unir la eficacia de los algoritmos por un lado y la potencia de los ordenadores por otro.

Ejemplos de aplicaciones prácticas pueden encontrarse en los trabajos de AKINC, (1.992), BALL, (1.988), HOLT y WATTS, (1.988), LEVI y BODIN, (1.988), LARSON y otros, (1.988), PAPE, (1.988) y GOLDEN, (1.988). Descripciones de cómo se integran estas técnicas algorítmicas en sistemas logísticos más complejos se pueden encontrar en FISHER y otros (1.982), LYSGAARD, (1.992), ROY y CRAINIC, (1.992), ROUSSEAU, (1.988), VAN VLIET y otros, (1.992) y en TANCHOCO y SINRIECH, (1.992).

En España destacan los trabajos de BENAVENT y otros, (1990) y (1.992), para los Problemas de Rutas con Restricciones de Capacidad, así como los de COBES y COROMINAS, (1.990), para problemas de Transporte Escolar.

Recientemente se está intentando abordar los problemas de rutas combinando las técnicas de Programación Matemática con las de Inteligencia Artificial; así se puede ver en BAGCHI y NAG, (1.991), o en POTVIN y otros, (1.990).

En este trabajo se aportan nuevas contribuciones en el desarrollo de técnicas de solución para modelos de Rutas que se adaptan a problemas reales. Más concretamente se desarrolla una serie de algoritmos heurísticos para el Problema de Rutas de Vehículos con Ventanas de Tiempo

(VRPTW) que pueden ser programados para funcionar en ordenadores personales, de los cuales se hace hincapié en dos, que se van a denominar VARIANTE RAMA IZQUIERDA, y VARIANTE TIEMPO MAXIMO DE COMPUTACION. Con estos dos algoritmos se obtienen mejores soluciones, para determinados conjuntos de clientes, que las obtenidas por otros algoritmos heurísticos muy utilizados actualmente.

Los pasos que se han dado para el desarrollo de estos algoritmos son los siguientes: desarrollo de un algoritmo exacto para el Problema del Viajante con Ventanas de Tiempo (TSPTW), adaptación de este algoritmo para obtener un algoritmo exacto para el VRPTW e incorporación a éste de determinados criterios de parada que dan lugar a la familia de heurísticos antes mencionada.

Con este procedimiento se recupera la técnica de los algoritmos exactos truncados o de *búsqueda incompleta*, desarrollada por LITTLE y otros (1.963) y más recientemente por BALAS y CHRISTOFIDES, (1.981). Por otra parte, además del objetivo inicial de desarrollar heurísticos efectivos para el VRPTW, se consiguen estos otros:

- Desarrollar algoritmos exactos para el TSPTW y el VRPTW. Estos pueden ser utilizados para problemas con pocos nodos, o en los que se disponga de tiempo suficiente para obtener la solución.

- El algoritmo exacto para el TSPTW puede ser incorporado a otros heurísticos para el VRPTW mejorando la eficacia de éstos. Por ejemplo en la segunda fase de las técnicas del tipo *Grupos 1°-Rutas 2°*, como se verá en el capítulo 2.

- El algoritmo obtenido para el TSPTW puede ser adaptado fácilmente a otro tipo de problemas más complejos como el TSPTW con Carga y Descarga, el Problema de Carga y Descarga con Ventanas de Tiempo (PDPTW) con un vehículo, el PDPTW con sistemas LIFO de entrada y salida de mercancía..., como se expone en el capítulo 6.

Este trabajo se estructura de la siguiente manera: en los capítulos 1 y 2 se recopilan los principales algoritmos existentes para la resolución del TSP y del VRP respectivamente. Además en el capítulo primero se proponen modificaciones en el algoritmo de HELD & KARP que reducen el tiempo de computación, y en el capítulo segundo se describe una nueva variante en el algoritmo de BEASLEY que aumenta su eficacia.

En el tercer capítulo se adaptan los anteriores algoritmos para problemas de rutas con ventanas de tiempo; la justificación de estos problemas es clara: por una parte los aspectos temporales (horarios, tiempos de conducción,...), suponen una característica fundamental que ha de ser incorporada a los modelos de rutas si se pretende resolver aplicaciones reales (ALVAREZ, (1.992)). Por otra parte, su adaptación ha supuesto en muchos casos, el desarrollo de nuevas técnicas, totalmente diferentes a las existentes hasta el momento.

En el cuarto capítulo se desarrollan los algoritmos exactos y los heurísticos que de ellos se derivan, según lo comentado anteriormente. La descripción de los algoritmos en el capítulo 4 está hecha de forma modular, describiendo por separado cada uno de los procedimientos, y describiendo cada procedimiento, a su vez, mediante pasos claramente diferenciados. El propósito es conseguir que sean fácilmente programables para ser implementados en lenguajes de alto nivel, aunque se recomienda sobretodo aquellos que admitan recursividad, como PASCAL, C, C++,... En concreto, la implementación de los algoritmos utilizados se ha realizado con los compiladores TURBOPASCAL 6.0 y BORLAND PASCAL (TP 7.0 Profesional), de Borland. El ordenador utilizado ha sido un PC-AT i486 dx2 a 50 Mhz.

En el quinto capítulo se examinan dos de estos algoritmos, VARIANTE RAMA IZQUIERDA y VARIANTE TIEMPO MAXIMO DE COMPUTACION y se comparan los resultados con los obtenidos por otros heurísticos. Para ello se simulará una serie de problemas con diferente número de nodos, y se planteará un problema real.

Finalmente, en el sexto capítulo se muestran diferentes adaptaciones de los algoritmos desarrollados, además de nuevas direcciones de investigación siempre dentro del campo de los problemas de rutas.

CAPITULO 1. ALGORITMOS PARA EL PROBLEMA DEL VIAJANTE

1.- INTRODUCCION

Quizás el problema de Optimización Combinatoria más estudiado en la literatura desde hace más de 40 años es el conocido Problema del Viajante o TSP (Traveling Salesman Problem). Su formulación es la siguiente: Un individuo tiene que visitar n-1 ciudades partiendo de una ciudad inicial (ciudad 1) y volver a ella, de forma que la ruta elegida sea la más corta o más barata.

En este problema y en su generalización al Problema de Rutas de Vehículos o VRP (Vehicle Routing Problem), es donde más "éxito" han tenido las aplicaciones de técnicas algorítmicas al desarrollo de sistemas eficientes en el mundo de la industria, comercio, defensa... Su importancia está, por tanto, fuera de toda duda.

El VRP tiene un planteamiento basado en el TSP y, por ello, ha adoptado las técnicas de solución de éste, es más: la gran mayoría de las técnicas de solución de los problemas de rutas son básicamente algoritmos del TSP con la incorporación de algunos pasos previos o posteriores. Resulta por tanto imprescindible para un correcto estudio de los problemas de rutas, un conocimiento profundo de los algoritmos del TSP.

En el TSP sin restricciones adicionales hay (n-1)! soluciones factibles, también llamadas rutas, y es un problema NP-completo. De hecho todos los algoritmos de solución exactos para este problema requieren un tiempo de computación exponencial en el número de ciudades.

Los métodos de solución suelen clasificarse en dos grandes grupos:

1.- **Algoritmos Exactos**, la mayoría de tipo Branch & Bound, (Syslo y otros, (1.983)), o basados en Programación Dinámica, que reducen la enumeración de todas las soluciones posibles, asegurando la obtención de la solución óptima. Sin embargo, el tiempo de computación al ser función exponencial del número de ciudades, hace "prohibitivo" su uso para un número grande de ciudades. Entre los algoritmos Branch & Bound destacan los de Little, (1.963), y Held & Karp, (1.970) y (1.971). El tiempo de computación en estos algoritmos es de orden $\theta(2^n)$. Menos eficiente es la resolución usando exclusivamente técnicas de Programación Dinámica, (Desrochers y otros, (1.988)); sin embargo éstas pueden ser útiles para hallar cotas que después sean introducidas en algoritmos de tipo Branch & Bound.

2.- **Algoritmos Heurísticos** o de aproximación, que no aseguran la obtención de la solución óptima, sino de una solución que suele estar próxima a la óptima, en un tiempo de computación menor (que normalmente crece polinomialmente en función del número de ciudades). Se pueden distinguir las siguientes grandes familias o grupos:

- Constructivos: Van *construyendo* una ruta de forma secuencial. Por ejemplo los de Inserción, Ahorros, Aproximación Geométrica, etc... (Golden y otros, (1.980)).
- De Mejora: A partir de una solución inicial determinan otra con menor coste (o distancia total) mediante una serie de cambios. Destacan fundamentalmente en este grupo los algoritmos de intercambio r-óptimos. (Lin, (1.965)).
- De Búsqueda Incompleta: Básicamente son algoritmos de tipo Branch & Bound en los que se *corta* o impide la *exploración* de determinadas ramas en el árbol de búsqueda. A diferencia de los correspondientes algoritmos Branch & Bound no garantizan que las ramas que no se exploran no contengan la solución óptima (Aragón y Pacheco, 1.992). Otros como el de Carlier y Villon, (1.990), se basan en un método de Programación Dinámica en el que se reduce la búsqueda de la solución a un número polinomial de rutas.

- Compuestos: Combinan dos o más técnicas anteriores; por ejemplo aplican un intercambio r-óptimo a una solución dada por un algoritmo de inserción o de búsqueda incompleta. (Golden y otros, (1.980)).

Por lo general, los heurísticos son los más empleados especialmente en la actividad empresarial, sobretodo cuando el tamaño del problema es grande.

En el trabajo de Nurmi, (1.991) se proponen modificaciones en algunos de estos algoritmos, para su programación en lenguaje C, realizándose además un estudio comparativo en un ordenador personal.

1.1. FORMULACION DEL TSP

Sea c_{ij} la distancia de la ciudad i a la j, y las variables:

$$x_{ijk} = \begin{cases} 1, & \text{si el tramo k va de la ciudad i a la j} \\ 0, & \text{en caso contrario} \end{cases}$$

la función a minimizar será:

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \sum_{k=1}^{n} c_{ij} \cdot x_{ijk}$$

sujeto a las siguientes condiciones:

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} \sum_{k=1}^{n} x_{ijk} = 1, \quad i=1,...,n; \quad \text{(de cada ciudad sólo se sale una vez)}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ijk} = 1, \quad k=1,...,n; \quad \text{(cada tramo k sólo une dos ciudades)}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} \sum_{k=1}^{n} x_{ijk} = 1, \quad j=1,...,n; \quad \text{(a cada ciudad sólo se llega una vez)}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ijk} = \sum_{\substack{r=1 \\ r\neq j}}^{n} x_{jrk+1}, \quad j=1,...,n; \quad k=1,...,n-1;$$

(el punto de llegada del tramo k es el punto de partida del tramo k+1),

$$\Sigma_{j=2}^{n} x_{1j1} = 1; \quad \text{(en el primer paso se sale de 1)},$$

$$\Sigma_{j=2}^{n} x_{j1n} = 1; \quad \text{(en el último paso se llega a 1)}.$$

Otra formulación clásica, que usa únicamente dos subíndices, es la de Miller, Tucker y Zemlin, (1.960):

$$\text{minimizar} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} c_{ij} x_{ij}$$

sujeto a las siguientes condiciones:

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1, \quad j = 1,...,n;$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1, \quad i = 1,...,n;$$

$$u_i - u_j + n x_{ij} \leq n - 1, \quad i,j = 2,...,n; \; i \neq j;$$

$$x_{ij} \in \{0,1\}, \quad i,j = 1,...,n; \; i \neq j$$

$$u_i \in Z^+, \quad i = 2,...,n;$$

siendo

$$x_{ij} = \begin{cases} 1, & \text{si la ciudad j se visita inmediatamente después de i} \\ 0, & \text{en caso contrario} \end{cases}$$

Observaciones

Un TSP se dice simétrico si la distancia de la ciudad i a la j es la misma que de la j a la i, esto es $c_{ij} = c_{ji}$, y asimétrico en caso contrario.

Un TSP cumple la desigualdad triangular si para cualesquiera i,j,k $\in \{1,...,n\}$, se verifica $c_{ij} \leq c_{ik} + c_{kj}$. Este problema se conoce como *TSP Euleriano.*

Siempre se puede conseguir un TSP con esta propiedad redefiniendo la matriz de distancias a partir de la *menor distancia entre dos ciudades* aunque esto suponga pasar por una tercera.

Como las ciudades pueden considerarse nodos de un grafo consideraremos sinónimos los términos *nodo* y *ciudad*, utilizando el más adecuado según el contexto.

2.- *ALGORITMOS EXACTOS*

Entre los algoritmos exactos destaca principalmente el algoritmo de Little y otros, (1.963). Se usa para matrices asimétricas, está basado en un proceso recursivo de reducción de la matriz de distancias y ha sido adaptado a otros problemas de rutas como el VRP, (Christofides y Eilon, (1.969)), o el TSP con carga y descarga, (Kalantari y otros, (1.985)). Otros algoritmos exactos son el de Eastman, (1.958), también para matrices asimétricas; el de Held & Karp, (1.970) y (1.971), basado en los *1-árboles de expansión*, al igual que el propuesto por Volgenant y Jonker, (1.982), ambos para matrices simétricas. Por su adaptabilidad a problemas con ventanas de tiempo, destacan las técnicas de *relajación del espacio de estados* de Christofides, Mingozzi y Toth, (1.981).

Todos los algoritmos exactos requieren tiempo exponencial. En el apartado correspondiente al algoritmo de Held & Karp se proponen unas modificaciones que reducen enormemente el tiempo de computación.

2.1.- ALGORITMO DE LITTLE

En 1.963 Little y otros, (1.963), describieron una técnica Branch & Bound para el TSP que durante años ha sido el algoritmo exacto más utilizado y que más veces se ha adaptado para la resolución de otros problemas de rutas más complejos como el VRP. Como todos los métodos Branch & Bound esta basado en un árbol de búsqueda donde en cada paso todas las posibles soluciones del problema son particionadas en dos o más subconjuntos. En este caso la partición da lugar a dos subconjuntos: uno con las soluciones que contienen un arco específico (i,j) y el otro no.

El método básico para elegir el arco (i,j), a partir del cual se divide el conjunto de soluciones, y obtener las cotas inferiores en cada uno de estos subconjuntos, consiste en un proceso de reducción de la matriz de costes (o distancias) basado en el Teorema 1.1.

Teorema 1.1.

Dada una solución factible $\mathbf{x}=(x_{ij})$ del TSP, sean $p_i = \min_{j=1,\ldots,n}\{c_{ij}\}$, $q_j = \min_{i=1,\ldots,n}\{c_{ij} - p_i\}$, $c'_{ij} = c_{ij} - p_i - q_j$. Se verifica que:

$$c'_{ij} \geq 0, \quad \text{y} \quad z = \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij}x_{ij} = \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c'_{ij}x_{ij} + \Sigma_{i=1}^{n} p_i + \Sigma_{j=1}^{n} q_j.$$

Observación.-

El Teorema 1.1. muestra que minimizar $\Sigma_{i,j}\, c_{ij}x_{ij}$ equivale a minimizar $\Sigma_{i,j}\, c'_{ij}x_{ij}$ y que $\Sigma_i p_i + \Sigma_j q_j$ es una cota inferior para el valor óptimo de z.

En cada vértice del árbol de decisión, el proceso de reducción consiste en obtener **C'** a partir de **C**. Inicialmente se toma el coste total 0. La cantidad $\Sigma_{i=1}^{n} p_i + \Sigma_{j=1}^{n} q_j$, se va a añadir al coste total acumulado, y va a ser la cota inferior de esta rama. Se elige para la ramificación el arco (i,j) que menos aumente el coste total, por tanto se elige un arco correspondiente a un 0 en la matriz **C'**, ya que **C'** tiene al menos un 0 por fila y columna. Una vez elegido el arco (i,j), se particiona o ramifica el conjunto de soluciones actuales en las que incluyen el arco (i,j), *rama izquierda*, y las que no, *rama derecha*.

Las operaciones a realizar en la rama izquierda son las siguientes:

- Se ha de asegurar que no se incluya en posteriores pasos arcos que *salgan* de i ni que *lleguen* a j.
- Se ha de asegurar que en pasos posteriores no se elija el arco (j,i).
- Se ha de prevenir la elección de arcos en pasos posteriores que puedan formar subrutas con los arcos ya seleccionados en esa rama.

En la rama derecha, para impedir la posible elección del arco (i,j) en ese paso y posteriores, se hace $c'_{ij} = \infty$.

Una vez realizadas las transformaciones de **C'** en cada una de las ramas, se redefine **C = C'**, y se repite el proceso de reducción y ramificación en cada una de las ramas.

Descripción del Algoritmo en seudocódigo

Para la descripción de la exploración en cada vértice α y del algoritmo principal se definen los siguientes parámetros y variables:

R(α): conjunto de arcos que forman parte de la ruta actual;

F(α): conjunto de arcos que no pueden añadirse a la ruta actual;

C*(α): matriz de distancias asociada al vértice α;

coste(α): coste total acumulado hasta el vértice α;

Distminima: coste de la mejor solución obtenida hasta el momento;

Soluc: conjunto de arcos que forman dicha solución;

Disttotal: coste de la solución que se halla en cada momento;

cotainf: cota inferior determinada tras el proceso de reducción.

Procedimiento EXPLORACION(R(α), F(α), **C***(α), coste(α))

$\forall(i,j) \in F(\alpha)$ poner $c^*_{ij}(\alpha) = \infty$;

Si los arcos de R(α) forman una ruta:

Poner *Disttotal* = suma de los costes de los arcos de R(α);

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

Distminima = *Disttotal*; *Soluc* = R(α); Parar;

en caso contrario, hacer:

Poner en **C'** la matriz **C***(α) reducida

Poner *cotainf* = coste(α) + $\Sigma_i p_i + \Sigma_j q_j$

Si *cotainf* < *Distminima* entonces hacer:

Seleccionar (i*,j*) correspondiente a un 0 de **C'**;

Poner R(α*) = R(α) ∪ (i*,j*);

{Prevención de Subrutas}

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

Poner *ult* = último nodo de la cadena de R(α) donde está j*;

Poner *pri* = último nodo de la cadena de R(α) donde está i*;

Poner F(α*) = F(α) ∪ (*ult*, *pri*)

en caso contrario Poner F(α*) = F(α);

{Fin de Prevención de Subrutas}

Poner en **C***(α*) la matriz **C'** sin la fila i* y la columna j*;

Poner coste(α*) = *cotainf;*

Ejecutar EXPLORACION(R(α*), F(α*), **C***(α*), coste(α*));

Poner: R(α**) = R(α); F(α**) = F(α) ∪ (i*,j*); **C***(α**) = **C'**;

coste(α^{**}) = *cotainf;*

Ejecutar EXPLORACION(R(α^{**}), F(α^{**}), **C*(α^{**})**, coste(α^{**})).

Parar.

Procedimiento PRINCIPAL

Leer datos iniciales: n y C; poner *Distminima* = ∞;
Ejecutar EXPLORACION($\varnothing$, $\varnothing$, C, 0)
Finalizar.

2.2.- ALGORITMO DE HELD y KARP

Más recientemente, Held & Karp, (1.970) y (1.971), proponen una estrategia de solución Branch & Bound para hallar una solución exacta al TSP simétrico, que utiliza los mínimos árboles de expansión tanto en el proceso de ramificación como el de acotamiento. Para explicar este proceso sea K_n un grafo completo no dirigido en los nodos {1,...,n}, y c_{ij} el peso asociado a la arista (i,j). Un 1-árbol consiste en un árbol definido sobre los nodos {2,3,...,n}, junto con dos aristas incidentes en el nodo 1. Por tanto, un 1-árbol de mínimo peso se puede encontrar a partir de un mínimo árbol de expansión en los vértices {2,3,...,n} añadiendo las aristas de menor coste incidentes en el nodo 1 como ilustra la figura 1.

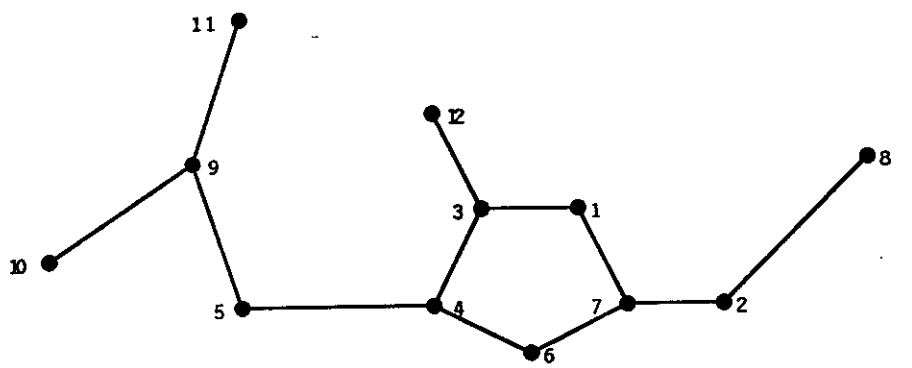


Figura 1. Ejemplo de 1-Arbol.

Se demuestra que si un 1-árbol de mínimo peso o coste es una ruta, entonces es una solución al TSP definido en este grafo.

Lema 1.1.-

Sea $\pi^T = (\pi_1, \pi_2, ..., \pi_n) \in IR^n$; si $\mathbf{x} = (x_{ij})$ es una ruta óptima para la matriz de costes $\mathbf{C} = (c_{ij})$, entonces lo es también para la matriz $\mathbf{C'} = (c'_{ij})$, siendo $c'_{ij} = c_{ij} + \pi_i + \pi_j$.

La elección de π afecta a la búsqueda del 1-árbol de mínima expansión o coste. Por tanto una posible estrategia para el TSP consiste en buscar, si es posible, un vector π de tal forma que un 1-árbol de mínima expansión con respecto a $c_{ij} + \pi_i + \pi_j$ sea una ruta.

Función hueco. Propiedades.-

Se define $f(\pi)$, función *hueco*, como la cantidad en la que el coste de la ruta óptima supera al coste del 1-árbol de mínima expansión, ambos con respecto a $c_{ij} + \pi_i + \pi_j$. Obviamente es, $f(\pi) \geq 0, \forall \pi$. Lo que no está garantizado es que $\min_\pi f(\pi) = 0$, sin embargo puede suponer una buena aproximación a la solución del TSP determinar donde se halla este mínimo.

Sean:

W el coste de la ruta óptima con respesto a (c_{ij});

c_k el peso del k-ésimo 1-árbol con respecto a (c_{ij});

d_{ik} el número de aristas incidentes en el nodo i en el k-ésimo 1-árbol;

K = conjunto de 1-árboles;

Lema 1.2.-

Se verifica que:

$$f(\pi) = W + 2\sum_{i=1}^{n} \pi_i - \min_{k \in K}\left[c_k + \sum_{i=1}^{n} \pi_i d_{ik}\right] = W - \min_{k \in K}\left[c_k + \sum_{i=1}^{n} \pi_i (d_{ik} - 2)\right].$$

En virtud del lema 1.2. minimizar $f(\pi)$ equivale a maximizar $w(\pi) = \min_{k \in K}\left[c_k + \sum_{i=1}^{n} \pi_i v_{ik}\right]$, donde se ha hecho $v_{ik} = d_{ik} - 2$. En forma vectorial $w(\pi) = \min_{k \in K}[c_k + \pi^T \cdot v_k]$. Será $w(\pi) \leq W$, ya que $f(\pi) \geq 0$.

Cálculo Iterativo del vector π

Se observa que $w(\pi)$ puede servir de cota inferior para el TSP, y mejor aún $\max_{\pi} w(\pi)$. En los trabajos mencionados se explican diversos métodos para aproximarse a $\max_{\pi} w(\pi)$, concretamente se propone el siguiente proceso iterativo:

$$\pi^{m+1} = \pi^m + t_m v_{k(\pi^m)};$$

siendo $k(\pi)$ el índice del 1-árbol de menor peso en π, es decir, respecto a $(c_{ij} + \pi_i + \pi_j)$, y t_m una sucesión de escalares (suelen tomarse $t_m = t$ constantes).

Descripción del Algoritmo en seudocódigo

Para describir el procedimiento de exploración en cada vértice del árbol de búsqueda se consideran los siguientes parámetros de estado (X, Y, π), donde

π es el vector de penalizaciones en ese momento;

X e Y son subconjuntos de aristas disjuntas del grafo inicial K_n;

y se define la siguiente función $W_{X,Y}(\pi) = \min_{k \in \Gamma(X,Y)} (c_k + \pi^T \cdot v_k)$, siendo $\Gamma(X, Y)$ el conjunto de 1-árboles que incluyen las aristas de X, y excluyen las de Y.

Además se consideran las siguientes variables:

Costeminimo: Coste de la mejor solución obtenida hasta ese momento;

Ruta: Conjunto de aristas que forman la mejor solución obtenida hasta el momento;

Contadormax: Número de iteraciones sin mejora en la cota inferior desde la última mejora;

Numaristas(i) : Número de aristas incidentes en cada nodo i en cada iteración;

$w1$: Mayor valor de la cota inferior obtenida en ese momento;

$\pi 1$: Valor de π para el que se obtiene el valor de $w1$;

y el parámetro MAXITERTOT: Máximo número de iteraciones sin mejora en la cota inferior permitido;

Procedimiento EXPLORACION(X, Y, π);

Si las aristas de X forman un ruta:

Poner *costetot* = suma de los costes de las aristas de X;

Si *costetot* < *Costeminimo*, entonces

poner: *Costeminimo* = *costetot*; *Ruta* = X;

Parar;

en caso contrario hacer:

$\pi 2 = \pi$, w1 = $-\infty$, *Contadormax* = 0. {Inicializar variables}

Repetir

Si $W_{X,Y}(\pi 2) > w1$ entonces poner: *Contadormax* = 0, $w1 = W_{X,Y}(\pi 2)$, $\pi 1 = \pi 2$;

en caso contrario poner *Contadormax* = *Contadormax*+1.

Definir *Numaristas*(i) como el número de aristas incidentes en el nodo i

correspondientes al cálculo de $W_{X,Y}(\pi 2)$;

Poner v(i) = *Numaristas*(i) - 2, i = 1, 2,..., n;

Poner $\pi 2 = \pi 2 + t2 \cdot v$.

Hasta ((*Contadormax* = MAXITERTOT) o ($w1 >= Costeminimo$) o (v = 0))

Si $w1 >= Costeminimo$ parar;

en caso contrario

Si v = 0 hacer:

Ruta = conjunto de aristas que forman el 1-árbol

correspondiente al cálculo de w1,

Costeminimo = suma de los costes de *Ruta*,

Parar;

en caso contrario hacer.

Seleccionar los aristas e_1, e_2, ..., e_q, que se van añadir a X e Y;

Para i = 1,...,q, hacer:

Determinar los conjuntos X_i e Y_i,

Ejecutar EXPLORACION(X_i,Y_i, π1);

Parar.

Procedimiento PRINCIPAL

Leer datos iniciales: n número de nodos y $\mathbf{C} = (c_{ij})$ matriz de distancias.
Poner *Costeminimo* = ∞, $X = \varnothing$, $Y = \varnothing$, $\pi = (0)$.
Ejecutar EXPLORACION(X, Y, π).
Parar.

Los conjuntos X_i e Y_i se construyen como sigue: las aristas que no están presentes en los conjuntos X e Y se ordenan según el valor de la cota que proporcionan si son excluidas, esto es:

$$W_{X,Y\cup\{e_1\}}(\pi') \geq W_{X,Y\cup\{e_2\}}(\pi') \geq .... \geq W_{X,Y\cup\{e_r\}}(\pi'),$$

Los conjuntos X_i e Y_i que se consideran son los siguientes:

$X_1 = X$, $Y_1 = Y \cup \{e_1\}$,
$X_2 = X \cup \{e_1\}$, $Y_2 = Y \cup \{e_2\}$,
$X_3 = X \cup \{e_1, e_2\}$, $Y_3 = Y \cup \{e_3\}$,
.....
.....
$X_q = X \cup \{e_1, e_2, ..., e_{q-1}\}$, $Y_q = Y \cup R_i$,

siendo q el menor entero para el cual hay un nodo i tal que en X no hay dos aristas incidentes en él pero en X_q sí; y R_i el conjunto de aristas incidentes en este vértice i que no están en X_q (no debe de haber más de dos aristas incidentes en cada vértice). Si hubiera dos o más vértices i,j de esta forma, entonces $Y_q = Y \cup R_i \cup R_j$.

2.3.- PROGRAMACION DINAMICA: METODOS DE RELAJACION

En principio, pocos problemas combinatorios pueden resolverse eficientemente usando exclusivamente la Programación Dinámica. Esto se debe a que el número de vértices del grafo asociado al espacio de estados posibles suele ser enorme. Christofides, Mingozzi y Toth, (1.981), proponen un procedimiento de *relajación* general para reducir el número de estados posibles del grafo asociado. Así, con la formulación recursiva de programación dinámica aplicada a este espacio de estados reducido, se consiguen *rápidamente* cotas que pueden aplicarse a algoritmos Branch & Bound para la resolución de estos problemas.

Concretamente para el TSP se realiza el siguiente planteamiento: Sea la red G = (X,A), $\mathbf{C}=(c_{ij})$ la matriz de costes o distancias y x_1 la ciudad inicial y final. Se define $\Gamma^{-1}(x_i) = \{x_j \in X / (x_j,x_i) \in A\}$. Si G es completa, entonces, $\Gamma^{-1}(x_i) = X-\{x_i\}$.

Sea $x_i \in S, S \subseteq X-\{x_1\}$, se define $f(S,x_i)$ como el coste del camino mínimo que comienza en x_1, recorre todos los vértices de S y acaba en x_i. Una fórmula recursiva de programación dinámica para calcular $f(S, x_i)$ es:

$$f(\{x_i\},x_i) = c_{1i};$$

$$f(S,x_i) = \min_{x_j \in S-\{x_i\}} \left[f(S-x_i,x_j) + c_{ji} \right], \forall x_i \in X, S \subseteq X-\{x_1\};$$

y la solución óptima viene dada por:

$$\min_{x_i \in X} \left\{ f(X',x_i) + c_{i1} \right\}, \qquad \text{donde } X' = X-\{x_1\}.$$

Este procedimiento recursivo, proporciona un método para hallar el camino mínimo en el grafo cuyos vértices son los estados (S,x_i), y los arcos representan la transición de un estado a otro. El número de vértices en este grafo es enorme.

Christofides, Mingozzi y Toth, (1.981), proponen una serie de *relajaciones* mediante proyecciones o aplicaciones de este grafo a otro con menor número de elementos. Estas proyecciones vienen dadas por funciones g(·) aplicadas al conjunto *S*, de tal forma que el nuevo conjunto de estados sea de menor cardinalidad. Algunas de estas funciones, propuestas por los autores citados, se muestran a continuación:

2.3.1.- RELAJACIONES G(·) DEL CONJUNTO S PARA EL TSP

Relajación Cardinal o del n-camino más corto.- Sea $g(S) = |S|$ el cardinal del conjunto S.

Entonces $g(S-\{x_i\}) = g(S)-1$, y la recursión queda de la forma siguiente:

$$f(k, x_i) = \min_{x_j \in E^{-1}(k,x_i)} \left[f(k-1, x_j) + c(x_j, x_i) \right],$$

cuya condición inicial es $f(1,x_i) = c_{1i}$.

Siendo $E^{-1}(k,x_i)$ el conjunto de vértices x_j de $X-\{x_1\}$, tales que en el grafo formado por el nuevo conjunto de estados los vértices $(k-1,x_j)$ y (k,x_i) son principio y fin de algún arco. Lógicamente, para $k \geq 2$, $E^{-1}(k,x_i) = \Gamma^{-1}(x_i)$.

Otras relajaciones son:

Relajación del q-camino más corto.- Corresponde a $g(S) = \sum_{x_j \in S} q_j$; siendo $q_i \geq 1$, $i \in S$ un conjunto de números asociados a cada uno de los nodos $x_i \in X - \{x_1\}$, $(q_1 = 0)$.

Relajación del (n-q)-camino.- Corresponde a $g(S) = (|S|, \sum_{x_j \in S} q_j)$.

Relajación del doble q-camino.- Se define $g(S) = \left(\Sigma_{x_j \in S} q_j^{(1}, \Sigma_{x_j \in S} q_j^{(2} \right)$. En este caso se asocian dos enteros $q_j^{(1}$ y $q_j^{(2}$ a cada nodo x_i.

2.3.2.- COTAS QUE PROPORCIONAN LAS RELAJACIONES G(·)

De la relajación cardinal. se puede obtener la siguiente cota B_1, utilizando la formula recursiva relajada:

$$B_1 = \min_{x_i \in \Gamma^{-1}(x_1)} \left[h(x_i) + c(x_i, x_1) \right];$$

donde $h(x_i) = f(n-1,x_i)$ es el coste del menor camino, (n-camino), que comienza en x_1, finaliza en x_i y conteniene n-1 arcos. De forma análoga se pueden conseguir cotas B_2, B_3 y B_4 correspondientes al resto de las relajaciones.

Otro tipo de cotas más ajustadas se pueden obtener de forma parecida pero *completando* el circuito. Para ello supóngase que, de igual forma que antes con la función f original, se define $f(S,x_i)$ como el coste mínimo de recorrer todos los nodos de S, comenzando por x_i y finalizando en x_1, (camino inverso). Se tiene que:

$$f'(S,x_i) = \min_{x_j / x_j \in \Gamma^{-1}(x_i)} [f(S-\{x_i\},x_j)+c_{ij}] \qquad \text{y} \qquad f'(\{x_i\},x_i) = c_{i1}.$$

Aplicando la relajación definida por g(S) a f':

$$f'(g(S),x_i) = [f(g(S-\{x_i\}),x_j)+c_{ij}] \qquad \text{y} \qquad f'(g(\{x_i\}),x_i) = c_{i1}.$$

De esta forma, para $g(S) = |S| = k$ se puede definir $\Psi(k,x_i)$ como el menor coste de un circuito, en el conjunto de espacios relajados, de n elementos que empieza y acaba en x_1 y con x_i en la k-ésima posición; $\Psi(k,x_i)$ se puede calcular como

$$\Psi(k,x_i) = f(k,x_i) + f'(n-k,x_i).$$

Llamando $b_{ik} = \Psi(k,x_i)$, para i=2,...,n, y k=1,...,n-1, se obtienen nuevas cotas:

(1) El valor de la solución del problema de asignación correspondiente a la matriz (b_{ik}), es una cota inferior para el TSP.

(2) Cualquier cota inferior de este problema de asignación lo es del TSP, por ejemplo:

$$\max_{x_i}\left[\min_{k=1..n-1} b_{ik}\right].$$

3.- ALGORITMOS HEURISTICOS

Estos algoritmos se pueden clasificar en cuatro grupos: Procedimientos de Construcción, de Mejora, de Búsqueda Incompleta y Procedimientos Compuestos. A continuación se describen los más notables incluyendo información sobre su complejidad y eficacia.

3.1.- ALGORITMOS DE CONSTRUCCION

Construyen una ruta añadiendo sucesivamente ciudades hasta incluir todas. Sirven para matrices asimétricas.

3.1.1.- VECINO MAS CERCANO

Esta técnica propuesta por Rosenkrantz y otros, (1.974), consta de los siguientes pasos:

Paso 1: Determinar el nodo inicial de la ruta.

Paso 2: Encontrar el nodo más cercano al último nodo añadido, y añadirlo a la ruta.

Paso 3: Repetir el paso 2 hasta que todos los nodos estén contenidos en el camino.
Entonces unir el primero y el último nodo.

El número de operaciones requeridas para determinar la solución es $\theta(n^2)$, tal como está descrito el algoritmo, y el comportamiento *en el peor de los casos* satisface:

$$\frac{\text{longitud de la ruta 'vecino cercano'}}{\text{longitud de la ruta óptima}} \leq \frac{1}{2}\left[\lg_2(n)\right]+\frac{1}{2} .$$

3.1.2.- AHORROS DE CLARKE y WRIGHT

El siguiente algoritmo es una variante desarrollada por Golden, (1.980), para el TSP, del conocido algoritmo de los ahorros de Clarke & Wrigth para el VRP, (1.964), y su descripción es:

Paso 1: Seleccionar el nodo 1, correspondiente a la ciudad inicial.

Paso 2: Calcular los ahorros $s_{ij} = c_{1i} + c_{1j} - c_{ij}$, para $i,j = 2,3,...,n$.

Paso 3: Ordenar los ahorros de mayor a menor.

Paso 4: Comenzando por el mayor de los ahorros unir los nodos correspondientes a estos ahorros de forma que se vayan formando subrutas cada vez mayores. Repetir hasta que se forme una ruta completa.

Existe una versión *secuencial* consistente en seleccionar sólo el mayor de los ahorros en el paso 4, y repetir los pasos 2, 3 y 4 en sucesivas iteraciones hasta formar una ruta.

En general no se conoce el comportamiento de estas técnicas *en el peor de los casos*, sin embargo, Golden, (1.980), demuestra que para la versión secuencial el cociente entre la longitud de la ruta heurística y la longitud de la ruta óptima está acotado por una función lineal de $\lg_2(n)$

El número de cálculos que se deben realizar es de orden $\theta\left(n^2 \cdot \lg_2(n)\right)$; el paso 2 requiere un número de operaciones múltiplo de n^2; para el paso 3 de ordenación de los ahorrros hacen falta un máximo de $\theta\left(n^2 \lg_2(n)\right)$ cálculos aplicando algoritmos de ordenación rápida, como los de Williams, (1.964) y Floyd, (1.964); finalmente el paso 4 requiere a lo sumo un múltiplo de n^2 soluciones.

3.1.3.- ALGORITMOS DE INSERCION

Un procedimiento de inserción básicamente consiste en lo siguiente: se elige un nodo inicial, en la k-ésima iteración se tiene una subruta con k nodos, se determina cual de los nodos que no está en la subruta se debe añadir a ésta (paso de selección), y se determina en que tramo de la subruta debe ser insertado (paso de inserción). Si se prueba con cada uno de los nodos como nodo inicial el número de cálculos del procedimiento completo se multiplica por n.

Algunas de las principales variantes de estos algoritmos, descritas por Rosenkrantz, Stearns & Lewis, (1.974), se describen a continuación:

3.1.3.1.- Inserción más cercana.

Paso 1: Comenzar con un subgrafo formado por un nodo i arbitrario.

Paso 2: Encontrar un nodo k tal que c_{ik} sea mínima y formar una subruta i - k - i.

Paso 3 (Paso de Selección): Dada una subruta encontrar el nodo k que no esté en ella, más cercano a la subruta actual.

Paso 4 (Paso de Inserción): Encontrar el arco (i,j) en la subruta actual que minimice $c_{ik}+c_{kj}-c_{ij}$. Insertar k entre i y j.

Paso 5: Si la subruta actual contiene las n ciudades parar; en otro caso volver al paso 3.

Comportamiento *en el peor de los casos*:

$$\frac{\text{longitud de la ruta 'Inserción más cercana'}}{\text{longitud de la ruta óptima}} \leq 2.$$

El número de cálculos que requiere este algoritmo es de orden $\theta(n^2)$.

3.1.3.2.- Inserción más barata.

Este procedimiento difiere del de inserción más cercana en que los pasos 3 y 4 son remplazados por el siguiente paso:

Paso 3': Encontrar un arco (i,j) en la subruta actual y un nodo k que no lo esté, de tal forma que $c_{ik}+c_{kj}-c_{ij}$ sea mínimo. Insertar k entre i y j.

El comportamiento *en el peor de los casos* es similar al del algoritmo de inserción más cercana.

El número de cálculos que requiere este algoritmo es de orden $\theta(n^2 \lg_2(n))$.

3.1.3.3.- Inserción arbitraria.

Difiere de la inserción más cercana en que en el paso 3, el nodo k a insertar se elige arbitrariamente entre todos los nodos que no estén en la subruta.

Comportamiento *en el peor de los casos*:

$$\frac{\text{longitud de la ruta 'Inserción Arbitraria'}}{\text{longitud de la Ruta óptima'}} \leq 2 \cdot \ln(n) + 0'16.$$

El número de cálculos que requiere este algoritmo es de orden $\theta(n^2)$.

3.1.3.4.- Inserción más lejana.

Difiere de la inserción más cercana en que en el paso 3 se elige como ciudad a insertar en la subruta actual la más alejada de ésta.

Tanto el comportamiento *en el peor de los casos* como el número de cálculos requeridos son similares al del algoritmo de inserción más cercana.

3.1.4.- ALGORITMOS BASADOS EN EL CASCO CONVEXO

Estos algoritmos se utilizan sólo para matrices simétricas. Entre los algoritmos constructivos basados en criterios geométricos destacan los basados en el concepto de *casco* o recubrimiento convexo de un conjunto de puntos. El *casco* convexo de un conjunto de puntos se define como el conjunto convexo cerrado *minimal* que contiene a todos los puntos del conjunto dado. En IR^2 su frontera está formada por segmentos que unen algunos de los puntos de este conjunto.

Para aplicar este método es necesario poder representar el conjunto de nodos del problema inicial como puntos en un espacio de dos dimensiones, de tal forma que la distancia euclídea entre las ciudades i y j en ese espacio coincida con la dada por la matriz de costes o distancias c_{ij}.

Lema 1. 3.-

Si H es el *casco* convexo del conjunto de nodos representados en IR^2, de forma que las distancias entre las ciudades coinciden con los costes c_{ij} originales, entonces el orden o secuencia de las ciudades situadas en la frontera de H coincide con su orden o secuencia en la solución óptima.

Demostración.- Norback y Love, (1.977).

Por lo tanto, una aproximación a la solución óptima consiste en determinar el casco convexo H, determinar como subruta inicial la formada por los nodos en la frontera de H en el orden en que se encuentran, y a continuación ir insertando en esta subruta inicial los demás nodos según algún criterio. (Ver figura 2). Por ello estos algoritmos, en cierta forma, también pueden ser considerados algoritmos de inserción.

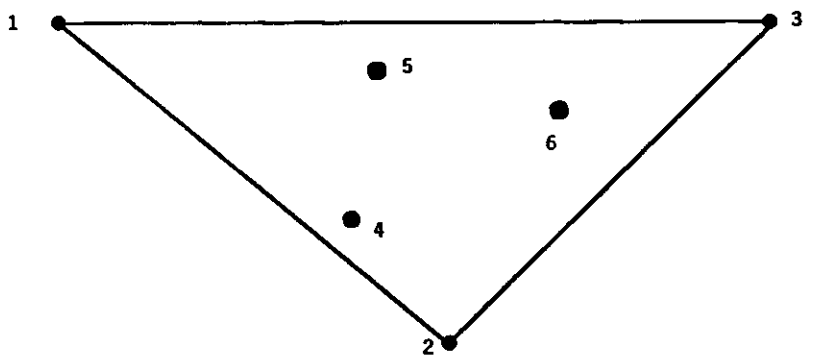


Figura 2. Ejemplo de Casco Convexo.

Construcción del Casco Convexo (Norback & Love, (1.977))

Paso 1: Leer las coordenadas $p_i = (x_i, y_i)$, $i = 1,...,n$. Calcular $x^* = \min\{x_i\}$. Elegir un nodo $h_1 = (x^*, y)$. Poner $t = 1$, ir al paso 2.

Paso 2: Calcular los ángulos α_{ij} entre las semirectas (h_1, p_i), (h_1, p_j), $\forall i,j$. Calcular $\alpha_{i^*j^*} = \max\{\alpha_{ij}\}$.

Paso 3: Poner $h_2 = i^*$ o $h_2 = j^*$. Poner $t = 2$.

Paso 4: Calcular los ángulos β_i entre las semirectas (h_t, h_{t-1}), (h_t, p_i), $\forall i$. Calcular $\beta_{i^*} = \max\{\beta_i\}$. Poner $h_{t+1} = p_{i^*}$. Poner $t = t+1$.

Paso 5: Si $h_t \neq h_1$, volver al paso 4; en caso contrario parar.

La figura 3 ilustra este proceso.

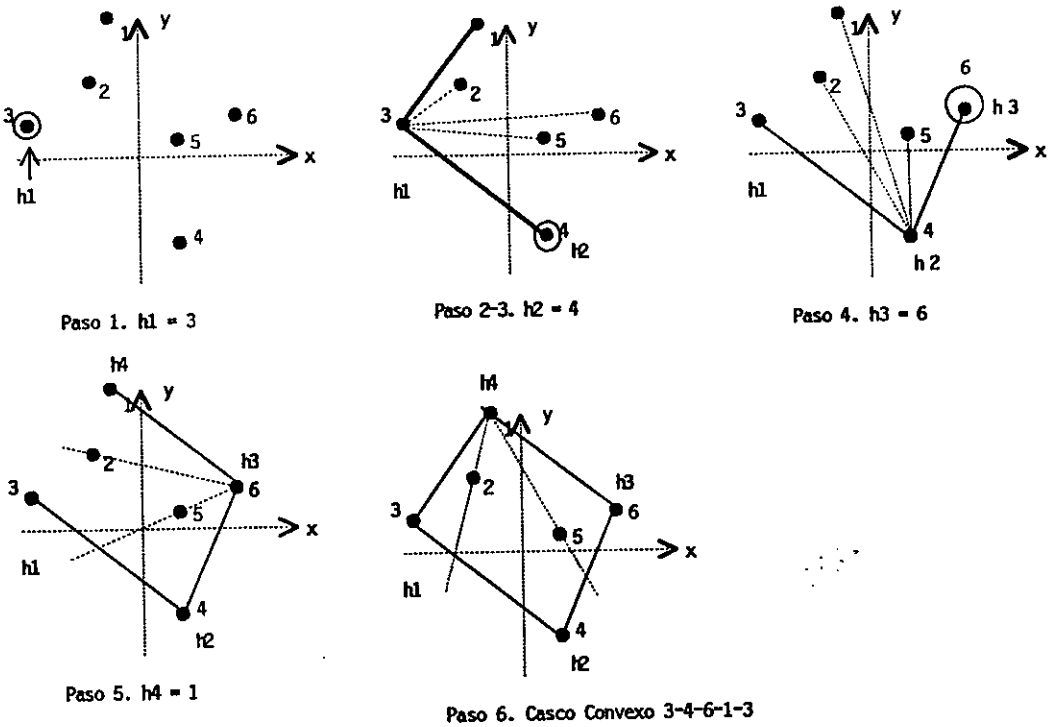


Figura 3. Proceso de Construcción del Casco Convexo.

3.1.4.1.- Algoritmo de Stewart

Paso 1: Formar el *casco* convexo del conjunto de nodos. Este casco determina una subruta inicial.

Paso 2: Para cada nodo k no contenido en la subruta actual, decidir entre que dos nodos i y j consecutivos de la subruta actual se debería insertar k. Es decir, para cada k, determinar el arco (i,j) en la subruta de tal forma que $c_{ik}+c_{kj}-c_{ij}$ sea mínimo.

Paso 3: Para todos los (i,k,j) determinados en el paso anterior determinar el (i*,k*,j*) tal que $(c_{i*k*}+c_{k*j*})/c_{i*j*}$ sea mínimo.

Paso 4: Insertar k* en la subruta entre los nodos i* y j*.

Paso 5: Repetir los pasos 2, 3 y 4 hasta formar una ruta que contenga todas las ciudades.

3.1.4.2.- Variantes de Norback y Love

Norback & Love, (1.977), proponen dos métodos alternativos para insertar los nodos remanentes en la subruta parcial una vez formado el *casco* convexo.

3.1.4.2.1.- Método del Angulo Mayor

Paso 1: Formar la subruta inicial con los nodos que forman el casco convexo.

Paso 2: Calcular los ángulos α_{ijk} formados por las semirectas (i,j), (i,k), $\forall$ j, k nodos consecutivos de la subruta actual y $\forall$ i no perteneciente a ella.

Paso 3: Determinar $\alpha_{i^*j^*k^*} = \max \{\alpha_{ijk}\}$ Insertar i* en la subruta entre los nodos j* y k*.

Paso 4: Si queda algún nodo por seleccionar ir al paso 2; en caso contrario finalizar.

La figura 4 ilustra este método

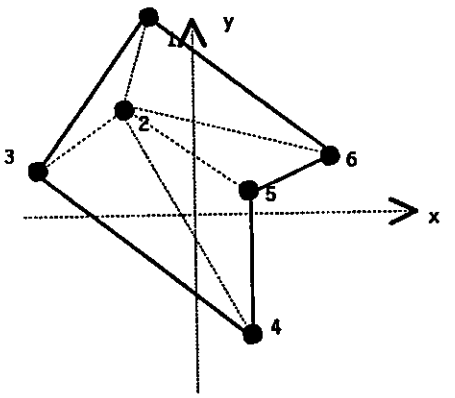
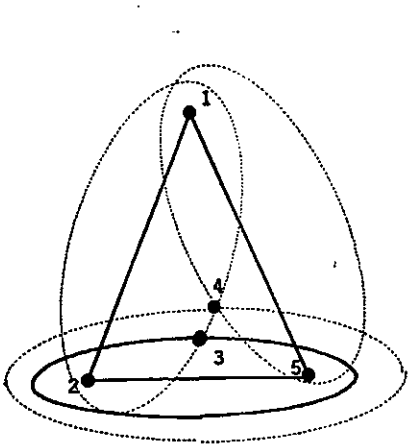


Figura 4. El mayor ángulo es 123. La ruta es 1-2-3-4-5-6-1.

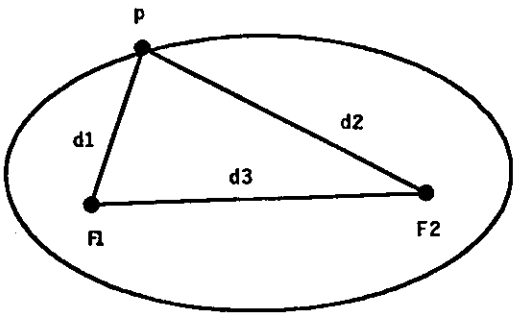
La ruta generada por este método no siempre es óptima. Un problema particular en su aplicación tiene lugar en situaciones en las que hay puntos interiores cercanos. Se puede superar, según los autores, utilizando la siguiente técnica.

3.1.4.2.2.- Método de la Elipse más Excéntrica

El método de la *elipse más excéntrica* parte del *casco* convexo e inserta sucesivamente los nodos remanentes en la subrutas que se forman. Para elegir el punto que va a ser insertado en cada paso, se consideran todas las elipses que se pueden formar tomando cada par de puntos consecutivos en la subruta parcial como focos, y cada nodo interior como punto de esa elipse. Se miden las excentricidades de dichas elipses, se elige la más *excéntrica*, y se inserta el nodo interior correspondiente entre los nodos consecutivos a dicha elipse para dar lugar a una nueva subruta. La figura 5 ilustra esta idea.



La elipse de mayor excentricidad viene dada por los puntos 2,3 y 5



La excentricidad viene dada por d3/(d1+d2)

Figura 5. Elipses de mayor excentricidad.

Estos procedimientos de *Casco Convexo* tienen una complejidad parecida al del algoritmo de Inserción más barata, de orden $\theta(n^2 \lg_2(n))$.

3.1.5.- ALGORITMO HEURISTICO DE CHRISTOFIDES

Christofides, (1.976), propone un algoritmo heurístico basado en los árboles de mínima expansión y en el uso del problema de emparejamientos. Es necesario que la matriz de costes sea simétrica y cumpla la desigualdad triangular. El procedimiento consta de los siguientes pasos:

Paso 1: Encontrar el árbol de mínima expansión T de G.

Paso 2: Identificar todos los nodos con un número impar de aristas incidentes en T. Resolver el problema del *emparejamiento perfecto*, (Apéndice 1.2.), entre estos nodos usando la matriz de costes C. Añadir las aristas correspondientes a los emparejamientos resultantes a las aristas que ya había en T. En el subgrafo resultante todos los nodos tendran un número par de aristas incidentes, y por tanto se podrá obtener un ciclo Euleriano.

Paso 3: Eliminar los subciclos que se formen con los nodos con más de dos arcos incidentes de tal forma que se transforme el ciclo Euleriano en un ciclo Hamiltoniano.

El comportamiento *en el peor de los casos* es:

$$\frac{\text{longitud de la 'Ruta de Christofides'}}{\text{longitud de la Ruta óptima}} \leq 1'5$$

Los resultados que se obtienen son, en general, mejores que los obtenidos por otros algoritmos constructivos, (Golden y otros, (1.980)).

Ya que la base de este procedimiento es el algoritmo de emparejamiento perfecto, que requiere para su resolución $\theta(n^3)$ operaciones, el número de cálculos necesarios para este procedimiento heurístico también va a ser $\theta(n^3)$.

3.2.- ALGORITMOS DE MEJORA

Estos procedimientos se basan en el intercambio de arcos en una ruta inicial para dar otra ruta mejor. Fueron introducidos por Lin, (1.965), y han dado lugar a técnicas de solución tan conocidas como, la de Lin & Kernighan, (1.973), y la de Or, (1.976), para matrices simétricas; posteriormente Webb, (1.971), Kanellakis y Papadimitriou, (1.980), hicieron adaptaciones para el TSP asimétricos. Básicamente proceden de la forma siguiente:

Paso 1: Encontrar una ruta inicial.

Paso 2: Mejorar la ruta actual usando uno de los procedimientos de intercambio de arcos.

Paso 3: Repetir el paso 2 hasta que no se encuentre mejora.

Estos algoritmos se basan en los conceptos de *k-intercambio* de arcos y de rutas *k-óptimas*, que se definen de la forma siguiente:

Definición.-

Un k-intercambio de arcos en una ruta consiste en eliminar k arcos de dicha ruta y sustituirlos por otros k arcos diferentes formándose una nueva ruta. Una ruta se dice que es k-óptima si es imposible obtener otra ruta con menor coste mediante un k-intercambio.

Un procedimiento de intercambio k-óptimo selecciona un k-intercambio que proporcione un coste menor. El algoritmo completo finaliza cuando no hay mejora, es decir cuando se encuentra una solución k-óptima. Algunas cuestiones a considerar son las siguientes:

a) Elección de k: El procedimiento tiene que seleccionar qué conjunto de k arcos en la ruta actual hay que eliminar, y de qué forma se unen los nodos que quedan *libres*. El número de conjuntos posibles es $\theta\binom{n}{k}$. Por tanto, en cada paso, el procedimiento k-óptimo requiere $\theta(n^2)$ cálculos. Cuanto mayor sea el k elegido cabe esperar mejores soluciones, sin embargo el número de cálculos necesarios crece rápidamente con k. Las experiencias de varios autores,

recomiendan la utilización de k=3, o bien combinar k=2 con k=3. Para k > 3, estos algoritmos utilizan en general un tiempo excesivo.

b) Ruta inicial: La ruta inicial suele ser una ruta obtenida por otro heurístico habitualmente constructivo, como el de *inserción más lejana* o el del *vecino más barato*; o bien se toma una obtenida aleatoriamente.

c) Elección del k-intercambio: Habitualmente se elige el intercambio que más mejore la ruta actual. Otra alternativa es elegir el primer intercambio encontrado que mejore la ruta.

3.2.1.- ALGORITMO DE LIN Y KERNINGHAN.

A continuación se describe el algoritmo propuesto por Lin & Kerninghan, (1.973), en el que en cada paso se selecciona el k, para proceder al correspondiente k-intercambio. El planteamiento que hacen estos autores es el siguiente:

Sea una ruta T, considerada como conjunto de aristas, se trata de encontrar un subconjunto $X=\{x_1,x_2,...,x_k\}$ en T, y un subconjunto $Y=\{y_1,y_2,...,y_k\}$ en A-T, de tal forma que al reemplazar X por Y den lugar a una ruta T' con menor coste. Sean los costes de las aristas x_i e y_i respectivamente $|x_i|$ e $|y_i|$, se define $g_i = |x_i|-|y_i|$ la *ganancia* de intercambiar la arista x_i por y_i. Aunque alguno de estos g_i sea negativo, si $\Sigma_{i=1}^{k} g_i = Coste(T)-Coste(T') > 0$, el intercambio, en principio, puede ser considerado. El algoritmo se basa en el siguiente:

Lema 1. 4.-

Toda sucesión finita de números con suma total positiva tiene una permutación cíclica tal que todas las sumas parciales son positivas.

El lema 1.4. muestra que, si se está buscando una sucesión finita de ganancias g_i con suma total positiva, se necesita considerar solamente aquellas que tengan todas las sumas parciales positivas. Este *criterio de ganancias* permite reducir en ocasiones el número de sucesiones a considerar, y es la base del criterio de parada del algoritmo.

El algoritmo básicamente actúa de la forma siguiente en cada iteración: a partir de dos aristas iniciales $x_1 = (t_1, t_2)$, $y_1 = (t_2, t_3)$, tal que $g_1 > 0$, se van eligiendo en pasos posteriores aristas de la forma $x_i = (t_{2i-1}, t_{2i})$ e $y_i = (t_{2i}, t_{2i+1})$, con $|y_i|$ lo más pequeño posible, verificando las siguientes condiciones:

(a) x_i se elige tal que si t_{2i} se uniera a t_1 se obtendría una ruta. (Criterio de *factibilidad*).

(b) Se debe garantizar que los conjuntos X e Y deben ser disjuntos.

(c) Se debe cumplir que $G_i = \Sigma_{k=1}^{i} g_i > 0$. (*Criterio de Ganancia*).

(d) Para asegurar el criterio de factibilidad en la iteración i+1, la elección de y_i debe *permitir* la ruptura de algún posible x_{i+1} posterior.

Los pasos son los siguientes:

Paso 1. Generar una ruta inicial aleatoria T.

Paso 2. Se toma G*=0. (G* es la mayor ganancia que se puede conseguir en cada momento).
Escoger algún nodo t_1 y elegir x_1 tal que sea una de las aristas de T incidentes en t_1.
Poner i = 1.

Paso 3. Desde el otro punto final t_2 de x_1, elegir una arista y_1 con nodo final t_3 tal que $g_1 > 0$. Si no se puede encontrar tal y_1 ir al paso 6.

Paso 4. Poner i = i+1. Escoger x_i (que une t_{2i-1} y t_{2i}) e y_i (que une t_{2i}, t_{2i+1}), con $|y_i|$ mínimo verificando las cuatro condiciones (a), (b), (c) y (d).
Sea y_i^* la arista que conecta t_{2i} y t_1, y sea $g_i^* = |y_i^*| - |x_i|$, si $G_{i-1} + g_i^* > G^*$ poner $G^* = G_{i-1} + g_i^*$, k=i.

Paso 5. Repetir el paso 4, hasta que no se puedan encontrar otros x_i e y_i cumpliendo las cuatro condiciones (a), (b), (c), (d) o hasta que $G_i \leq G^*$ (Este es el *Criterio de parada*).
Si $G^* > 0$ se tiene una nueva ruta T' tal que coste(T')=coste(T)-G*. Volver al paso 2 tomando T' como nueva ruta inicial.

Paso 6. Si $G^* = 0$ volver al paso 4 eligiendo diferentes x_1 e y_1 .

Paso 7. El procedimiento termina cuando los n valores de t_1 se examinan sin mejora.

3.3.- ALGORITMOS DE BUSQUEDA INCOMPLETA

Son básicamente algoritmos Branch & Bound en los que se añaden criterios heurísticos para impedir o aceptar la exploración en determinadas ramas del árbol de búsqueda. Estos criterios heurísticos refuerzan o se añaden a los criterios ya existentes en el algoritmo original. Se trata de impedir la exploración de aquellas ramas del árbol que, según alguno de estos criterios, se sospeche no vayan a contener la solución óptima. No se garantiza que las ramas *rechazadas* no contengan la solución óptima, sin embargo, con algunos criterios puede asegurarse que la solución obtenida se alejará del óptimo a lo sumo una cantidad máxima predeterminada.

En un reciente trabajo, Aragón y Pacheco, (1.992), proponen dos variantes de este tipo para el algoritmo de Little, que denominan: *Rama Izquierda* y *Rechazo Heurístico*.

3.3.1.- VARIANTE RAMA IZQUIERDA

Considérese el cálculo de las cotas inferiores propuesto por Syslo, Deo y Kowalik, (1.983), y la forma de elegir dicho arco (r,c) entre los 0 de la matriz reducida. Esta elección asegura una mayor cota inferior de ese coste en la rama derecha, es decir: coste acumulado en pasos anteriores y valor de la redución de la fila r y columna c en la siguiente iteración en esa rama derecha. Por tanto la primera solución que se obtenga siguiendo las ramas izquierdas, es posible que no esté muy alejada del óptimo, ya que las correspondientes ramas derechas tienen *menos garantías* de incluir la solución.

Así, en esta variante se exploran las ramas izquierdas. De esta forma, el número de veces que se exploran de las ramas izquierdas, hasta obtener una solución, es igual a n. Sin embargo, antes de la elección de cada arco (r,c), es necesaria la reducción de la matriz de costes. El número de cálculos requeridos será en principio, de orden $\theta(n^3)$.

3.3.2- VARIANTE RECHAZO HEURISTICO

En esta variante no se eliminan las exploraciones de la rama derecha, pero se imponen condiciones restrictivas para la realización de dichas exploraciones.

Concretamente se supone que el coste acumulado crece, al menos, según una función creciente f(n-k) del número de arcos que quedan por elegir en esa dirección del árbol, siendo k el número de arcos ya elegidos. Para *admitir* la exploración de la rama derecha, se cambia la condición

cota inferior de esa rama < coste de la mejor solución obtenida hasta el momento

por

cota inferior de esa rama + f(n-k) < coste de la mejor solución obtenida hasta el momento

De esta forma se ahorra la realización de una serie de pasos o exploraciones *supuestamante* innecesarias. El algoritmo de *la Rama Izquierda* es un caso particular de éste con $f(n-k) = \infty$.

La elección de la función f, va a determinar lo máximo que va a estar alejado el coste de la solución obtenida por este procedimiento del coste óptimo:

coste ruta óptima + f(n) > coste ruta 'Rechazo Heurístico';

aunque resultados computacionales muestran que la diferencia, en la mayoría de los casos, dista bastante menos que f(n).

3.4.- ALGORITMOS COMPUESTOS

Estos algoritmos combinan varias de las técnicas anteriores. Fundamentalmente se basan en aplicar procedimientos de intercambio r-óptimos a las rutas obtenidas por la aplicación de algoritmos constructivos o de búsqueda incompleta. De esta forma se aumenta la eficacia de estos procedimientos r-óptimos en dos sentidos: se reduce el tiempo de computación, y la solución final suele ser mejor que la que se obtendría partiendo de una solución aleatoria.

Los procedimientos constructivos, e incluso los de búsqueda incompleta como el de *Rama Izquierda*, son muy rápidos y, aunque se tarda menos tiempo en generar una solución aleatoria, el algoritmo r-óptimo, al partir de una solución mejor, hará menos r-intercambios hasta encontrar una solución r-óptima. De esta forma disminuye el tiempo de cálculo.

3.4.1.- ALGORITMO DE GOLDEN

El procedimiento básico del algoritmo de Golden, (1.980), consta de los siguientes pasos:

Paso 1: Obtener una ruta inicial usando un procedimiento costructivo.
Paso 2: Aplicar un procedimiento 2-óptimo a la ruta obtenida en el paso 1.
Paso 3: Aplicar un procedimiento 3-óptimo a la ruta obtenida en el paso 2.

Este procedimiento compuesto es, computacionalmente, relativamente rápido y ofrece buenos resultados.

Algunas variantes consisten en utilizar rutas iniciales generadas por diferentes algoritmos constructivos partiendo de diferentes nodos iniciales.

3.4.2.- ALGORITMOS DE ARAGON Y PACHECO

Por su parte, Aragón y Pacheco, (1.992), proponen la aplicación de un procedimiento r-óptimo, para r=2 y 3, a la ruta obtenida por el algoritmo de la *Rama Izquierda.* También constatan, mediante experiencias computacionales, la mejora en los resultados, tanto en tiempo como en la calidad de la solución obtenida, respecto al uso de estos procedimientos por separado (r-óptimos y rama izquierda).

4.- CONCLUSIONES Y MEJORAS

4.1.- CONCLUSIONES

Algunas conclusiones que se derivan del comportamiento y eficacia los algoritmos para el TSP, tanto por experiencias propias como por las de otros autores, son las siguientes:

- Los algoritmos heurísticos son preferibles a los exactos, sobretodo si no se dispone de tiempo suficiente para obtener la solución y el tamaño del problema es grande, (Syslo y otros, (1.983)). Otra razón es que los algoritmos heurísticos son conceptualmente más sencillos, y más fáciles de adaptar a problemas más complejos, (Assad, (1.988)). Finalmente los algoritmos exactos son, en general, más difíciles de programar.

- Entre los algoritmos heurísticos los más eficaces son los basados en los intercambios r-óptimos, especialmente el de Lin & Kernighan para matrices simétricas, y el Kanellakis y

Papadimitriou para matrices asimétricas. Sin embargo, mientras en el caso del TSP simétrico estos algoritmos alcanzan la solución óptima en muchas ocasiones o se desvían poco de ella, en el caso del TSP asimétrico esta desviación es mucho mayor, (Nurmi, (1.991)).

- Para problemas de 100 nodos o más, y matrices simétricas, la variante de Or, que utiliza $\theta(n^2)$ operaciones, ha resultado ser muy eficaz, ya que utiliza mucho menor tiempo de computación. De esta forma pueden probarse varias rutas iniciales y llegar a soluciones muy aproximadas a las obtenidas por el de Lin & Kerninghan y mejores que las obtenidas por el 3-óptimo en menos tiempo, (Nurmi, (1.991)).

- En cuanto a los algoritmos constructivos el más eficaz es el propuesto por Christofides, especialmente en matrices que cumplan la desigualdad triangular, (Golden y otros, (1.980)). Entre los algoritmos de inserción el más eficaz parece ser el de inserción más lejana. Esta eficacia se justifica por diversos experimentos computacionales, ya que al ir eligiendo los nodos más alejados, los más cercanos se incluirán posteriormente sin que aumente considerablemente la longitud de la ruta, (Adabinski y Syslo, (1.983), Golden y otros (1.980)).

- En cuanto a los algoritmos de búsqueda incompleta Aragón y Pacheco, (1.992), señalan que, para un número no muy grande de nodos, los algoritmos denominadas *Rama Izquierda* y *Rechazo Heurístico* se desvían muy poco del óptimo en un tiempo de computación relativamente pequeño.
En este trabajo se muestran los resultados computacionales de la aplicación de estos algoritmos a problemas simulados, con diferentes número de nodos. (Las matrices simuladas toman valores enteros de forma uniforme entre 0 y 100. Se ha definido $f(n) = n$)

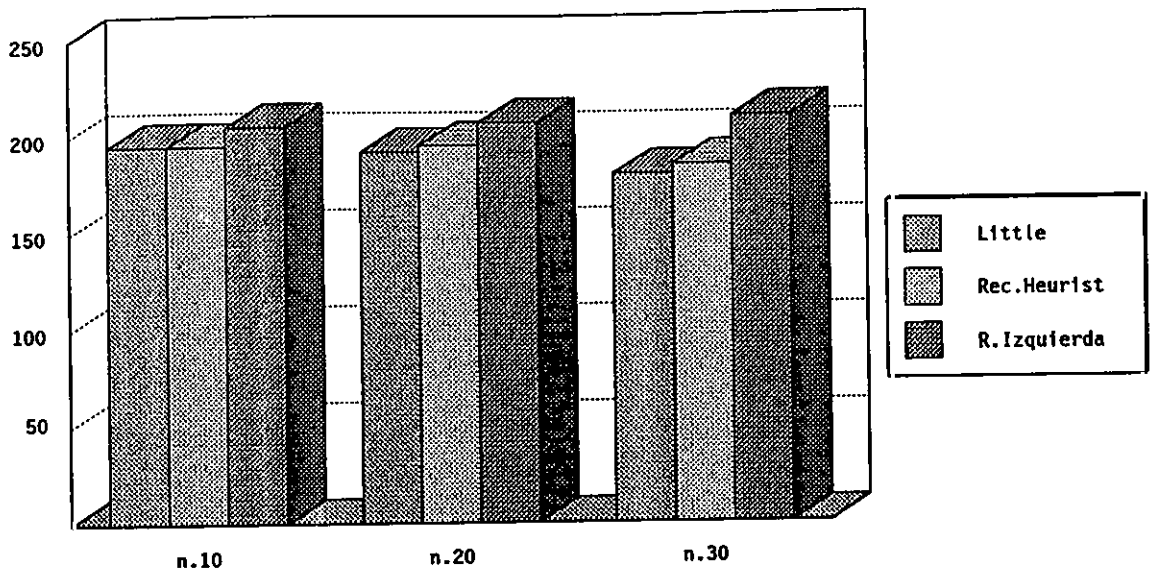


Figura 6. Coste Medio de las rutas obtenidas en función del número de nodos.

	10 nodos	20 nodos	30 nodos
Rechazo Heurístico	0	1'56	2'77
Rama Izquierda	5'12	6'25	16'66

Desviación porcentual de los resultados obtenidos por los algoritmos Rechazo Heurístico y Rama Izquierda respecto a la solución óptima

- Aragón y Pacheco, (1.992), proponen combinar los algoritmos r-óptimos con soluciones iniciales buenas, como las generadas por algoritmos de búsqueda incompleta. Así se obtienen algoritmos compuestos más eficaces que los que surgen de combinar los algoritmos de inserción con los r óptimos.

A continuación, se muestran los resultados de dichas experiencias, en los que se analizan los algoritmos: inserción/2-óptimos y inserción/3-óptimos, *Rama Izquierda*, y los compuestos *Rama Izquierda*/2-óptimo y *Rama Izquierda*/3-óptimo. Se consideraron varios niveles (50, 60, 70 y 80 nodos); en cada uno de estos niveles se chequearon 10 problemas correspondientes a matrices de costes (o distancias) simuladas. La implementación de los algoritmos es debida a los autores.

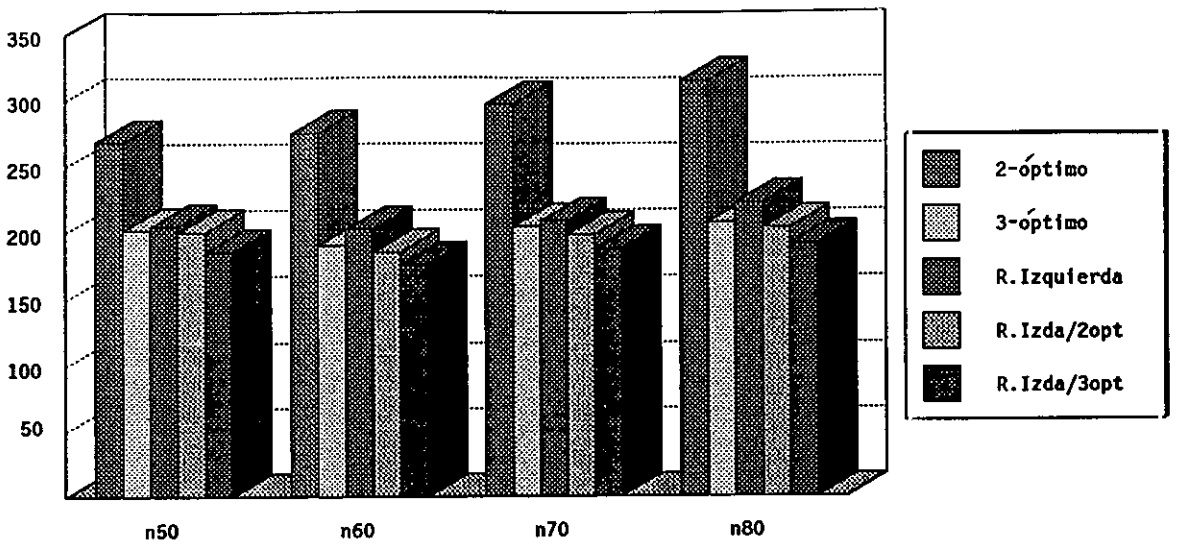


Figura 7. Resultados de los Costes Medios en función del número de nodos para los diferentes algoritmos.

	50 nodos	*60 nodos*	*70 nodos*	*80 nodos*
2-óptimo	268	275	297	315
3-óptimo	201	190	204	207
R.Izquie.	204	202	208	222
RIz/2opt	200	185	198	203
RIz/3opt	185	175	187	191

Resultados de los Costes Medios en función del número de nodos para los diferentes algoritmos. (Datos de la figura 7)

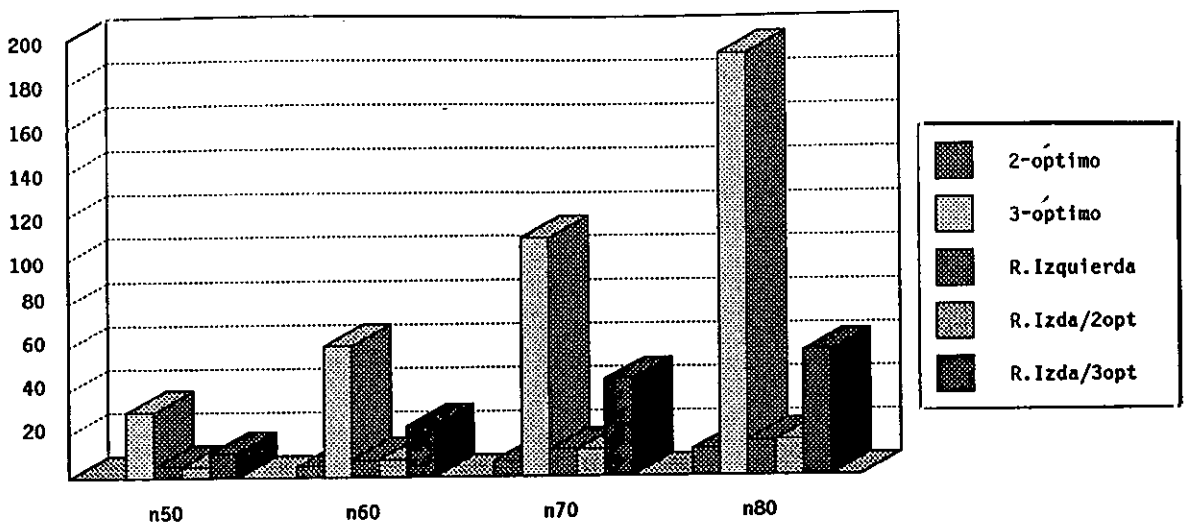


Figura 8. Resultados de los Tiempos Medios de Computación (en segundos) en función del número de nodos.

	50 nodos	***60 nodos***	***70 nodos***	***80 nodos***
2 óptimo	0	5	7	12
3 óptimo	30	60	109	193
R.Izquie.	5	8	12	16
RIz/2opt	5	8	12	16
RIz/3opt	11	23	44	57

Resultados de los Tiempos Medios de Computación en función del número de nodos. (Datos de la Figura 8)

- En los algoritmos exactos se ha observado por experiencias propias, que el tiempo de computación puede reducirse si se emplea una serie de estrategias: utilización de una solución inicial proporcionada por un heurístico efectivo. (r-óptimos, compuestos,...) y generación de una solución rápida en cada vértice con los arcos seleccionados, dando lugar a una cota superior en dicho vértice.

- Sin embargo, a medida que crece el tamaño de los problemas, también se observa que el mejor método para reducir el tiempo es calcular cotas inferiores más ajustadas al óptimo en cada vértice, ya que cuanto mayores sean las cotas inferiores más ramas innecesarias se pueden

dejar de explorar. En este sentido el método basado en los 1-árboles de Held & Karp da cotas inferiores mejores que las obtenidas por el método de reducción de matrices de Little, con tiempos de computación mucho menor.

4.2.- MEJORAS EN EL ALGORITMO DE HELD Y KARP

4.2.1.- IDEA BASICA

A continuación proponemos dos pequeñas modificaciones en el algoritmo de Held & Karp original que afectan al proceso de cálculo de la cota inferior en cada vértice, es decir del cálculo de $\max_{\pi} w(\pi)$, mediante el proceso iterativo $\pi^{m+1} = \pi^m + t_m v_{k(\pi^m)}$ descrito en la sección 2.2.

Frente al criterio de los autores, que proponen mantener t_m constante ($t_m = t$), se propone variar este parámetro, según los valores que va tomando $w1$, de la siguiente forma:

- Comenzar con un valor inicial para t_1;
- Cada vez que $w(\pi)$ mejore el valor del $w1$ obtenido hasta ese momento aumentar el valor de t_m;
- Si en un determinado número de iteraciones no hay mejora en $w1$ reducir el valor de t_m.

El objeto de este proceso es evitar caer en máximos locales, de los que muchas veces no se puede salir si se mantiene t constante, a la vez que se impide que el proceso emplee un tiempo excesivo de cálculo. El tiempo empleado por esta modificación es, en cualquier caso, mayor que el empleado por el procedimiento original, sin embargo se obtienen cotas inferiores realmente muy ajustadas al óptimo en cada vértice, con lo que el número de exploraciones se reduce enormemente al igual que el tiempo total de computación del algoritmo, como se verá mas adelante.

La segunda variación que se propone sigue la misma idea que la anterior, salvo que en este caso el valor inicial de t_1 no es constante sino que viene establecido como un parámetro de entrada para cada vértice, junto con X, Y, π, y $W_{X,Y}(\pi)$. El valor t' que van a *heredar* las siguientes ramas va a ser el valor de t_m en el momento que se alcanza dicho máximo.

El objeto de esta variante es *dar continuidad* a todo el proceso de acotación en las diferentes ramas de la misma forma que se hace con el vector de penalizaciones π. Los resultados obtenidos por estas dos variantes son parecidos y mejores que con el algoritmo original.

4.2.2.- RESULTADOS COMPUTACIONALES

A continuación se muestran los resultados conseguidos por estas variantes para la resolución de una serie de problemas correspondientes a la simulación de 20 matrices de distancias simétricas de dimensión 100. Estos resultados se comparan con los obtenidos por el algoritmo original de Held & Karp. Para cada uno de estos 20 problemas se recoge el tiempo máximo de computación, el valor de la cota inferior obtenida en el vértice inicial de los algoritmos examinados y el valor óptimo, (el valor de la cota inferior en el vértice inicial para ambas variantes es el mismo ya que parten del mismo valor inicial de t).

Problemas	**Tiempos de Computación (seg) en la ejecución del algoritmo completo**			**Cota Inferior Inicial**		**Valor Optimo**
	Variante 1	*Variante 2*	*H. & Karp*	*Variantes*	*H. & Karp*	
n.1	179.93	171.48	17403.18	174.00	153.00	175
n.2	360.31	238.49	15921.12	164.00	143.00	164
n.3	193.72	226.84	23486.20	167.00	139.00	167
n.4	352.79	261.44	9941.31	151.00	132.00	151
n.5	404.91	336.86	13940.43	162.00	139.00	162
n.6	98.75	106.17	11787.98	140.00	120.00	140
n.7	320.44	334.88	15017.05	162.00	137.00	162

Problemas	Tiempos de Computación (seg) en la ejecución del algoritmo completo			Cota Inferior Inicial		Valor Optimo
	Variante 1	*Variante 2*	*H. & Karp*	*Variantes*	*H. & Karp*	
n.8	120.39	128.09	9699.08	167.00	148.00	168
n.9	515.92	533.54	17921.08	181.00	158.00	181
n.10	459.84	408.75	19324.84	162.00	136.00	162
n.11	215.69	179.44	14061.21	201.00	181.00	201
n.12	138.35	165.27	22977.46	165.00	148.00	165
n.13	802.79	241.56	10542.13	162.00	143.00	163
n.14	264.80	280.23	24110.12	148.00	129.00	149
n.15	112.98	112.65	16981.18	173.00	150.00	173
n.16	204.93	248.81	22986.33	135.00	113.00	135
n.17	369.93	308.02	17098.55	163.00	143.00	163
n.18	587.37	621.92	23224.45	168.00	142.00	169
n.19	351.53	312.47	26248.71	168.00	141.00	168
n.20	298.96	791.97	18611.19	158.00	133.00	158

Estos resultados se resumen en el siguiente cuadro

		Variante 1	*Variante 2*	*Held & Karp*
Tiempo de Computación (seg.) (algoritmo completo)	**Media**	317.717	300.444	17564.18
	Mínimo	98.75	106.17	9699.08
	Máximo	802.79	791.97	26248.71
Cota Inferior Inicial		163.55		141.4
Desviacion del Optimo de la cota inferior (%)	**Media**	0.15		13.67
	Mínimo	0		9.95
	Máximo	0.67		16.76

4.2.3.- OBSERVACIONES

De los resultados anteriores se desprenden las siguientes observaciones:

- la obtención de una cota inferior ajustada al óptimo en cada vértice, especialmente en el inicial, es fundamental para la reducción del tiempo de computación, especialmente cuando el número de nodos es grande (la obtención de la cota inferior utiliza un tiempo polinomial, mientras que el número de ramas crece exponencialmente);
- las variantes propuestas aportan un método intuitivamente sencillo pero eficaz para la obtención de una buena cota inferior ajustada al óptimo ya que, al variar el parámetro t permite *salir* de óptimos locales en los que se puede caer en el proceso iterativo;
- las dos variantes emplean unos tiempos de computación muy parecidos en todos los casos, aunque la segunda variante parece ser ligeramente mejor.

Aunque los resultados son en conjunto satisfactorios, se pueden introducir algunas mejoras. Una de ellas es cambiar o combinar el heurístico utilizado para obtener la solución inicial. El heurístico utilizado es el *Rechazo Heurístico*/3-óptimo de Aragón y Pacheco, (1.992), que en pocas ocasiones da la solución óptima para problemas grandes. El algoritmo de Lin & Kerninghan o el método de Or son dos heurísticos de reconocida eficacia y que en muchas casos consiguen el óptimo. De esta forma, en estos casos, en el proceso iterativo para alcanzar la cota inferior inicial se puede *identificar* este óptimo, finalizando la ejecución del algoritmo en la primera rama. Así mismo también se podrían generar cotas superiores en cada uno de los vértices.

CAPITULO 2: CLASIFICACION Y DESCRIPCION DE LOS PRINCIPALES ALGORITMOS DE RUTAS

1.- INTRODUCCION

1.1.- FORMULACION DEL VRP

Un número n-1 de clientes en ciudades concretas con demandas conocidas de cierto artículo, han de ser atendidos desde una central (ciudad 1) por un número de vehículos m de capacidad conocida. Se trata de diseñar rutas de vehículos con distancia total a recorrer mínima.

Sean

d_{ij} : distancia de i a j, i,j = 1,..,n;

q_i : demanda de la ciudad i, i = 2,...n;

Q : máxima carga de un vehículo;

R : máximo radio de acción de un vehículo;

y las variables

$$x_{ijk} = \begin{cases} 1 \text{ si el vehículo k viaja de i a j,} \\ \\ 0 \text{ en caso contrario.} \end{cases}$$

El problema se formula como sigue:

$$\text{minimizar} \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot x_{ijk}$$

sujeto a:

$$\sum_{j=2}^{n} x_{1jk} = \sum_{i=2}^{n} x_{i1k} = 1, \quad \text{k=1,...,m;}$$

(cada vehículo debe visitar al menos una ciudad, y empezar y acabar en la central)

$$\sum_{k=1}^{m} \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ijk} = \sum_{k=1}^{m} \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{jik} = 1, \quad \text{i=2,...,n;}$$

(Todas las ciudades han de ser visitadas exactamente una vez, es decir, uno y solamente un vehículo puede entrar y salir de ella)

$$\sum_{k=1}^{m} \sum_{i \in S} \sum_{j \notin S} x_{ijk} > 1, \quad S \subset \{1,...,n\} \text{ y } S \neq \varnothing; \quad \text{(No se pueden formar subrutas)}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} q_i \cdot x_{ijk} \leq Q, \quad \text{k=1,...,m;} \quad \text{(Limitaciones de capacidad)}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} d_{ij} \cdot x_{ijk} \leq R, \quad \text{k=1,...,m;} \quad \text{(Restricciones en el radio de acción de cada vehículo)}$$

la última restricción es opcional y algunos autores no la consideran.

1.2.- UN ESQUEMA DE CLASIFICACION

La mayor parte de los algoritmos para los problemas de rutas pueden ser clasificados en alguno de los siguientes grupos:

1.-Algoritmos Heurísticos. Que a su vez se pueden subdividir en alguno de los siguientes grupos (en la denominación de estos grupos se ha optado por respetar la terminología que habitualmente se utiliza)

- 1.- *Cluster 1° - Ruta 2°*
- 2.- *Ruta 1°. Cluster 2°*
- 3.- *Ahorros e Inserción*
- 4.- *Mejora e Intercambio*
- 5.- *Programación Matemática*

Estos grupos no son excluyentes entre sí; por ejemplo, algunos algoritmos basados en métodos de Programación Matemática pueden englobarse en el grupo de Cluster 1°-Ruta 2°. Existen estrategias de solución comunes a varios algoritmos. Un análisis comparativo de estos algoritmos se puede encontrar en los trabajos de Ball y Magazine (1.981), Fisher, (1.980), y Fisher y Rinnoy Kan, (1.988).

2.-Algoritmos Exactos: En este grupo destacan por una parte los algoritmos basados en métodos de relajación de la Programación Dinámica, (Christofides, Mingozzi y Toth, (1.981)), y por otra parte las adaptaciones de los algoritmos exactos para el TSP, como el de Little, (Christofides y Eilon, (1.969)).

Esquemas de clasificación parecidos a éste se pueden encontrar en los trabajos de Bodin y Golden, (1.981) y (1.983).

2.- ALGORITMOS HEURISTICOS

2.1.- 'CLUSTER 1º - RUTA 2º'

El algoritmo más *representativo* de esta familia de algoritmos es el del *barrido* o *sweep*, debido a Gillet & Miller, (1.974), y Gillet & Johnson, (1.976). A continuación se describen someramente los principales características de estas técnicas.

2.1.1.- ALGORITMOS DE BARRIDO (SWEEP) EN PROBLEMAS CON UN DEPOSITO

El algoritmo *sweep* básicamente consiste en dos procesos o fases: En la primera se dividen las localizaciones de los clientes en grupos o *clusters*, según su distribución geográfica; en la segunda se resuelve el problema del TSP para cada uno de esos clusters y el depósito.

2.1.1.1.- Algoritmo

Se define

An(i) : Coordenada polar angular de i, que se define como

$$An(i) = \arctan[(Y(i)-Y(1))/(X(i)-X(1))]$$

donde $-\pi < An(i) < 0$ si $Y(i) - Y(1) < 0$

y $0 \leq An(i) \leq \pi$ si $Y(i) - Y(1) \geq 0$, $(i = 2,3,...,n)$,

R(i) : Coordenada polar radio de la i-ésima localización. $(i = 2,3,...,n)$.

Sin pérdida de generalidad, se suponen las localizaciones reordenadas de tal forma que $An(i) < An(i+1)$, y en el caso de que $An(i) = An(j)$ entonces $i < j$ si $R(i) < R(j)$.

El algoritmo se puede dividir en dos partes: el *forward sweep* o *barrido 'hacia adelante'*, y el *backward sweep* o *barrido 'hacia atrás'*. A continuación se describe el *forward sweep*.

Paso 1.- (Agrupar los clientes en Rutas)

Paso 1.1.: Poner k = 2, nr = 1.

Paso 1.2.: Si no se violan las restricciones del vehículo añadir el cliente k a la ruta nr; en caso contrario hacer nr = nr+1, y añadir k a la ruta nr.

Poner k = k+1.

Si k < n repetir el paso 1.2.

Paso 1.3.: Resolver el TSP para cada uno de los grupos formados.

Paso 2.- (Modificaciones en las Rutas iniciales)

Para k = 1,..., nr-1 hacer:

Eliminar el cliente i de la ruta k que minimice la función

$$f = R(i) + An(i) \cdot AVR$$

(siendo AVR la media del radio de todas las localizaciones).

Sea p el cliente de la ruta k+1 más cercano al último cliente que se añadió a la ruta k en el paso 1.2.

Mientras haya mejora y no se violen las restricciones, hacer:

Añadir p a la ruta k;

Poner en p el índice del cliente de la ruta k+1 más cercano al último cliente añadido a k.

Paso 3.- (Rotación)

Repetir:

Rotar los ejes en sentido opuesto al de las agujas del reloj, de tal forma que el primer cliente pase a ser el último, el segundo el primero, ...

Volver al paso 1;

hasta la ordenación original.

Como solución del *forward sweep* se toma la mejor de las obtenidas en cada una de las iteraciones o rotaciones.

El *backward sweep* es semejante al anterior; en este caso el proceso de selección y agrupación de clientes empieza por el último; es decir, la ruta 1 la formarán los clientes n, n-1, n-2... Normalmente un algoritmo *sweep* consta de *forward* y *backward*, siendo la solución del algoritmo *sweep* la mejor de las soluciones obtenidas por cada una de ambas partes

2.1.2.- ALGORITMOS DE BARRIDO EN PROBLEMAS CON MULTIPLES DEPOSITOS

Gillet & Johnson (1.976), adaptan el algoritmo *sweep* a problemas de rutas con múltiples depósitos. Si M es el número de estos depósitos o terminales, el algoritmo reduce el problema original a una colección de M problemas con un depósito, y aplica a cada uno de éstos el algoritmo anteriormente expuesto.

Un primer paso consiste en la formación de *clusters* o conjuntos de puntos demanda asignados a un depósito. El procedimiento para particionar o agrupar se basa en criterios *espaciales o geográficos*. En un segundo paso cada terminal *atrae* a puntos de los clusters adyacentes. Estos puntos son *candidatos* a ser *arrastrados* hacia el cluster del terminal que los atrae.

Paso 1.- (Asignación de puntos demanda a clusters)

Mientras no se hayan asignado todos los clientes, hacer:

Seleccionar un cliente j no asignado.

Para cada depósito k, encontrar los clientes i_k y l_k de k entre los que se situaría j.

Determinar k* verificando $\min_{k=1,\ldots,M} \{d_{i_k j} + d_{j l_k} - d_{i_k l_k}\} = d_{i_{k^*} j} + d_{j l_{k^*}} - d_{i_{k^*} l_{k^*}}$.

Asignar j a la terminal k*.

Resolver el VRP correspondiente a cada terminal con el algoritmo del *barrido*.

Paso 2.- (Mejora: Reasignación de puntos demanda)

Para cada terminal k:

Ordenar los nodos asignados a las terminales adyacentes según el ángulo que forman con la terminal k.

Para cada uno de estos nodos:

Considérese no asignados él y sus adyacentes en su terminal.

Reasignarlos como en el paso 1.

Si hay cambio de terminal:

Resolver el VRP en cada depósito afectado

Si hay mejora registrarla.

En el primer paso los criterios de elección del cliente que va a ser asignado son variados, y no hay ninguno que se aconseje especialmente, (Gillet y Johnson, (1.976)). En el segundo paso, cuando se produce mejora, se aconseja eliminar el punto reasignado al nuevo cluster de la lista para posibles cambios posteriores. El propósito es ahorrar operaciones que probablemente no vayan a dar mejora o la den pequeña.

Otros trabajos más recientes en los que se pueden encontrar algoritmos de este tipo son los de Chapleau y otros, (1.985), y Chung y Norback, (1.991).

2.2.- 'RUTA 1º - CLUSTERS 2º'

Dentro de este grupo de algoritmos, destacan principalmente aquellos destinados a resolver el problema del Cartero Chino (CCP) para varios vehículos. Entre ellos destacan los de Bodin & Kursh, (1.978) y (1.979), para redes dirigidas y de Stern & Dror, (1.979), semejante al anterior para redes no dirigidas. En el trabajo de Beasley, (1.983), puede verse una aplicación de estos algoritmos para el VRP clásico.

2.2.1.- DESCRIPCION DE UN ALGORITMO RUTA 1º - CLUSTER 2º PARA PROBLEMAS DE LIMPIADORES DE CALLES

El siguiente algoritmo, debido a Bodin & Kursh, (1.978) y (1.979), resuelve el problema del CPP en redes dirigidas con varios vehículos. Considerese la red G = {N,A} no necesariamente conexa. La red G se suele extraer de la red más general de calles F = (M,B). A veces es imposible formar una ruta con los arcos de G solamente; hace falta añadir algún arco más de F, además de los que necesitan ser barridos. Estos arcos *improductivos* conectan los arcos *productivos* y se desea minimizar el tiempo que se emplea en recorrerlos.

Teorema.-
En una red dirigida conexa G = (N,A) existe ruta euleriana si y sólo si $d^+(v) = d^-(v)$, $\forall\, v \in N$, siendo $d^+(v)$, $d^-(v)$ el semigrado exterior e interior, respectivamente del vértice v.

2.2.1.1.- Algoritmo

El algoritmo inicialmente añade arcos a la red G para obtener una red balanceada ($d^+(v) = d^-(v)$, $\forall\, v \in N$). A continuación se forma una ruta euleriana en cada una de las componentes de esta nueva red. En el siguiente paso se obtiene una red conexa para dar lugar a una ruta mayor y finalmente se particiona ésta en diferentes rutas según las restricciones de tiempo máximo de conducción u otras que pudiera haber.

Paso 0.- (Obtención de una red balanceada H')

Resolver el siguiente problema de transporte:

Sean

$S = \{v \in N / d^+(v) > d^-(v)\}$;
$D = \{v \in N / d^-(v) > d^+(v)\}$;
$\forall j \in S \quad s_j = d^+(j) - d^-(j)$;
$\forall k \in D \; d_k = d^-(k) - d^+(k)$;

$$\min \quad z = \Sigma_{j \in S} \Sigma_{k \in D} c_{jk} x_{jk}$$

sujeto a:

$$\Sigma_{k \in D} x_{jk} = s_j \quad j \in S,$$
$$\Sigma_{j \in S} x_{jk} = d_k \quad k \in D,$$
$$x_{jk} \geq 0.$$

siendo c_{jk} la distancia mínima entre los nodos j y k, a través de la red F.

Para todo (j,k)/ $x_{jk} > 0$:

sea la sucesión de arcos (j,p), (p,q),... , (t,k) que define el camino más corto de j a k , a través de F. Añadir estos arcos un número x_{jk} de veces.

Poner H = (N', A') la red G con los arcos añadidos.

Paso 1.- (Obtención de subciclos)

Cada nodo de N' se *examina* de la forma siguiente:

Para cada arco u_i que *llega* a este nodo, o *predecesor*, y cada arco u_j que *sale* de él, o *sucesor*, se define d_{ij} el coste de pasar del arco u_i al u_j a través de este nodo.

Resolver el problema de asignación correspondiente en cada nodo, dando lugar a pares de arcos *predecesor-sucesor*.

Comenzando en un nodo arbitrario, y siguiendo los pares predecesor-sucesor, obtener uno o varios ciclos que recorren todos los arcos de H.

Paso 2.- (Eliminación o fusión de *subciclos*)

En cada componente repetir hasta obtener solo un ciclo:

Elegir dos subciclos que tengan un vértice común v. Formar un subciclo mayor insertando uno de esos subciclos en el lugar de v en el otro subciclo.

Paso 3.- (Fusionar las rutas de cada componente)

Si H no es conexa definir la red L = (N*,A*), completa como sigue:

Poner N* ={1,2,...,nc} donde nc es el número de componentes de H;

Poner como longitud del arco que une los nodos k y p de L la distancia de las componentes p y k de H.

Sean n_p y n_q los nodos que definen la distancia entre las componentes p y q de N*.

Hallar el árbol de expansión de mínimo peso de la red L.

Para cada arco (k,p) de este árbol, añadir a H los arcos correspondientes al camino mínimo entre n_k y n_p y entre n_p y n_k.

Sea E = (N",A") la red H con los arcos añadidos.

Formar una ruta Euleriana en ésta repitiendo los pasos 1 y 2 para la red E.

Paso 4.: (Obtención de subrutas)

Elegir un punto de N" como el comienzo de la ruta.

A partir de este punto particionar la ruta obtenida en diferentes rutas respetando las restricciones del problema.

2.2.1.2.- Ejemplo Gráfico

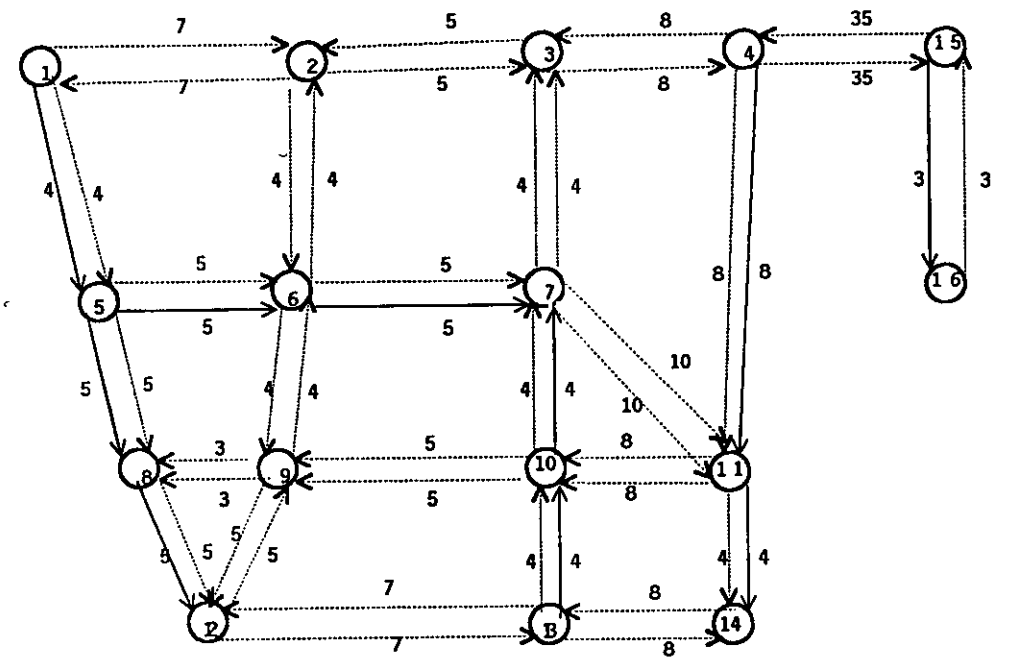


Figura 9. Plano de las calles de un municipio ficticio. Las líneas continuas indican las lados de las calles con prohibición de aparcar

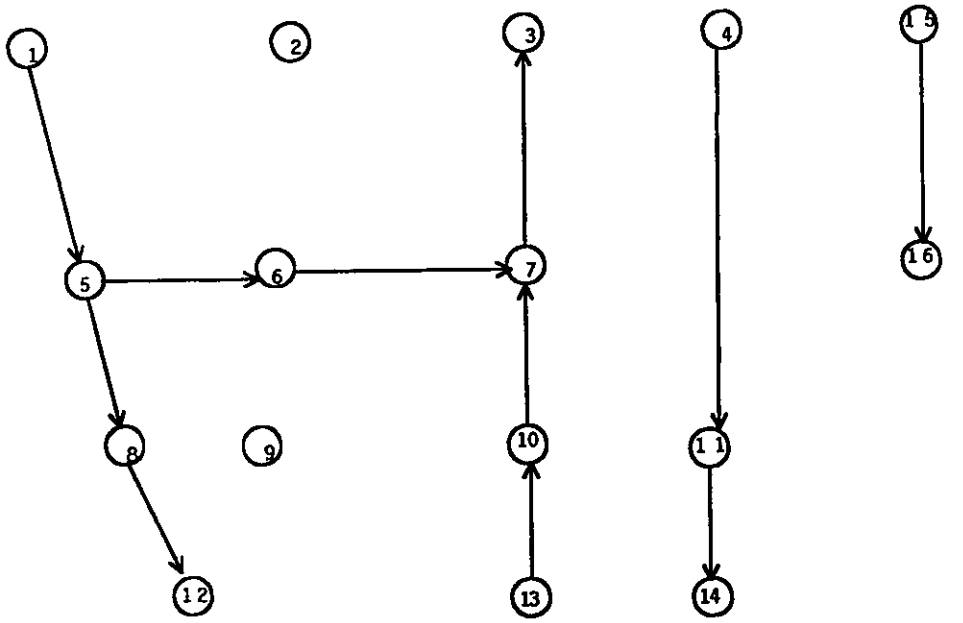


Figura 10. Arcos que deben ser limpiados

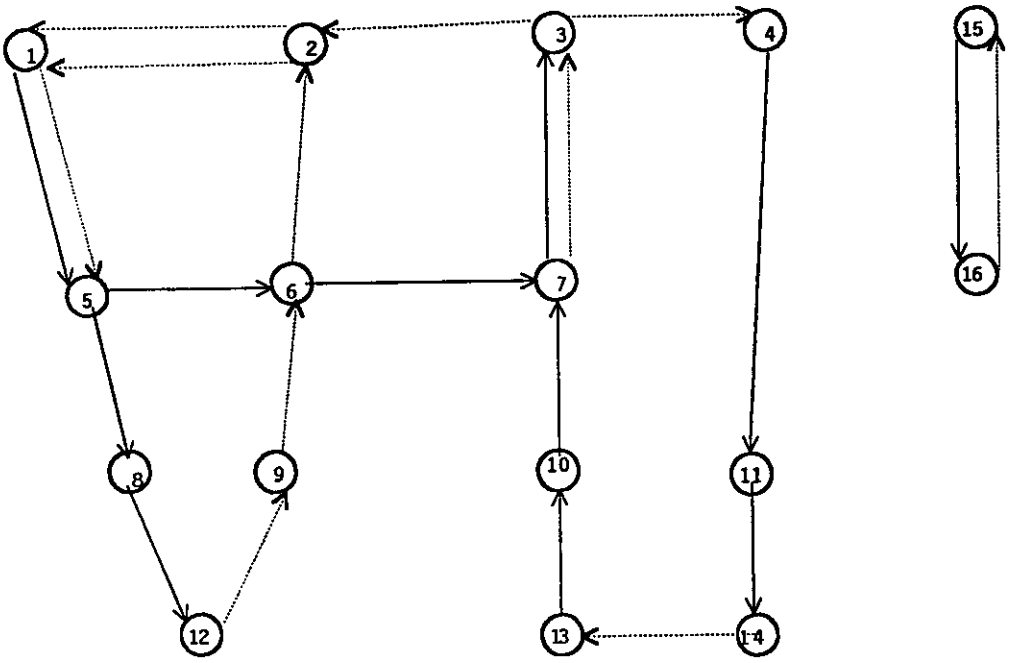


Figura 11. Con líneas punteadas se indican los arcos que se añaden para obtener una red balanceada (Paso 0).

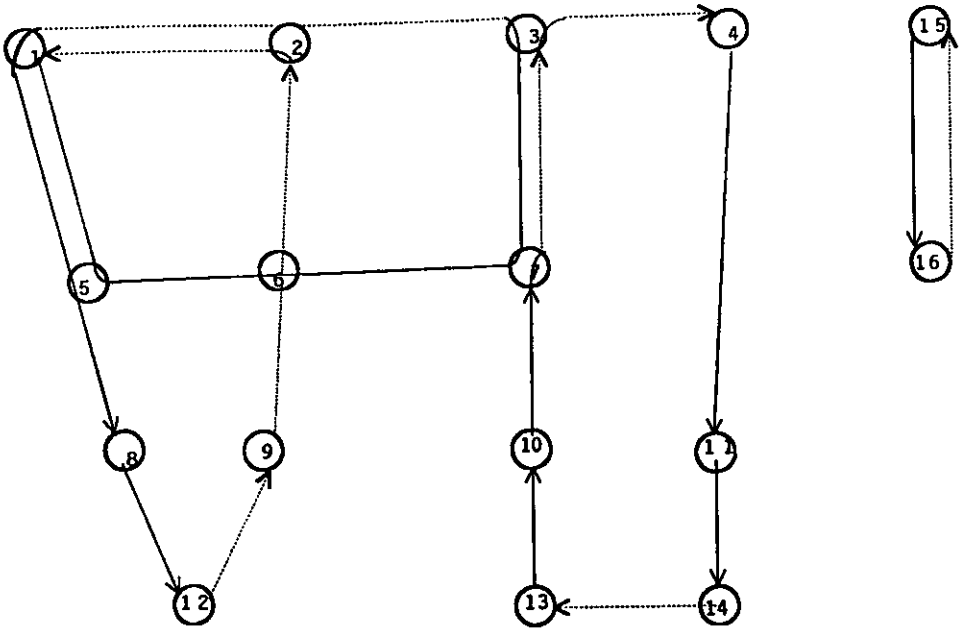


Figura 12. En cada nodo se unen los diferentes pares predecesor-sucesor, formándose subciclos (Paso 1).

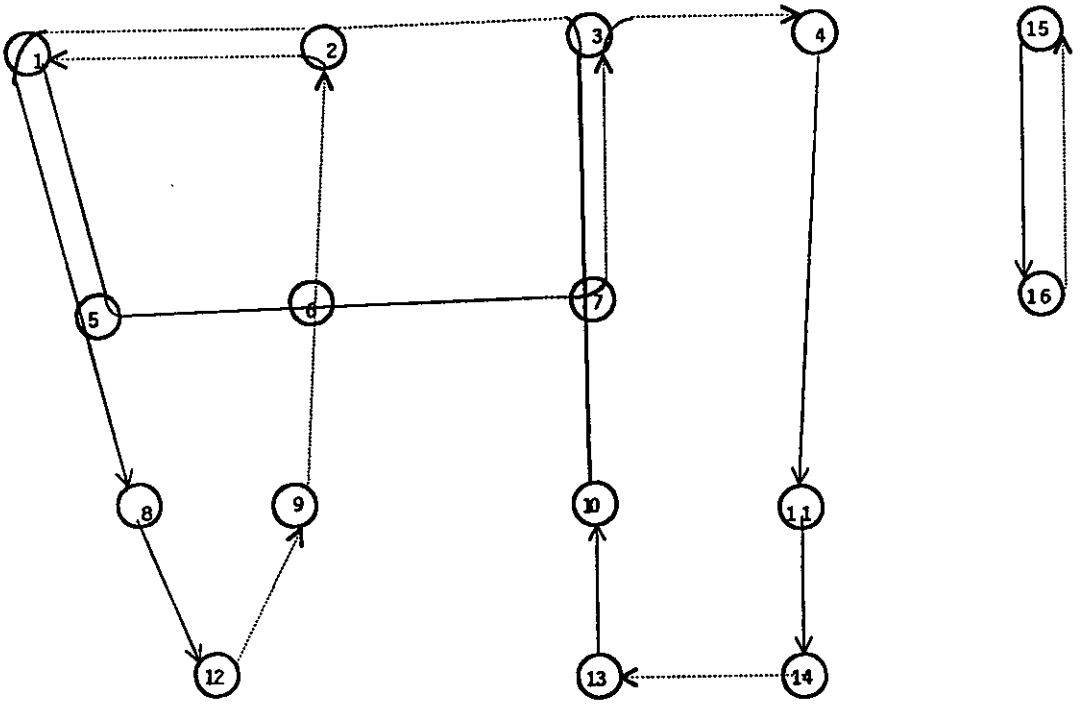


Figura 13. En cada componente de la red se unen los diferentes subciclos para formar un ciclo mayor (Paso 2).

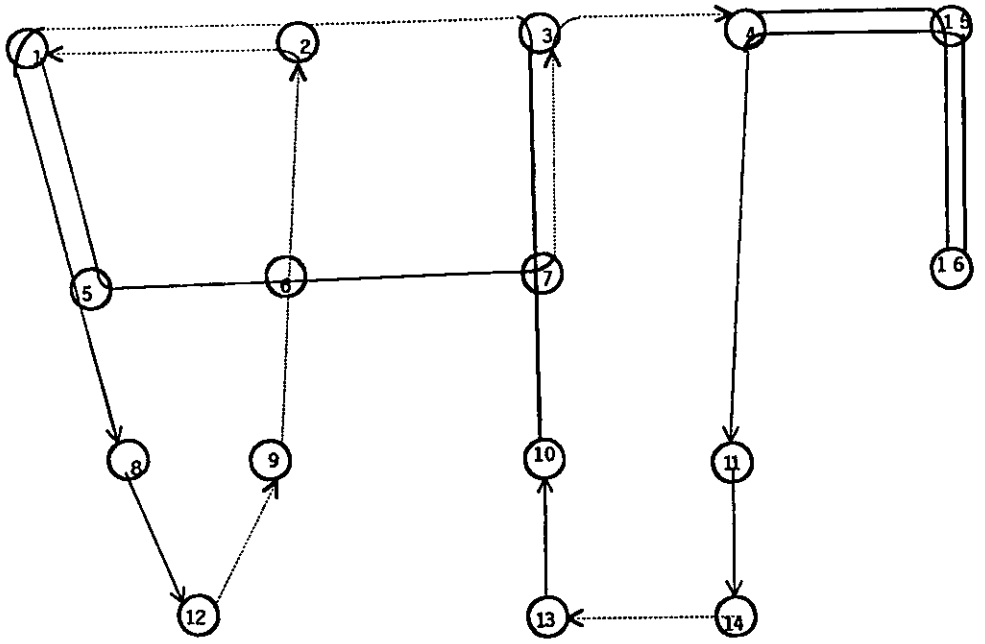


Figura 14. Se unen los ciclos formados en cada una de las componentes, dando lugar a la gran ruta final (Paso 3).

2.2.2.- METODOS 'RUTA1ª-CLUSTER 2º' PARA EL VRP

Beasley, (1.983), propone una serie de modificaciones en las técnicas de este tipo aplicadas al VRP. Las modificaciones propuestas se basan en el siguiente Método Básico:

Paso 1.- (Gran Ruta Inicial)

Formar una *gran ruta*, resolviendo el problema del TSP con todos los clientes que hay que visitar y el depósito. (Se supone, sin pérdida de generalidad, que el primer cliente visitado tiene asignado el índice 1, el segundo el 2, ... y así sucesivamente).

Paso 2.: (Partición Optima)

Definir la matriz $\mathbf{c} = (c_{ij})$ de la forma siguiente:

$$c_{ij} = \begin{cases} d_{1,i+1} + \sum_{k=i+1}^{j-1} d_{k,k+1} + d_{j,1}, & \text{si } i < j \text{ y la ruta } 1 - i+1 - i+2 - \ldots - j - 1 \text{ es factible} \\ \infty, & \text{en caso contrario.} \end{cases}$$

Definir la red completa cuyos nodos van a representar los clientes, y donde la longitud del arco (i,j) es c_{ij}, esto es, el coste de la ruta 1 - i+1 - i+2 - ... - j - 1.

Resolver el problema del camino más corto del nodo 1 hasta el n en esta red.

Registrar cada una de las rutas asociadas a cada arco (i,j) del camino mínimo obtenido.

2.2.2.1.- Extensiones y Mejoras

Algunas de las sugerencias propuestas para mejorar la eficacia de esta técnica son:

- Los parámetros c_{ij} se pueden definir como el *coste total* de realizar la ruta 1 - i+1 - i+2 - ... - j - 1, incluyendo costes de combustibles, costes del vehículo, del conductor, etc,...; incluso se pueden incorporar las preferencias de los clientes por un determinado tipo de vehículo o conductor, preferencias por el tipo de recorrido,..
- Una vez obtenida cada parte i+1, i+2, ..., j, se puede intentar hacer mejoras a la correspondiente ruta con algoritmos para el TSP.
- Se puede excluir el depósito de la gran ruta inicial para dar más *flexibilidad* al proceso de partición: cualquier cliente puede ser el origen y final. Se requeriría un algoritmo que determinara el camino mínimo entre cada par de puntos en una red, (como el de Floyd, (1.964), que usa $\theta(n^3)$ operaciones).
- Una vez obtenido el itinerario de cada ruta, éste se puede *chequear* y cambiar, si así lo requiere la incorporación de restricciones como *ventanas de tiempo* u otras.

$$s_{ij} = t_{1j}^{k} + t_{1i}^{k} - t_{ij}^{k},$$

si la unión se produce dentro de un mismo periodo k; de esta forma se asegura la simetría de S.

En cuanto al segundo punto, si quedan m horas del periodo k cuando un vehículo comienza el recorrido desde el punto i al j, y $t_{ij}^{k} > m$, entonces el tiempo de viaje entre i y j va a ser

$$m + (1-(m/t_{ij}^{k}))\cdot t_{ij}^{k}.$$

Esta técnica se ha aplicado especialmente a problemas de rutas en empresas de servicios urbanos, donde las condiciones de tráfico hacen cambiar la duración de las travesías.

2.4.- INTERCAMBIO Y MEJORA

Estas técnicas son adaptaciones al VRP de los algoritmos de intercambios o r-óptimos descritos por Lin (1.965), y Lin & Kernighan; (1.973), para el TSP. Ejemplos de estas últimas son las propuestas por Christofides & Eilon (1.969), y Russell (1.977). Estas adaptaciones se usan solo para matrices simétricas.

2.4.1.- ADAPTACION DE CHRISTOFIDES Y EILON DE LOS ALGORITMOS DE INTERCAMBIO

A continuación se describe como se pueden adaptar los algoritmos de intercambio al VRP reformulando este como el TSP.

Si hay N vehículos se incorporan N-1 depósitos artificiales, todos en la misma posición con el depósito inicial, y con las mismas distancias a los otros nodos. Se considera que la distancia entre

dos depósitos es infinita. De esta forma se plantea el TSP donde hay que visitar a los clientes originales y a los depósitos artificiales partiendo del depósito original y volviendo a él. Los clientes visitados entre dos llegadas sucesivas a los depósitos definen la ruta de un vehículo. El haber definido infinito la distancia entre los depósitos asegura que en la solución final no se visitan dos depósitos sin visitar otros clientes entre ambas llegadas a los depósitos. En el proceso de formación de esta gran ruta, se han de incorporar el chequeo de las restricciones de capacidad máxima y distancia del problema original.

El algoritmo propuesto se desarrolla básicamente de la forma siguiente:

(1) Empezar con un recorrido aleatorio inicial;
(2) Encontrar un recorrido 2-óptimo, que sirva de inicio al paso (3);
(3) Encontrar un recorrido 3-óptimo; volver al paso (2);

repetir los pasos (2) y (3) hasta que no haya mejora. En el proceso de intercambios se ha de verificar la factibilidad del problema original.

2.4.2.- ADAPTACION DE RUSSELL

En este apartado se describe una adaptación del algoritmo de Lin & Kernighan para el VRP, llamado *N-TOUR*. Se comienza reformulando el VRP como un TSP, incorporando N-1 nodos (depósitos) artificiales.

Más concretamente, sea S el conjunto de aristas en la definición de este TSP; en el algoritmo de Lin & Kerninghan se busca un subconjunto $T \subset S$, que de lugar a N rutas y respetando las restricciones minimice la longitud total.

Para ello en cada paso, si el conjunto de aristas T define un conjunto de N rutas factibles, el algoritmo identifica una arista y_i en S-T, y la reemplaza por alguna x_i de T. El proceso de

2.3.- AHORROS E INSERCION

Estas técnicas se basan en el algoritmo desarrollado por Clarke & Wright, (1.964), y que ha sido durante mucho tiempo, uno de los más utilizados y adaptados por otros autores. Así, hay que destacar la variante descrita por Eilon, (1.971), la adaptación de esta variante para problemas con matriz de tiempos variable realizada por Beasley, (1.983), y la variante descrita por Christofides & Eilon, (1.969).

2.3.1.- ALGORITMO DE AHORROS DE CLARKE Y WRIGHT

El algoritmo empieza con una solución inicial consistente en un conjunto de rutas, cada una de las cuales parte del nodo inicial, visita un cliente concreto y regresa al nodo inicial. En cada uno de los pasos siguientes se unen en una misma ruta dos nodos de dos rutas diferentes según los ahorros que se obtengan. Básicamente consiste en los pasos siguientes:

Paso 0.- (Rutas Iniciales)

Para i = 2,..., n: Formar una ruta que parte del nodo inicial visita el nodo i y regresa.

Paso 1.- (Cálculo de ahorros)

Para cada par de nodos i, j respectivamente principio y final de dos de las rutas formadas en ese momento poner $s_{ij} = d_{1i}+d_{j1}- d_{ij}$.

Paso 2.- (Elección de todos los ahorros posibles)

Ordenar los s_{ij} de mayor a menor.

Poner k = 1.

Repetir:

Si la unión correspondiente al ahorro k-ésimo produce solución factible, registrarla, si no rechazarla;

Poner k = k+1.

hasta que la lista esté agotada o no queden ahorros positivos.

Paso 3.: Si en el paso 2 se ha realizado alguna unión ir al paso 1; en otro caso parar.

2.3.2.- VARIANTE DE EILON Y OTROS

La variante desarrollada por Eilon y otros, (1.971), consiste en que una vez formadas las rutas iniciales según el paso 0 del algoritmo 2.3.1. se procede de la siguiente manera:

Paso 1.: Calcular $s_{ij} = d_{1i} + d_{j1} - d_{ij}$ para cada par de puntos i, j respectivamente principio y final de dos de las rutas formadas en ese momento

Paso 2.: Ordenar los s_{ij} de mayor a menor.

Comenzando por el s_{ij} mayor:

1. Si la unión correspondiente a este ahorro produce solución factible, registrarla, si no rechazarla;
2. Escoger el siguiente ahorro de la lista y repetir (1), hasta que la lista esté agotada.

Paso 3.: Con todas las uniones registradas formar las correspondientes rutas de forma simultánea.

Variantes:

(1) *Construcción Múltiple*: tal como se ha descrito en el paso 3. Se va formando simultáneamente un determinado número de rutas; no se garantiza que la solución sea factible.

(2) *Construcción Secuencial*: En el paso 2 solo se registra una unión factible, con lo que hace falta realizar varias iteraciones a todo el proceso para vaciar la lista. Por contra se asegura la factibilidad.

2.3.3.- ADAPTACION DEL ALGORITMO DE EILON A PROBLEMAS CON DISTANCIAS QUE DEPENDEN DEL TIEMPO

Beasley, (1.981), describe una adaptación del algoritmo de Eilon para el VRP simétrico en el que hay que minimizar el tiempo total en ruta del conjunto de los vehículos utilizados, y la matriz de tiempos entre los diferentes nodos depende del periodo del día en el que se realizan los diferentes desplazamientos.

Sean:

K : número de períodos disjuntos en los que se divide el día;

t_{ij}^{k} : tiempo de desplazamiento entre los nodos i y j en el periodo k, k=1,...,K;

(T_{k-1}, T_k) período k-ésimo del día, dentro del cual no cambia la matriz de tiempos.

En el algoritmo de Eilon se realizan los siguientes ajustes:

- 1. Como definir los ahorros s_{ij} en función de t_{ij}^{k}.
- 2. Determinar una forma para calcular el tiempo de viaje entre dos clientes cuando el recorrido de algún tramo tiene lugar en dos o más períodos de tiempo.
- 3. Chequear la factibilidad de las rutas teniendo en cuenta que el tiempo total de viaje puede ser distinto según en que sentido se realice: por ejemplo los tiempos de recorridos de las rutas 1 - i - j - 1, y 1 - j - i - 1 no tienen por qué ser iguales.

Para definir s_{ij} de tal forma que $S=(s_{ij})$ sea simétrica se toma

$$s_{ij} = \max \{t_{1j}^{k} + t_{1i}^{k+1} - t_{ij}^{k+1}, t_{1i}^{k} + t_{1j}^{k+1} - t_{ij}^{k+1}\}$$

si la unión va a tener lugar en dos periodos de tiempo diferentes, o

intercambio continúa mientras haya mejora: Si $|x_i|$ es la longitud de la arista x_i , entonces k aristas x_i son seleccionadas para ser reemplazadas por k aristas y_i si

$$G_k = \sum_{i=1}^{k} \left(|x_i| - |y_i| \right) > 0.$$

En esta adaptación en cada iteración se consideran varios niveles o estados k, k=1,2,3... . Cada nivel se refiere al número de aristas que van a ser intercambiadas simultáneamente. En cada nivel se examina y se procesa una lista de grupos de k aristas de T para posibles intercambios. Los grupos que al intercambiarse den ganancias pasan al siguiente nivel, incorporando una arista nueva. De esta forma el proceso queda como:

nivel 2 $\rightarrow$ nivel 3 $\rightarrow$ .... $\rightarrow$ hasta llegar a un último nivel K cuando ya no haya mejora;

los K intercambios se examinan, y se realiza aquel con mayor ganancia $G^* = G_K$ que sea factible.

Se repite el proceso hasta que no haya mejora. La factibilidad se verifica sólo en el último nivel.

2.5.- PROGRAMACION MATEMATICA

En este grupo se incluye una gran colección de estrategias de solución que usan o bien los modelos o bien las técnicas de programación matemática como método principal. En el primer caso se encuentran las *aproximaciones* o modelizaciones de los problemas de rutas como problemas de programación matemática; por ejemplo el tratamiento del VRP como un problema de Asignación Lineal Generalizada, debido a Fisher & Jaikumar, (1.981), o como un problema de Partición de Conjuntos, como el descrito por Cullen y otros, (1.981), o Manganti, (1.981).

En el segundo caso se incluyen aquellos que usan técnicas propias de la programación matemática, como los métodos de Relajación Lagrangiana propuestos por Stewart (1.984).

2.5.1.- ALGORITMO DE FISHER Y JAIKUMAR

El algoritmo de Fisher y Jaikumar, (1.981), para el VRP además de minimizar el coste de todas las rutas determina también el número mínimo de vehículos que deben ser usados.

2.5.1.1.- Formulación del VRP en términos de GAP

En primer lugar, se redefine el VRP como un problema de Asignación no Lineal Generalizada. Para ello se definen las variables

$$y_{ik} = \begin{cases} 1, \text{ si el cliente i es visitado por el vehículo k} \\ \\ 0. \text{ en caso contrario;} \end{cases}$$

$$x_{ijk} = \begin{cases} 1, \text{ si el vehículo k va del cliente i al j;} \\ \\ 0, \text{ en caso contrario;} \end{cases}$$

la formulación que se propone es la siguiente:

$$\min \sum_{k=1}^{m} f(y_k) \qquad (1')$$

sujeto a:

$$\sum_{i=1}^{n} q_i \cdot y_{ik} \leq Q, \qquad k=1,..,m; \qquad (2)$$

$$\sum_{k=1}^{m} y_{ik} = \begin{cases} m, & \text{si } i=1, \\ & \\ 1, & \text{si } i=2,\ldots,n; \end{cases} \qquad (3)$$

$$y_{ik} \in \{0,1\}, \quad i=1,\ldots,n;\ k=1,\ldots,m;$$

donde $f(y_k)$ es el coste de la la solución del TSP para los clientes asignados al vehículo k, es decir de $N(y_k) = \{i / y_{ik} = 1\}$. La función $f(y_k)$ puede ser definida matemáticamente como:

$$f(y_k) = \min \sum_{i,j,k} c_{ij} \cdot x_{ijk} \qquad (1'')$$

sujeto a:

$$\sum_{i=1}^{n} x_{ijk} = y_{jk}, \qquad j=1,\ldots,n; \qquad (5')$$

$$\sum_{j=1}^{n} x_{ijk} = y_{ik}, \qquad i=1,\ldots,n; \qquad (6')$$

$$\sum_{i,j \in S \times S} x_{ijk} \leq |S| - 1; \qquad S \subset \{1,..,n\}, \quad 2 \leq |S| \leq n \qquad (7')$$

$$x_{ijk} \in \{0,1\}, \qquad i,j = 1,\ldots,n. \qquad (8')$$

Obviamente $f(y_k)$ no puede determinarse, en general, de forma explícita.

2.5.1.2.- Aproximación Lineal

El algoritmo se basa en la construcción de una aproximación lineal de $f(y_k)$ reemplazando (1') por

$$\sum_{k=1}^{m}\sum_{i=1}^{n} e_{ik}\cdot y_{ik}.$$

De esta forma, se tiene un modelo de asignación lineal generalizado, y su solución define una serie de asignaciones factibles de clientes a vehículos. En cada una de estas asignaciones, se resuelve el TSP correspondiente.

Para construir una aproximación lineal de $f(y_k)$ se definen un conjunto de *clientes semilla* i_1, $i_2,...,i_m$, que se asignan a los vehículos 1, 2,...,m respectivamente. El coeficiente e_{ik} se define como el coste de insertar el cliente i en la ruta que va desde el origen 1 al cliente semilla i_k y vuelve. Más concretamente:

$$e_{ik} = \min\{d_{1,i} + d_{i,i_k} + d_{i_k,1}, d_{1,i_k} + d_{i_k,i} + d_{i,1}\} - \{d_{1,i_k} + d_{i_k,1}\}.$$

2.5.2.- FORMULACION COMO PROBLEMAS DE PARTICION DE CONJUNTOS

Manganti, (1.981), expone que una de las estrategias de solución más exitosa se deriva de la formulación del VRP como un problema de Particionamiento de Conjuntos. En principio se requiere la enumeración de todos los *clusters* factibles posibles, es decir, de todas las posibles asignaciones factibles de puntos-demanda a rutas de vehículos.

Sean:

$$z_j = \begin{cases} 1, \text{ si se usa el cluster j} \\ 0, \text{ en caso contrario;} \quad j=1,...,J; \end{cases}$$

$$a_{ij} = \begin{cases} 1, \text{ si el punto demanda i está asignado al cluster j;} \\ 0, \text{ en caso contrario;} \quad i=1,...,n; \quad j=1,...,J; \end{cases}$$

siendo n el número de clientes, y J el número de clusters posibles. La formulación del VRP como problema de Partición de Conjuntos es:

$$\min \sum_{j=1}^{J} \hat{c}_j \cdot z_j$$

sujeto a:

$$\sum_{j=1}^{J} a_{ij} \cdot z_j = 1, \qquad i=1,2,...,n;$$

$$z_j \in \{0,1\}, \qquad j=1,2,...,J.$$

Las restricciones indican que cada punto demanda i debe asignarse exactamente a uno de los posibles clusters j, j=1,...,J, Los coeficientes de la función objetivo $\hat{c}_j$ son el coste de la solución óptima del TSP correspondiente a visitar a los clientes del cluster i desde el depósito y volver a él.

Como el número de clusters posibles es enorme, se proponen criterios de reducción en el conjunto de clusters, que den lugar a algoritmos heurísticos efectivos; por ejemplo criterios espaciales o geográficos.

2.5.3.- RELAJACION LAGRANGIANA HEURISTICA DE STEWART

La relajación de Stewart, (1.984), consiste en *relajar* en la formulación del VRP las restricciones de capacidad, y aplicar un proceso de intercambio r-óptimo a cada una de las rutas que se obtiene de la solución del m-TSP asociado.

Sea:

$$x_{ijk} = \begin{cases} 1, \text{ si el arco (i,j) está en la ruta k-ésima,} \\ 0, \text{ en caso contrario;} \end{cases}$$

S_m = Conjunto de soluciones factibles del m-TSP:

se propone la siguiente formulación del VRP:

$$\min \quad \sum_{k=1}^{m} \sum_{i,j=1}^{n} d_{ij} x_{ijk} + \lambda \cdot \sum_{k \in K^*} \left(\sum_{i,j=1}^{n} q_i x_{ijk} - Q \right), \quad (1)$$

sujeto a:

$$\sum_{i,j=1}^{n} q_i x_{ijk} - \sum_{i,j=1}^{n} q_i x_{ijl} \geq 0, \quad l=k+1; \quad k=1,...,m-1; \quad (2)$$

$$\mathbf{x} = (x_{ijk}) \in S_m; \quad (3)$$

donde $\lambda \geq 0$ y $K^* = \{k \,/\, \Sigma_{i,j=1}^{n} q_j \cdot x_{ijk} > Q\}$.

Las restricciones (2) obligan a que la ruta con mayor demanda sea la ruta 1, la siguiente la 2, etc. λ es un factor de penalización que se añade en la función objetivo cuando se viola las restricciones de capacidad. El algoritmo de Stewart considera las restricciones de capacidad sólo *indirectamente* en la función objetivo. Por ello tiene más flexibilidad ya que soluciones intermedias, que pueden ser infactibles, se admiten siempre que el coste total mejore de una iteración a la siguiente. Se basa en un proceso de intercambios 3-óptimo para resolver el problema relajado (1) - (3) en el que se parte de un valor pequeño de λ que va aumentando hasta obtener una solución factible.

En la práctica, la solución para el m-TSP pura, ($\lambda = 0$), está bastante lejos de ser factible para el VRP. La demanda de la ruta con mayor demanda para el m-TSP, suele ser mucho mayor que la capacidad del vehículo Q. Las experiencias computacionales recomiendan como valor inicial para λ

$$\lambda = \frac{Ave}{(20 \cdot CMAX)}$$

donde Ave = $\frac{\sum_{i=1}^{n} q_i}{n}$, y CMAX = max d_{ij}..

También se suele elegir $\lambda = 0'05$.

El algoritmo hace modificaciones sobre λ sólo *en el sentido* de cambiar desde una solución no factible para el VRP a otra que si lo sea y cuando la consigue para. Se recomienda un aumento geométrico, por ejemplo $\lambda = 2 \cdot \lambda$ hasta encontrar una solución factible.

3.- ALGORITMOS EXACTOS

La mayoría de los métodos de solución exactos para el VRP son adaptaciones de los algoritmos Branch & Bound para el TSP, especialmente el de Little. Entre estas adaptaciones destaca la propuesta por Christofides & Eilon (1.969). Durante los últimos años, los *avances* o mejoras en este tipo de algoritmos han venido por el desarrollo de métodos para el cálculo de cotas inferiores y superiores en las ramas del árbol de búsqueda, más *ajustadas* al valor óptimo. En este sentido destacan los métodos de Relajación Espacio-Estado con el uso de la Programación Dinámica de Christofides, Mingozzi y Toth, (1.981), adaptados al VRP.

3.1.- PROGRAMACION DINAMICA: METODOS DE RELAJACION

Christofides, Mingozzi & Toth, (1.981), extienden al VRP las técnicas de *relajación* propuestas para el TSP descritas en el capítulo uno.

Sea la red G=(X,A), donde X son los nodos que representan a los clientes y al depósito inicial x_1, y A es el conjunto de arcos que unen los diferentes nodos; sea X'=X-{x_1} el conjunto de clientes.

Para el caso del VRP, se proponen unas formulaciones en términos de Programación Dinámica, y sus correspondientes *relajaciones*, que se describen a continuación.

3.1.1.- FORMULACIONES

3.1.1.1.- Formulación 1

Sea $S \subset X'$, y m' entero entre 1 y m. Se define f(m',S) como el menor coste de visitar al conjunto S de clientes con m vehículos, y v(S) como el coste de la solución óptima del TSP definido para el conjunto de clientes S y el depósito x_1.

Se tiene la siguiente fórmula recursiva:

$$f(m',S) = \min_{L \subset S} [f(m'-1,S-L) + v(L)], \quad (1)$$

sujeto a:

$$\sum_{x_j \in S} q_j - \left(m' - 1\right) \leq \sum_{x_j \in L} q_i \leq Q, \quad (2)$$

para m' = 2,...,m y S ⊆ X' satisfaciendo:

$$Q^* - \left(m - m'\right) \leq \sum_{x_j \in S} q_i \leq m' \cdot Q, \quad \text{donde } Q^* = \sum_{x_j \in X'} q_j, \quad (3)$$

para m' = m, solo hay que considerar S = X'.

Se inicializa con f(1,S)=v(S).

3.1.1.2.- Formulación 2.

Se define f(m',S,k) como el menor coste de satisfacer un conjunto S de clientes usando m' vehículos, donde los últimos clientes que se visitan en cada una de estas rutas pertenecen al conjunto $\{x_2,x_3,...,x_k\}$. Sea v(S,k) el coste asociado a la solución óptima del TSP, correspondiente a visitar el conjunto S, partiendo de x_1, y siendo x_k el último cliente que se visita. La recursión queda de la siguiente forma:

$$f(m',S,k) = \min[f(m',S,k-1), \min_{L \subset S} f(m'-1,S-L,k-1)+v(L,k)], \qquad (4)$$

sujeto a:

$$\sum_{x_j \in S} q_j - \left(m' - 1\right) \cdot Q \leq \sum_{x_j \in L} q_j \leq Q. \qquad (5)$$

3.1.2.- RELAJACIONES

3.1.2.1.- Relajación de la Formulación 1

Para $g(S) = q = \sum_{x_j \in S} q_j$, se obtiene la relajación

$$f(m', q) = min\{f(m' - 1, q - p) + v*(p)/Q^* - (m' - 1) \cdot Q \leq p \leq \min[q, Q]\},$$

donde v*(p) en este caso viene dada por

$$v^*(p) = \min_{x_i \in \Gamma^{-1}(x_1)}[l(p,x_i) + c_{i1}],$$

siendo $l(p,x_i)$ el coste del *circuito con carga total q que finaliza en* x_p en este nuevo conjunto de estados según se definió en las recursiones relajadas para el TSP.

Una cota inferior en el VRP viene dada por f(m,Q*).

3.1.2.2.- Relajación de la Formulación 2

Se define la misma función de proyección g(S) que en el caso anterior. La recursión queda de la siguiente forma:

$$f(m',q,k) =$$
$$\min\left[f(m',q,k-1), \min\left\{f(m'-1,q-p,k-1)+v^*(p.k)/Q^* - (m-1)\cdot Q \le p \le \min[q,Q]\right\}\right].$$

De forma análoga a la anterior, se define $v^*(p,k) = l(p,x_k)+c_{k1}$. Una cota inferior para el VRP viene dada por f(m,Q*,n) ($\geq$ f(m,Q*), definida antes).

3.2.- OTROS METODOS EXACTOS

3.2.1.- ADAPTACION DEL ALGORITMO DE LITTLE PARA EL VRP

Christofides & Eilon (1.969), proponen una estrategia Branch & Bound para el VRP mediante la aplicación del algoritmo de Little y otros, (1.963), para el TSP en la que mejoran el cálculo de las cotas inferiores en cada rama. Para ello se formula el VRP como el TSP, eliminando el depósito real e incorporando N depósitos artificiales como ya se ha hecho en otras ocasiones.

3.2.1.1.- Algoritmo

Antes de *ramificar* un nodo en el árbol de búsqueda, se *chequean* las siguientes cuestiones:

- (a) ¿Para cada vehículo, la ruta que se está formando excede la capacidad de ese vehículo?
- (b) ¿Excede la distancia máxima o radio máximo de acción de ese vehículo?
- (c) Si se han formado k rutas y quedan por formarse N-k, ¿Estás últimas pueden no satisfacer las demandas de los n' clientes sin asignar a las rutas ya formadas?; es decir

$$¿\frac{\left(\sum_{k=1}^{n'} q_i\right)}{C} > N-k?.$$

Si alguna de estas respuestas es positiva se corta la exploración en esa dirección del árbol de búsqueda.

3.2.1.2.- Mejoras de las cotas inferiores

En caso de matrices simétricas se pueden mejorar las cotas inferiores de la siguiente forma. En la ramificación de cada vértice del árbol de búsqueda considérese la red formada por los arcos correspondientes a los elementos de la matriz reducida. Sea G el grafo que resulta de no considerar las direcciones esta red.

Sobre este grafo G se construye el Mínimo Arbol de Expansión, añadiendo la arista de longitud mínima. La longitud total de este nuevo grafo (árbol más arista), más el coste acumulado en pasos anteriores va a ser una cota inferior de la solución del TSP para estos vértices, y por tanto de la solución del TSP que incluya las aristas que han sido seleccionadas; (se supone la desigualdad triangular en la matriz de distancias original).

Se observa que si a la solución al TSP para los vértices de un determinado grafo G, se le *quita* una arista queda convertida en un árbol de expansión. Por tanto la longitud del mínimo árbol de expansión en este grafo G, más la de la arista más *corta*, es una cota inferior de este TSP. (Ver figuras 15 y 16).

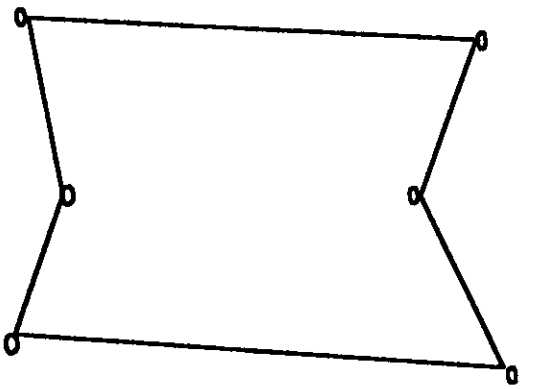


Figura 15. Ruta óptima para el TSP

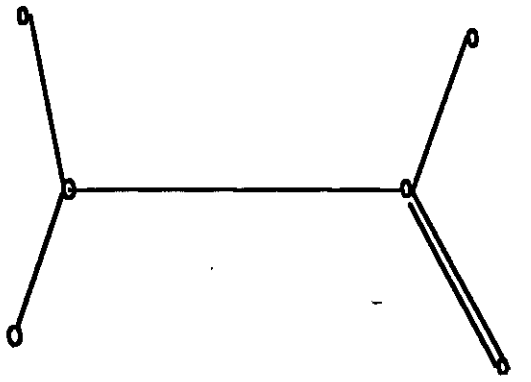


Figura 16. Determinación de cotas inferiores para el TSP: Mínimo Arbol de Expansión añadiendo el tramo más corto.

4.- CONCLUSIONES Y MEJORAS

4.1.- CONCLUSIONES

A continuación se exponen algunas de las conclusiones que se derivan acerca del comportamiento y eficacia los algoritmos para el VRP, tanto por experiencias propias como por las de otros autores. Muchas de estas conclusiones son análogas a las que se extrajeron en el capítulo 1 para los algoritmos del TSP debido al parecido planteamiento de ambos problemas y a que los algoritmos para el VRP son básicamente adaptaciones de los del TSP:

- Al igual que en el caso del TSP los algoritmos heurísticos son preferibles a los exactos, sobretodo si se dispone de poco tiempo para obtener la solución o el tamaño de los problemas es grande. Está preferencia se debe a que en los algoritmos Branch & Bound para el VRP es difícil obtener cotas inferiores ajustadas al óptimo y por consiguiente el tiempo de computación es aún mayor que en el TSP.

- La eficacia de los algoritmos heurísticos depende de los algoritmos escogidos para resolver el TSP en alguna de sus fases o pasos. Así en los algoritmos del tipo *Ruta 1º-Cluster 2º* en la obtención de la ruta inicial, si el problema no es de gran tamaño, es aconsejable utilizar un algoritmo exacto especialmente en matrices simétricas; en problemas de mayor tamaño es preferible la utilización de un algoritmo r-óptimo, puesto que el tiempo de computación es mucho menor y la solución obtenida apenas se empeora.

- Los algoritmos de tipo *Cluster 1º-Rutas 2º* son más eficaces si se usan algoritmos exactos en la segunda fase, puesto que el tamaño de cada uno de los subproblemas que se generan es *asequible* y una mejora, aunque sea ligera, en cada una de las rutas da lugar a una mejora más importante en la solución final.

- Los algoritmos del tipo *Cluster 1º-Ruta 2ª*, entre los que se puede incluir también el algoritmo de Fisher & Jaikumar, son en general los de mayor eficacia, como reflejan Benavente y otros, (1.985), Fisher y otros, (1.981) y (1.988), Haouari, (1.990),.... Sin embargo esta eficacia depende de obtener una representación de los puntos en un plano de tal forma que las distancias euclídeas en éste coincidan o se aproximen, con la distancias reales. Habitualmente se usan los mapas geográficos, pero a veces es necesario una representación más ajustada. En este sentido se recomienda, como paso previo en estos algoritmos, el uso de métodos factoriales para la representación de matrices de distancias (Cuadras, (1.991)).

- Los algoritmos del tipo *Ruta 1º- Cluster 2º* son especialmente eficaces para el CCP, frente a otros. En el trabajo de Bodin & Kursh, (1.978) y (1.979), se indica que la aplicación de estos algoritmos al problema de determinar los recorridos de vehículos limpiadores de calle en la ciudad de Nueva York, podían producir ahorros de 4 o más vehículos, dependiendo de los horarios de aparcamientos frente a otro tipo de técnicas. En cuanto a la aplicación de estos algoritmos al VRP el más eficaz es el expuesto por Beasley, (1.983), aunque puede ser mejorado con pequeñas modificaciones como se verá en el siguiente apartado.

4.2.- MEJORAS EN EL ALGORITMO DE BEASLEY

4.2.1.- IDEA BASICA

En esta sección se describe una pequeña modificación que mejora en muchas ocasiones la eficacia del algoritmo de Beasley para el VRP, especialmente en problemas en los que hay que minimizar el número de vehículos, pudiéndose incorporar esta mejora a otros algoritmos de tipo *Ruta 1º-Cluster 2º*.

Considérese un problema con depósito inicial O, 3 clientes A, B y C, con demandas respectivas 4, 6 y 8 unidades, y vehículos con capacidad de 10 unidades. Supóngase que en la primera fase se obtiene una *gran ruta* de la siguiente forma:

O - A - C - B - O

Con esta gran ruta inicial la única partición posible que se obtiene en pasos posteriores consta de tres rutas, una por cliente:

O - A - O, O - B - O y O - C - O;

sin embargo, se observa que existe una solución con sólo dos rutas de la forma:

O - A - B - O y O - C - O.

En este caso con el algoritmo de Beasley, o en general con las técnicas de tipo *Ruta 1º - Cluster 2º*, no se puede llegar a esta solución ya que al estar el cliente C intercalado entre los clientes A y B impide que éstos puedan pertenecer a la misma ruta.

Para evitar situaciones de este tipo se propone incorporar al algoritmo de Beasley un paso previo, consistente en separar del conjunto de clientes aquellos que por su demanda deben ir solos en una ruta, como en el ejemplo anterior el cliente C; de esta forma el algoritmo de Beasley modificado consiste en los siguientes pasos:

- Extraer los clientes que deben ir solos y asignarles una ruta a cada uno de ellos;

-Ejecutar el algoritmo de Beasley para el problema con los restantes clientes;

- Registrar en una misma solución las rutas obtenidas en los dos pasos anteriores.

Obsérvese que utilizando esta modificación se puede llegar en el problema anterior a la solución de dos rutas antes mencionada.

4.2.2.- RESULTADOS COMPUTACIONALES

En esta sección se muestran los resultados obtenidos con los algoritmos de Beasley, y de Beasley Modificado, en la resolución de un conjunto de problemas con matrices de distancias simuladas. Se han realizado 10 simulaciones para cada número de nodos: 10, 15, 20, 25, 30, 35, 40, 45 y 50. Algunos aspectos de la simulación son los siguientes:

- A cada cliente i se la asigna un punto p_i en una retícula cuadrada de longitud 100: las coordenadas de este punto toman valores enteros uniformes entre 0 y 99.

- Los elementos (i,j) de la matriz de distancias, se obtienen como $d_{ij} = \lfloor \|p_i - p_j\|_2 \rfloor$. Posteriormente se perturban estos valores multiplicándolos por un coeficiente aleatorio entre 0.9 y 1.1.

- Las demandas q_i se obtienen generando valores enteros con distribución uniforme entre 3 y 6. Se toma la capacidad de los vehículos como 8 unidades de carga.

A continuación se muestra un cuadro que resume los resultados medios obtenidos por cada algoritmo en cuanto a número de vehículos, distancia total recorrida y tiempo de computación (en segundos).

	Beasley			**Beasley Modificado**		
Nodos	*Dist. Total*	*Vehículos*	*T.Comput.*	*Dist. Total*	*Vehículos*	*T.Comput.*
10	725.6	6.5	0.018	691.1	4.8	0.015
15	1138.8	10.5	0.061	1145.9	8.5	0.065
20	1344.1	13.4	0.142	1304.2	10.4	0.105

	Beasley			Beasley Modificado		
Nodos	*Dist. Total*	*Vehículos*	*T.Comput.*	*Dist. Total*	*Vehículos*	*T.Comput.*
25	1607.8	18.4	0.195	1562.5	14.0	0.175
30	2258.0	20.9	0.362	2176.8	15.5	0.253
35	2765.9	24.3	0.567	2680.4	19.1	0.428
40	3120.9	26.5	0.785	3064.3	21.6	0.625
45	3517.3	32.7	1.028	3340.3	25.1	0.751
50	3790.1	33.9	1.440	3710.6	28.1	1.098

En el siguiente cuadro se muestra más claramente la evolución del número medio de vehículos obtenidos:

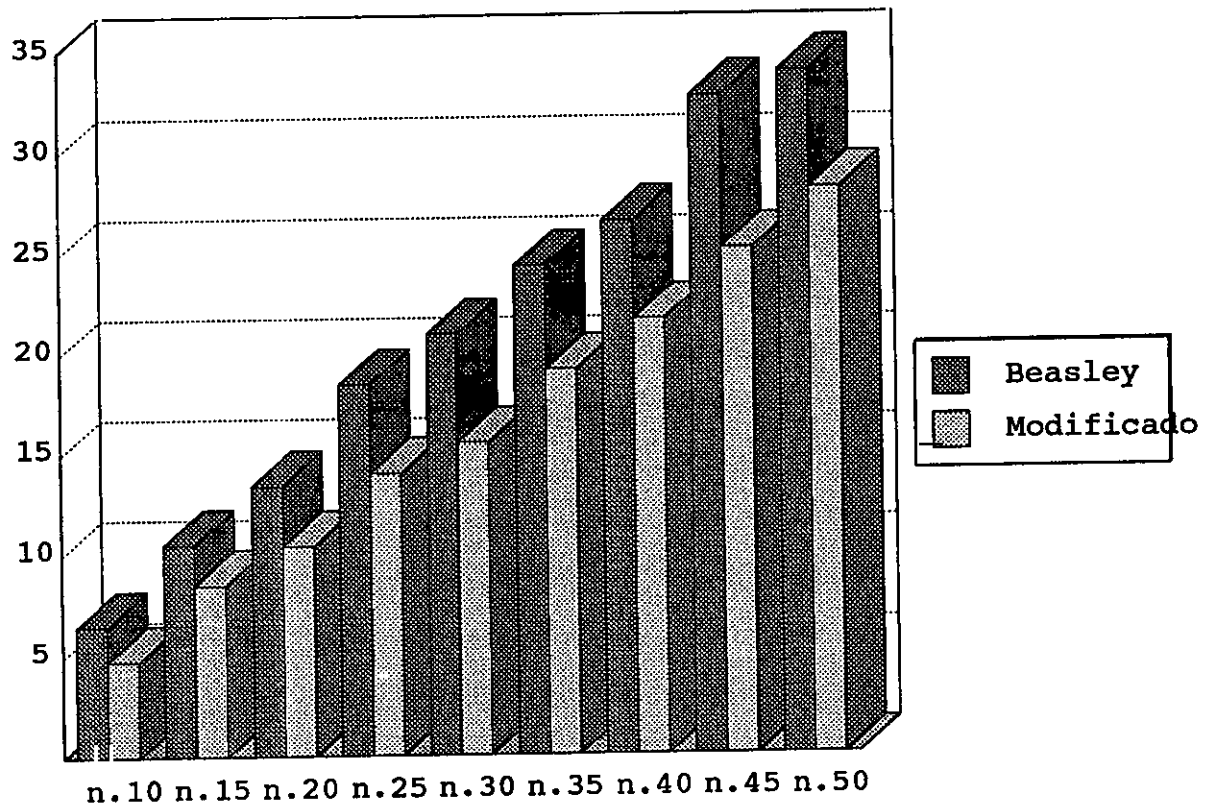


Figura 17. Evolución del número medio de vehículos en función del número de nodos.

4.2.3.- OBSERVACIONES

Los resultados obtenidos permiten concluir que la modificación propuesta, además de poder incorporarse fácilmente al algoritmo original supone ahorros importantes en cuanto al número de vehículos que se obtiene. Además este mejora también se refleja en la distancia total obtenida aunque de forma más suave. El tiempo de computación también resulta ser ligeramente inferior con la modificación descrita. Por consiguiente se concluye que está modificación es especialmente interesante en aquellos problemas en los que hay que minimizar el número de vehículos a usar.

Esta modificación puede ser adaptable fácilmente a problemas con otras restricciones además de la carga total como el radio máximo de conducción, ventanas de tiempo, compatibilidad de mercancías, etc.,... De esta forma todas estas restricciones son chequeadas con el fin de extraer de la lista inicial los clientes que deben ir solos en una ruta.

Para ahorrar tiempo de computación en problemas de gran tamaño, una mejora en esta modificación consiste en extraer de la lista no únicamente los clientes que deben ir solos, sino además los que solamente pueden ir acompañados de otros clientes con los que formarían una ruta *mala* aunque esta sea factible, (por ejemplo supóngase dos nodos muy alejados). Para ello se ha de establecer previamente un criterio para determinar cuando una ruta es *mala* o no.

CAPITULO 3. ADAPTACION DE LOS ALGORITMOS DE RUTAS A LAS VENTANAS DE TIEMPO

1.- INTRODUCCION

Muchos de los modelos y algoritmos diseñados para resolver problemas de Rutas sólo consideran aspectos *espaciales* y no son capaces de captar o recoger todo el conjunto de restricciones del problema práctico. Una de estas restricciones es que la llegada del vehículo, o vendedor, a la localización de cada cliente debe producirse en una determinada *ventana de tiempo*, es decir, intervalo de tiempo en el cual éste puede ser servido. La incorporación de estas restricciones da lugar a Problemas Mixtos de Programación y Rutas, y *requiere* algoritmos en los que además se tengan en cuenta aspectos temporales.

En este capítulo, siguiendo el esquema propuesto por Desrochers y otros, (1.988), se van a describir los principales métodos de solución para problemas de rutas con restricciones de ventanas de tiempo. Los problemas que consideraremos son el Problema del Viajante (TSPTW), el Problema de Rutas de Vehículo (VRPTW), el Problema de Cargas y Descargas (PDPTW), y el Dial-A-Ride Problem (DARPTW). Tal como se ha venido haciendo, los algoritmos se dividirán en exactos y heurísticos. Entre los primeros se consideran fundamentalmente las técnicas Branch & Bound, las basadas en Programación Dinámica y las basadas en técnicas de Partición de Conjuntos. Entre los algoritmos heurísticos destacan los métodos Constructivos, los métodos de Mejora e Intercambio y la Optimización o Búsqueda Incompleta.

En los modelos descritos a continuación, $N = \{2,3,...,n\}$ denota el conjunto de clientes y $A = \{(i,j) / i, j \in N \cup \{1\}, i \neq j\}$ el conjunto de arcos. Los problemas que se tratan en este capítulo no

agotan los existentes. Otras formulaciones de problemas de rutas con ventanas de tiempo se pueden encontrar en los trabajos de Daganzo, (1.987a) y (1.987b), y Tsitsiklis (1.992).

1.1.- FORMULACIONES

Las siguientes formulaciones se deben a Desrochers y otros, (1.988).

1.1.1.- TSP CON VENTANAS DE TIEMPO (TSPTW)

Básicamente el TSPTW es un TSP en el que cada ciudad lleva asociado un intervalo de tiempo $[e_i,l_i]$, i = 1,...,n, en el que tiene que ser visitada: si se llega a la ciudad i antes del instante e_i hay que esperar hasta ese momento, y nunca se puede llegar más tarde del instante l_i. Se supone que el tiempo de salida inicial es e_1.

Se definen:

d_{ij} : distancia entre las ciudades i y j;

t_{ij} : tiempo de trayecto entre las ciudades i y j;

y las siguientes variables

$$x_{ij} = \begin{cases} 1, \text{ si se va a j inmediatamente después de i;} \\ \\ 0, \text{ en caso contrario;} \end{cases} \qquad i,j = 1,...n;$$

D_i : Tiempo de llegada a la ciudad i.

El problema se formula como sigue:

$$\text{minimizar } \Sigma_{i,j=1}^{n} d_{ij} \cdot x_{ij}$$

sujeto a:

$$\Sigma_{j=1}^{n} x_{ij} = 1, \quad i = 1,...,n;$$

$$\Sigma_{j=1}^{n} x_{ij} - \Sigma_{j=1}^{n} x_{ji} = 0, \quad i = 1,...,n;$$

$$x_{ij} = 1 \Rightarrow D_i + t_{ij} \leq D_j, \quad i = 2,...,n; \quad j = 1,...,n;$$

$$x_{1j} = 1 \Rightarrow D_j \geq e_1 + t_{1j}, \quad j = 2,...,n;$$

$$e_i \leq D_i \leq l_i, \quad i = 1,...,n;$$

$$x_{ij} \in \{0,1\}, \quad i,j = 1,...,n.$$

1.1.2.- VRP CON VENTANAS DE TIEMPO (VRPTW).

El VRPTW es un VRP en el que a cada ciudad se le asocia un intervalo de tiempo $[e_i,l_i]$, i = 1,...,n, dentro del que tiene que ser visitada; si se llega a la ciudad i antes del instante e_i se debe esperar hasta ese momento, y nunca se puede llegar después de l_i. Se supone que el tiempo de salida inicial es e_1.

Se definen:

d_{ij} : distancia entre las ciudades i y j;

t_{ij} : tiempo de trayecto entre las ciudades i y j;

y las siguientes variables

$$x_{ij} = \begin{cases} 1, \text{ si se va a j inmediatamente después de i,} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i,j = 1,...n; \\ 0, \text{ en caso contrario;} \end{cases}$$

D_i : Tiempo de llegada a la ciudad i por algún vehículo, $i = 2,...,n;$

y_i : carga que lleva el vehículo cuando llega a la ciudad i, $i = 2,...,n.$

El problema se formula de la siguiente manera:

$$\text{minimizar } \Sigma_{i,j=1}^{n} d_{ij} \cdot x_{ij}$$

sujeto a:

$$\Sigma_{j=1}^{n} x_{ij} = 1, \quad i = 2,...,n;$$

$$\Sigma_{j=1}^{n} x_{ij} - \Sigma_{j=1}^{n} x_{ji} = 0, \quad i = 1,...,n;$$

$$x_{ij}=1 \Rightarrow D_i + t_{ij} \leq D_j, \quad i,j = 2,...,n;$$

$$x_{1j}=1 \Rightarrow D_j \geq e_1 + t_{1j}, \quad j = 2,...,n;$$

$$x_{j1}=1 \Rightarrow D_j + t_{j1} \leq l_1, \quad j = 2,...,n;$$

$$e_i \leq D_i \leq l_i, \quad i = 1,...,n;$$

$$x_{ij}=1 \Rightarrow y_i + q_i \leq y_j, \quad i,j = 2,...,n;$$

$$x_{j1}=1 \Rightarrow y_j + q_j \leq Q, \quad j = 2,...,n;$$

$$0 \leq y_i \leq Q, \quad i = 2,...,n;$$

$$x_{ij} \in \{0,1\}, \quad i,j = 1,...,n.$$

1.1.3.- PROBLEMA DE CARGA Y DESCARGA CON VENTANAS DE TIEMPO (PDPTW)

En este caso 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. Cada punto j de carga o descarga debe ser visitado dentro de un intervalo de tiempo $[e_j,l_j]$. Se trata de diseñar rutas de vehículos con distancia total a recorrer mínima.

Se definen :

$N^+ = \{i^+ / i \in N\}$ el conjunto de orígenes, $N^- = \{i^- / i \in N\}$ el conjunto de destinos,

$V = N^+ \cup N^- \cup \{1\}$;

c_{ij} : distancia entre los puntos i y j, $\forall i,j \in V$;

t_{ij} : tiempo de travesía entre los puntos i y j, $\forall i,j \in V$,

Q : capacidad máxima de cada vehículo;

y las siguientes variables:

$$x^k_{ij} = \begin{cases} 1, \text{ si el vehículo k viaja desde i a j,} \\ 0, \text{ en caso contrario,} \end{cases} \qquad \forall i,j \in V; \quad k = 1,...,m;$$

D_i : Tiempo de llegada al punto i por algún vehículo, $\forall i \in N^+ \cup N^-$;

y_i : Carga que lleva el vehículo cuando llega a la ciudad i, $\forall i \in N^+ \cup N^-$.

Una posible formulación es la siguiente:

$$\text{minimizar } \Sigma_{k=1}^{m} \Sigma_{i,j \in V}^{n} c_{ij} \cdot x_{ij}^{k}$$

sujeto a:

$$\Sigma_{k=1}^{m} \Sigma_{j \in V} x_{ij}^{k} = 1, \quad i \in N^{+};$$

$$\Sigma_{j \in V} x_{ij}^{k} - \Sigma_{j \in V} x_{ji}^{k} = 0, \quad i \ N^{+} \cup N^{-}, \quad k = 1,...,m;$$

$$\Sigma_{j \in V} x_{i+j}^{k} - \Sigma_{j \in V} x_{ji-}^{k} = 0, \quad i \in N, \quad k = 1,...,m;$$

$$x^{k}_{ij}=1 \Rightarrow D_i + t_{ij} \leq D_j, \quad i,j \in N^{+} \cup N^{-}, \quad k = 1,...,m;$$

$$e_i \leq D_i \leq l_i, \quad i \in N^{+} \cup N^{-};$$

$$x_{i+j}^{k} = 1 \Rightarrow y_i + q_i \leq y_j, \quad i \in N, \quad j \in N^{+} \cup N^{-}, \quad k = 1,...,m;$$

$$x_{i-j}^{k} = 1 \Rightarrow y_i - q_i \leq y_j, \quad i \in N, \quad j \in N^{+} \cup N^{-}, \quad k = 1,...,m;$$

$$0 \leq y_i \leq Q, \quad i \in N^{+} \cup N^{-};$$

$$x^{k}_{ij} \in \{0,1\}, \quad i,j \in V, \quad k = 1,...,m.$$

Una variante muy frecuente de estos tres problemas consiste en minimizar el tiempo de llegada al depósito central.

El caso especial en el que las demandas requeridas son todas iguales se denomina el *Dial-A-Ride-Problem* (DARP), que tiene lugar en el transporte de viajeros, transporte escolar, etc.

2.- ALGORITMOS EXACTOS

Los algoritmos exactos para problemas de rutas con ventanas de tiempo emplean principalmente dos métodos de enumeración implícita: la programación dinámica y las técnicas Branch & Bound. La Programación Dinámica es aplicable a problemas con un sólo vehículo en los que hay pocos clientes. Sin embargo, el número de estados que se derivan de este planteamiento crece rápidamente, y para problemas con un número de nodos más elevado es más aconsejable la relajación de este conjunto de estados para su incorporación a métodos Branch & Bound.

Entre los métodos Branch & Bound destacan dos tipos de algoritmos. El primer tipo es un conjunto de técnicas que parten de una formulación del problema como particionamiento de conjuntos, usan métodos de generación de columnas para resolver una relajación continua del problema y métodos Branch & Bound para obtener la solución entera. El otro tipo de técnicas usan relajaciones del conjunto o espacio de estados, asociado a una formulación de Programación Dinámica, para obtener cotas inferiores en las ramas del árbol de búsqueda. El algoritmo de Baker, (1.983), también de tipo Branch & Bound, obtiene las cotas en cada rama mediante resolución de problemas de camino más largo.

2.1.- PROGRAMACION DINAMICA

2.1.1.- PROBLEMAS DE UN SOLO VEHICULO CON VENTANAS DE TIEMPO

2.1.1.1.- Formulación del TSPTW de Christofides y otros

Considérese el conjunto de estados de la forma (S,j), con $S \subset N$, $j \in S$, y f(S,j) indica la más corta duración a través de una ruta factible que comienza en 1, visita todos los vértices de S y

finaliza en j. La solución óptima f(N∪{n+1},n+1) viene determinada por las siguientes ecuaciones de recurrencia:

$$f(\{i\},i) = e_1+t_{1i},$$
$$f(S,j) = \min_{i\in S-\{j\},(i,j)\in A}\{f(S-\{j\},i)+t_{ij}\}, \quad \text{para } j \in N\cup\{n+1\},$$

se redefine:

$$f(S,j) = e_j \text{ si } f(S,j) < e_j,$$
$$f(S,j) = \infty \text{ si } f(S,j) > l_j.$$

2.1.1.2.- Formulación del TSPTW en 'línea de costa' de Psaraftis y otros

Más recientemente, Psaraftis, Solomon, Magnanti y Kim, (1.990), proponen una modificación de la anterior recursión para el caso en que los nodos estén situados en una *línea de costa*, es decir, que se pueda encontrar una reordenación de los nodos 1, 2, 3, ..., n, de forma que $\forall i,j/\ 1 \le i \le k \le j \lneq n$, se cumpla $t_{ij} = t_{ik}+t_{kj}$. Este caso se da en muchas aplicaciones: trenes, ríos, autopistas,... o, en general, problemas de transporte marítimo donde las localizaciones o puertos están situadas en un casco convexo.

Los autores estudian el caso en el que $l_i = \infty$. Sin pérdida de generalidad se supone que el vehículo comienza en el origen 1 en el tiempo t=0. La anterior recursión queda de la forma siguiente:

$$f(S,i) = \min_{y\in S-\{i\}}\{\max(e_i, t_{yi}+f(y,S-\{i\}))\},\ \forall S\subseteq\{2,\ldots,n\};$$

con la condición inicial $f(\{i\},i) = \max(e_i, e_1+t_{1i})$.

En el caso en que no sea necesario volver al origen, ni finalizar en ningún nodo predeterminado, se simplifica el conjunto de estados de la anterior recursión; sólo es necesario considerar los estados de la forma (S,i), con S e i definidos de la forma siguiente:

a) $S = S_1 \cup S_2$ con $S_1 = \{x / 2 \le x \le j\}$, y $S_2 = \{x / k \le x \le n\}$ para algunos índices j y k tales que $1 \le j < k \le n+1$; (por convenio $S_2 = \varnothing$, si $k = n+1$);

b) Si $S_2 = \varnothing$, entonces $i = j$. En caso contrario, $i = j$ o $i = k$.

De esta forma solo dos índices, j y k, son necesarios para representar el conjunto S de la recursión, (Ver Figura 18), y por tanto, ésta puede reescribirse de la siguiente forma

$$f^*(j,k,j) = \min\{\max(e_j, t_{j-1,j} + f^*(j-1,k,j-1)), \max(e_j, t_{kj} + f^*(j-1,k,k))\},$$

$$f^*(j,k,k) = \min\{\max(e_k, t_{k+1,k} + f^*(j,k+1,k+1)), \max(e_k, t_{jk} + f^*(j,k+1,j))\},$$
$$\forall j,k \,/\, 1 \le j < k \le n+1,$$

con $f^*(1,n+1,1) = e_1$, como condición inicial.

Por convenio, en la recursión se considera $f^*(0,k,0) = f^*(j,n+1,n+1) = +\infty$, $\forall k, j / k > 1, j < n$.

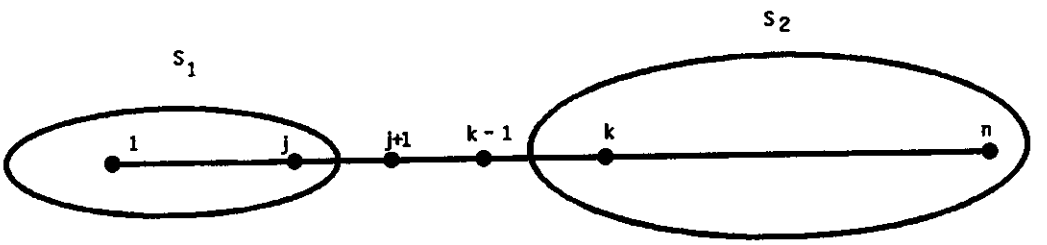


Figura 18. Representación de los conjuntos S_1 y S_2 en el caso de línea de costa

Esta recursión puede ser ejecutada en un tiempo $\theta(n^2)$, y el valor óptimo del problema viene dado por:

$$C^*_{\max} = \min_{1 \le j \le n} f^*(j,j+1,j) = \min_{1 \le k \le n+1} f^*(k-1,k,k).$$

2.1.1.3.- Formulación del DARPTW con un vehículo de Psaraftis

En este caso se trata de minimizar el tiempo total y 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:

$$k_i = \begin{cases} -1, & \text{si la demanda del cliente i no ha sido cargada,} \\ 0, & \text{si esta demanda ha sido cargada en } i^+, \text{ pero no descargada,} \\ +1, & \text{si la demanda ha sido descargada en } i^- ; \end{cases}$$

así, sólo se considerará el conjunto de estados de la forma (j^+, k_2,..., k_n), tales que $k_j = 0$, y de la forma (j^-, k_2,..., k_n) con $k_j = +1$;

entonces, la solución óptima viene dada por $\min_{j^- \in N^-} f(j^-, +1, \ldots, +1)$,

con las siguientes ecuaciones de recurrencia

$$f(j^+, k_2, \ldots, \overbrace{0}^{j}, \ldots, k_n) =$$

$$\min\Big(\min_{\substack{(i^+ j^+) \in A \\ k_j = 0}} f(i^+, k_2, \ldots, \overbrace{-1}^{j}, \ldots, k_n) + t_{i^+ j^+}, \min_{\substack{(i^- j^+) \in A \\ k_j = +1}} f(i^-, k_2, \ldots, \overbrace{-1}^{j}, \ldots, k_n) + t_{i^- j^+}\Big).$$

para $j^+ \in N^+$;

$$f(j^-, k_2, \ldots, \overbrace{+1}^{j}, \ldots, k_n) =$$

$$\min\Big(\min_{\substack{(i^+ j^-) \in A \\ k_j = 0}} f(i^+, k_2, \ldots, \overbrace{0}^{j}, \ldots, k_n) + t_{i^+ j^-}, \min_{\substack{(i^- j^-) \in A \\ k_j = +1}} f(i^-, k_2, \ldots, \overbrace{0}^{j}, \ldots, k_n) + t_{i^- j^-}\Big)$$

para $j^- \in N^-$;

las condiciones iniciales son $f(j^+, -1, -1, \ldots, \overbrace{0}^{j}, \ldots, -1) = t_{0j^+}$, para $j^+ \in N$.

Además, se redefine $f(j, k_2,..., k_n) = e_j$ si $f(j, k_2,..., k_n) < e_j$; y $f(j, k_2,..., k_n) = \infty$, si $f(j, k_2,..., k_n) > l_j$, $\forall j \in N^+ \cup N^-$.

El algoritmo se desarrolla en 2n etapas, en cada una de las cuales se extiende el camino eligiendo el arco adecuado. El total de tiempo requerido es de $\theta(n^2 \cdot 3^n)$. Psaraftis estima que con esta técnica se puede resolver problemas de hasta 10 clientes. Desrosiers y otros, (1.986a), adaptan este algoritmo de 2n estados, al PDPTW con un sólo vehículo.

Las técnicas descritas en este apartado también pueden usarse como subrutinas en otros métodos de solución, por ejemplo, en los de tipo *Cluster 1°-Ruta 2ª*. Ejemplos de este tipo de aproximaciones pueden verse en los trabajos de Desrosiers, Soumis & Desrochers, (1.986b), Desrosiers, Sauvé & Soumis, (1.988).

2.2.- METODOS DE RELAJACION DEL ESPACIO DE ESTADOS

2.2.1.- RELAJACION DE CHRISTOFIDES Y OTROS

En esta sección se describen métodos para la obtención de cotas inferiores usando Programación Dinámica y espacios de estados relajados.

2.2.1.1.- Formulación

La relajación del espacio de estados está basada en una proyección g del espacio de estados original a un espacio de menor cardinalidad de tal forma que, si existe una transición desde S_1 a S_2 en el estado de espacios original, debe existir también una transición desde $g(S_1)$ a $g(S_2)$ en el nuevo espacio de estados. Esta idea la aplicaron los autores al TSPTW de la forma siguiente.

Para cada vértice i, se asocia un entero arbitrario β_i, con $\beta_1 = \beta_{n+1} = 0$. La proyección se define por g(S,j) = (k, β, j), donde k=|S|, y $\beta = \Sigma_{k \in S} \beta_k$. (Según se vió en anteriores capítulos, se pueden definir diversas proyecciones g(S)). Las ecuaciones de recurrencia quedan:

$$f(0,\beta,1) = \begin{cases} 0 & \text{si } \beta = 0 \\ \infty & \text{si } \beta \neq 0. \end{cases}$$

$$f(k,\beta,j) = \min_{i \neq j,(i,j) \in A} \{f(k-1, \beta - \beta_j, i) + t_{ij}\}, \ \forall j \in N \cup \{n+1\};$$

donde se redefine:

$$f(k,\beta,j) = \begin{cases} e_j & \text{si } f(k,\beta,j) < e_j \\ \infty & \text{si } f(k,\beta,j) > l_j. \end{cases}$$

La cota inferior viene dada por $\min_{j \in N;(j,n+1) \in A} \{f(n-1, \Sigma_{j \in N} \beta_j, j) + t_{j,n+1}\}$.

Esta cota inferior, y las obtenidas de forma análoga, pueden mejorarse usando penalizaciones en los vértices y modificaciones en el espacio de estados. Estas penalizaciones son ajustadas por métodos de optimizacion del subgradiente de igual forma que los pesos β_i.

2.2.2.- ADAPTACION AL VRPTW DE KOLEN Y OTROS

Kolen, Rinnoy Kan & Trinekens, (1.987), extienden los métodos de relajación propuestos por Christofides, Mingozzi & Toth, (1.981), al VRPTW.

2.2.2.1.- Exploración en cada vértice

A cada vértice α del árbol de búsqueda le corresponde un conjunto $F(\alpha)$ de rutas fijas, que comienzan y finalizan en el depósito 1, una ruta parcial $P(\alpha)$ que comienza en el depósito central, y un conjunto $C(\alpha)$ de clientes que *tienen prohibido* incorporarse a la ruta parcial $P(\alpha)$. En el vértice inicial $F(\alpha)$ y $C(\alpha)$ están vacíos y $P(\alpha)$ consta sólo del depósito central.

Inicialmente se calcula una cota inferior en el conjunto de todas las posibles extensiones de la solución parcial caracterizada por $F(\alpha)$, $P(\alpha)$ y $C(\alpha)$, según se muestra en 2.2.2.2. En la ramificación se selecciona un cliente j que no esté en las rutas fijas ni en la ruta parcial, ni tampoco *esté prohibido*. En la ramificación se crean dos nuevos nodos α' y α''. En α', la ruta parcial $P(\alpha)$ se extiende con j y $C(\alpha') = C(\alpha)$. En α'', $P(\alpha'') = P(\alpha)$ y $C(\alpha'') = C(\alpha) \cup \{j\}$. Si $j = 1$, la ruta parcial extendida se añade al conjunto de rutas fijas y se genera una nueva ruta parcial, ($P(\alpha') = \{1\}$, $C(\alpha') = \varnothing$).

El cliente j que se elige es el primero que aparece en la extensión de esta ruta parcial en el cálculo de la cota inferior. Si no hay ruta parcial, $P(\alpha) = \{1\}$, se comienza una nueva seleccionando el cliente que aparece más frecuentemente en las rutas calculadas en la cota inferior. En caso de empate, se recomienda elegir el cliente con mayor demanda, ya que esto hará reducir más el tamaño del problema en el cálculo de las cotas inferiores en pasos posteriores.

2.2.2.2.- Cálculo de las cotas

Para el cálculo de las cotas inferiores se usa la relajación del espacio de estados en dos niveles:

Paso 0.: Poner $H = N - F(\alpha) \cup P(\alpha) \cup C(\alpha)$ y $QTOT = \Sigma_{i \in H} q_i$;
Supóngase $H = \{2,3, ..., l\}$.

Paso 1.: $\forall j \in H, \forall q = 0, ..., QTOT$, hacer :

Definir con el conjunto de clientes H el problema del camino mínimo desde 1 hasta j, con carga total q de los clientes visitados;

Poner $F(q,j)$ = Valor óptimo de este problema + d_{j1}.

Si la ruta parcial $P(\alpha)$ no es vacía :

Poner k^* = último cliente de $P(\alpha)$:

Definir en H el problema del camino mínimo desde k^* hasta j, con carga total q de los clientes visitados;

Poner $F^*(q,j)$ = Valor óptimo de este problema + d_{j1} .

Paso 2.: Poner K^* = m -cardinal($F(\alpha)$).

Establecer la siguiente red:

- Definir el conjunto de nodos v(k,q,j), para k = 1,...,K^*, q = 0,...,QTOT, $j \in H$, correspondientes a k caminos de carga total q y que comienzan en 1 y cuyos últimos clientes se encuentran en {2,...,j}, junto con un nodo inicial v(0,0,1).
- Definir el conjunto de arcos que unen los nodos v(k,q,j) y v(k+1,q',j+1) con coste F(q'-q,j+1), para k =0,...,K^*-1 y q > q', j = 1,...,l-1;
- Si la ruta parcial $P(\alpha)$ no esta vacía añadir el conjunto de nodos $v^*(k,q,j)$, para k = 0,...,K^*, q = 0,...,QTOT, $j \in H$, correspondientes a k caminos de carga total cuyos últimos clientes se encuentran en {2,...,j} que comienzan en el nodo 1 los k-1 primeros y en el nodo k^* el último. Añadir los arcos necesarios.

Si la ruta parcial $P(\alpha)$ está vacía hallar el camino mínimo en esta red desde en nodo v(0,0,1) hasta el nodo v(K^*,l,QTOT);

en caso contrario desde v(0,0,1) a v^*(K^*,l,QTOT) .

2.3.- PARTICION DE CONJUNTOS

Desrosiers y otros, (1.984), proponen un método para resolver el VRPTW, reformulándolo como un problema de partición de conjuntos.

Se definen

Ω : conjunto de rutas factibles para el VRPTW;

y_r : variable que toma el valor 1 si la ruta es usada y 0 en caso contrario;

δ_{ri} = { 1 si el cliente i es visitado en la ruta r,
0 en caso contrario;

c_r : coste de la ruta r (si el número de vehículos no es fijo se incluye también el coste inicial por vehículo).

El problema se puede formular:

$$\text{minimizar} \sum_{r\in\Omega} c_r y_r$$

sujeto a:

$$\Sigma_{r\in\Omega} y_r \delta_{ri} = 1, \quad i \in N;$$

$$y_r \in \{0,1\}, \quad r \in \Omega.$$

Teorema 3.1.-

Si en este problema se *relaja* o cambia la restricción $y_r \in \{0,1\}, r \in \Omega$, por la siguiente $0 \le y_r \le 1$, $r \in \Omega$, (*relajación lineal*), el problema resultante tiene al menos una solución entera óptima.

Demostración.- Ver Desrosiers y otros, (1.984).

2.3.1.- GENERACION DE COLUMNAS

Como el número de rutas en Ω es grande, la relajación lineal del problema anterior se resuelve por generación de columnas. En cada iteración se generan nuevas columnas de mínimo coste

marginal resolviendo el siguiente problema del camino mínimo con restricciones de tiempo (SPTW). Se definen

t_i : tiempo de llegada al punto i;

$$x_{ij} = \begin{cases} 1 \text{ si se usa el arco (i,j)}, \\ 0 \text{ en caso contrario}; \end{cases}$$

σ_i : coste marginal resultante de la resolución de la relajación lineal del problema anterior;

el problema se formula

$$\min \sum_{(i,j)\in A} (c_{ij} - \sigma_i) \cdot x_{ij}$$

sujeto a:

$$\sum_{j\in N\cup\{1\}} x_{ij} = \sum_{i\in N\cup\{1\}} x_{ji}, \quad i \in N;$$

$$\sum_{j\in N} x_{1j} = \sum_{j\in N} x_{j1} = 1,$$

$$x_{ij} > 0 \Rightarrow t_i + t_{ij} \le t_j, \quad (i,j) \in A;$$

$$e_i \le t_i \le l_i, \quad i \in N;$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in A.$$

En cada iteración se generan las rutas necesarias para cubrir todas las demandas; estas rutas corresponden a las columnas que se utilizan para resolver por el método símplex la relajación lineal del problema de partición de conjuntos. La resolución de este problema proporciona los costes marginales necesarios para la resolución del SPTW que da lugar a nuevas columnas. Esquemáticamente el proceso es el siguiente:

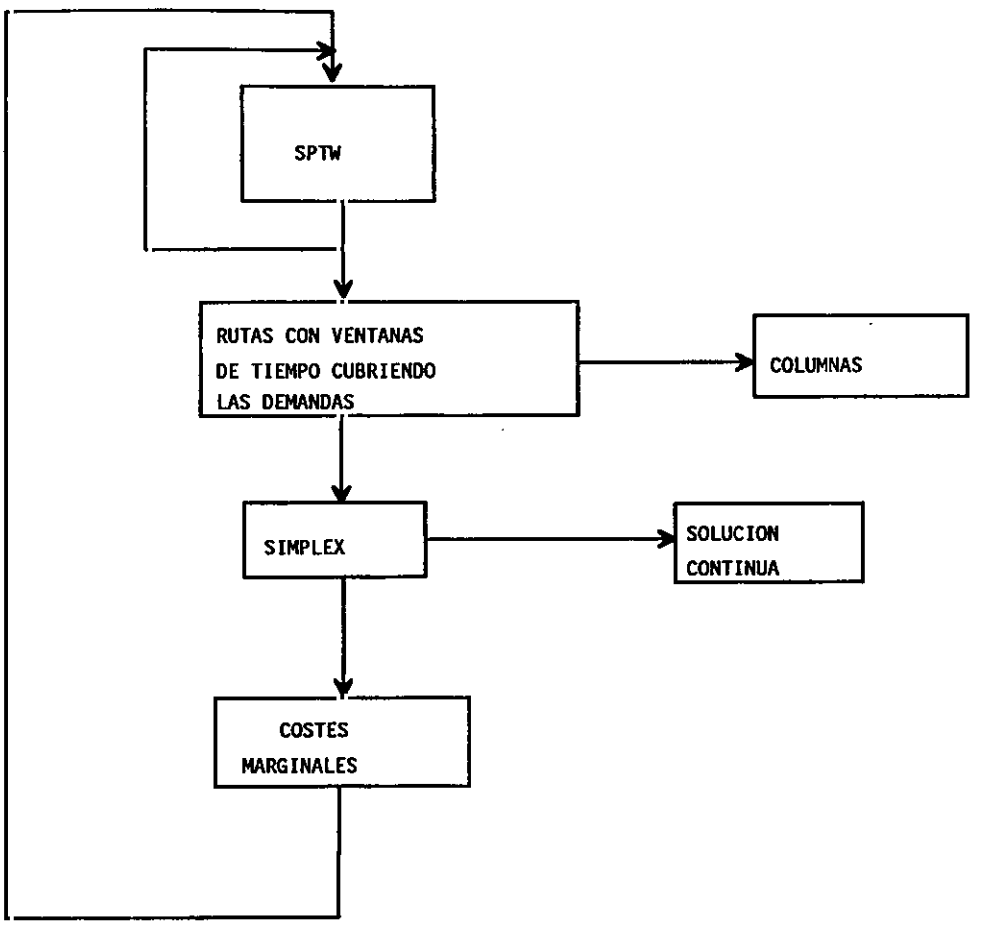


Figura 19. Esquema del proceso iterativo en el que en cada iteración se generan varias columnas

La experiencia, según Desrosiers y otros, (1.984), indica que la posibilidad de obtener una solución entera es alta; de cualquier forma si no se obtienen soluciones enteras, se propone incorporar el procedimiento anterior a un proceso Branch & Bound en el que se ramifica el conjunto de soluciones.

2.3.2.- PROCESO DE RAMIFICACION

La ramificación se realiza en los arcos correspondientes a la ruta con menor coste adicional de acuerdo con la estimación dada por los costes marginales, de entre las asociadas a las variables y_r

que no sean enteras. Por ejemplo, si y_r es la variable con valor no entero elegida y la ruta correspondiente está formada por los arcos (0,1), (1,2) y (2,0), la ramificación es la siguiente:

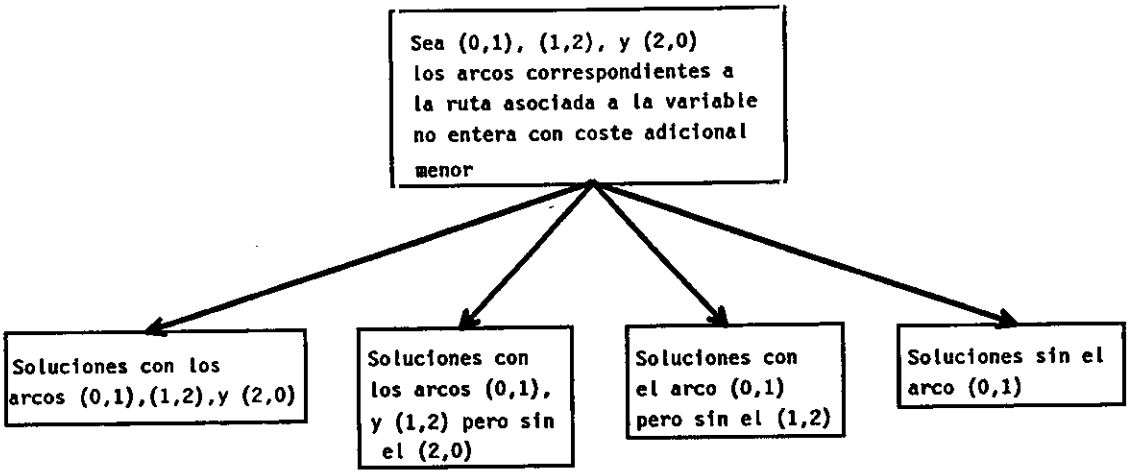


Figura 20. Esquema del árbol de ramificación

2.4.- OTROS METODOS: ALGORITMO DE BAKER

Entre los métodos exactos diferentes a los anteriormente descritos, destaca el propuesto por Baker, (1.983), en el que presenta un algoritmo Branch & Bound para el TSPTW, con el objetivo de minimizar el tiempo total de viaje y donde las cotas se derivan de la resolución de problemas del camino más largo.

Sea u_i la variable la variable que indica el instante de llegada a la ciudad i, i = 1,...,n. Como hay que volver a 1 al final del recorrido, se define una variable adicional u_{n+1} que determina cuando finaliza el recorrido. Se asume que la matriz de tiempos de viaje (t_{ij}) es completa, simétrica y no negativa. Además se supone que esta matriz cumple la desigualdad triangular. Baker redefine el TSPTW como

minimizar u_{n+1}-u_1 (1)

sujeto a:

$$u_i - u_1 \geq t_{1i}, \quad i = 2,...,n; \qquad (2)$$

$$|u_i - u_j| \geq t_{ij}, \quad i = 3,...,n;\ 2 \leq j < i; \qquad (3)$$

$$u_{n+1} - u_i \geq t_{i1}, \quad i = 2,...,n; \qquad (4)$$

$$u_i \geq 0, \qquad i = 1,..., n+1; \qquad (5)$$

$$e_i \leq u_i \leq l_i, \qquad i = 1,...,n. \qquad (6)$$

2.4.1.- IDEA BASICA: RESOLUCION DEL PROBLEMA DUAL

El método Branch & Bound comienza *relajando* las restricciones con valor absoluto (3) y las restricciones de ventanas de tiempo (6) de la formulación propuesta. El problema *relajado* resultante es

(P) minimizar u_{n+1}-u_1

sujeto a:

$$u_i - u_1 \geq t_{1i}, \quad i = 2,..., n;$$

$$u_{n+1} - u_i \geq t_{i1}, \quad i = 2,..., n;$$

$$u_i \geq 0, \qquad i = 1,..., n+1.$$

La solución del problema P se obtiene observando que su dual es un problema de camino más largo en una red dirigida con n+1 nodos. El problema dual D es:

$$\text{(D)} \qquad \text{maximizar } \Sigma_{j=2}^{n}\left(t_{j1}x_j + t_{j1}x_{j+n}\right)$$

sujeto a:

$$-\Sigma_{j=2}^{n} x_j \leq -1;$$

$$x_j - x_{j+n} \leq 0, \qquad j = 2,\ldots, n;$$

$$\Sigma_{j=2}^{n} x_{j+n} \leq 1,$$

$$x_j, x_{j+n} \geq 0, \qquad j = 2,\ldots, n.$$

(Obsérvese que para j = 2, ...,n la variable x_j indica el número de veces que se utiliza el arco (1,j) en la red dual y x_{j+n} el número de veces que se utiliza el arco (j,n+1). El nodo n+1 que se añade, se corresponde con el *regreso* a la ciudad inicial en el problema original). La solución del problema dual (D) es una cota inferior del problema original.

En cada ramificación se crean dos subproblemas correspondientes a cada desigualdad a que dan lugar las restricciones con valor absoluto $|u_i - u_j| \geq t_{ij}$. En cada subproblema se incorporan las restricciones lineales correspondientes al problema P. Estas restricciones en el problema primal crean columnas (variables) adicionales en el problema dual. Cada una de estas columnas en el dual se corresponde con los arcos dirigidos que se van añadiendo en la red definida por (D).

En los vértices terminales del árbol de búsqueda se incorporan fácilmente las restricciones (6) correspondientes a las ventanas de tiempo.

3.- ALGORITMOS HEURISTICOS

En esta sección se describen tres tipos de algoritmos de aproximación: los métodos de *Construcción*, que forman una solución factible a partir de la matriz de datos; métodos de *Mejora iterativa*, que partiendo de una solución factible llegan a otra mejor mediante una serie de modificaciones locales; y los métodos de *Optimización Incompleta*, que combinan métodos de enumeración del espacio de soluciones (Branch & Bound principalmente) y criterios heurísticos para truncar esta búsqueda.

Un estudio comparativo de estos algoritmos se puede encontrar en el trabajo de Haimovich y otros, (1.988) y más recientemente en Haouari y otros, (1.990).

En esta sección, se asumirá, sin pérdida de generalidad, que una ruta viene dada por (1,...,i,...,n,n+1), donde i es el (i-1)-ésimo cliente visitado por el vehículo a partir del nodo inicial. En la descripción de los algoritmos siguientes se utilizarán las siguientes cantidades asociadas a cada camino (h,...,k):

El *posible salto hacia adelante* S^+_{hk}, que se define como el mayor incremento posible en el tiempo de salida D_h de h que no viola las restricciones de tiempo a lo largo del camino (h,...,k); se formula como

$$S^+_{hk} = \min_{h \le j \le k} \{ l_j - (D_h + \Sigma_{h \le i < j} t_{i,i+1}) \}.$$

El *posible salto hacia atrás* S^-_{hk}, que se define como el mayor adelanto posible en D_h, que no produce tiempos de espera a lo largo de (h,...,k); se formula como

$$S^-_{hk} = \min_{h \le j \le k} \{ D_j - e_j \}.$$

El tiempo requerido para el cálculo de estas cantidades es de orden $\theta(n)$.

3.1.- METODOS DE CONSTRUCCION

En el diseño de los métodos de construcción hay dos cuestiones importantes:

(1) Criterio de Selección: es decir, determinar en cada etapa que cliente se selecciona para insertar en la ruta que se está construyendo.
(2) Criterio de Inserción: determinar donde se va a insertar dicho cliente.

Mientras unos algoritmos resuelven estas cuestiones al mismo tiempo (*vecino más cercano* o *ahorros*), otros (algoritmos de inserción), las resuelven separadamente. En esta sección se describen adaptaciones para algunos algoritmos de este tipo.

3.1.1.- ADAPTACION DEL METODO DE AHORROS O 'SAVINGS'

Es un procedimiento en el que inicialmente cada cliente define una ruta por sí mismo. En cada iteración, se selecciona un arco para combinar dos rutas en una, de forma que se ahorre más coste y se respeten las restricciones de capacidad del vehículo. Sólo se consideran arcos desde el último cliente de una ruta al primero de otra.

Si dos rutas se combinan, puede haber un incremento en este tiempo de llegada al primer cliente de la segunda ruta.

Teorema 3.2.-

Una condición necesaria para la factibilidad, es que el nuevo tiempo de llegada al primer cliente de la segunda ruta no sea posterior a la finalización de su tiempo de servicio. Además se ha de verificar que este incremento no supere S^{+}_{hk}, siendo h y k respectivamente el primer y el último cliente de la segunda ruta.

Para seleccionar un arco que sea *efectivo* en cuanto al ahorro del coste y tenga en cuenta las ventanas de tiempo, se debería *unir* dos clientes cuyas ventanas de tiempo estén cercanas. Para ello se deben hacer modificaciones en el algoritmo original con vistas a seleccionar arcos que tengan en cuenta la cercanía espacial y temporal de los clientes. Estas modificaciones pueden consistir, por ejemplo, en añadir penalizaciones a los ahorros de costes si hay tiempo de espera.

3.1.2.- ADAPTACION DEL METODO 'VECINO MAS CERCANO'

Inicialmente, la ruta consta del depósito solamente; en cada iteración se añade a la ruta actual el cliente que no esté en ella y sea el más cercano al último cliente de la ruta actual. El proceso finaliza cuando no haya más clientes que seleccionar.

La selección de un cliente se restringe a aquellos cuya incorporación a la ruta sea factible respecto a las restricciones de capacidad y ventanas de tiempo. Cuando los clientes aún no seleccionados *fallan* en estas condiciones se forma una nueva ruta.

La medida de *cercanía* puede incluir tanto aspectos espaciales y temporales. Solomon, (1.986), propone la siguiente:

$$\alpha_1 t_{ij} + \alpha_2(\max\{e_j, D_i + t_{ij}\} - D_i) + \alpha_3(l_j - (D_i + t_{ij})),$$

que considera el tiempo de trayecto entre los clientes i y j, la diferencia entre sus respectivos tiempos de llegada y la urgencia de descargar en j.

3.1.3.- ADAPTACION DE LOS METODOS DE INSERCION

Los métodos de inserción tratan los criterios de selección e inserción de forma separada. La forma de insertar puede ser secuencial si se hace uno a uno, o paralela, si se insertan varios a la vez. Las adaptaciones consideradas aquí son para métodos secuenciales.

El esquema general de un método de inserción es simple. Sean ι y σ los criterios de inserción y selección; sea U el conjunto de clientes que no están en la ruta actual. Para cada cliente u ∈ U, se determina el mejor punto i_u después del cual puede ser insertado de forma factible en dicha ruta, es decir:

$$\iota(u, i_u) = optimo_{\ 1 \le i \le n}\{\iota(u, i)\}, \quad \text{para u} \in \text{U}.$$

A continuación se selecciona el cliente u* que va a ser insertado en la ruta:

$$\sigma(u^*, i_{u^*}) = optimo_{u \in U}\{\sigma(u, i_u)\}.$$

En las modificaciones de estos métodos, los criterios ι y σ de inserción y selección se deben definir teniendo en cuenta consideraciones espaciales y temporales. Igual que antes, se inicializa una nueva ruta cuando no es factible realizar más inserciones. El proceso continúa hasta que todos los clientes estén en alguna ruta.

La inserción de u entre i e i+1 puede cambiar los tiempos de llegada en la subruta (i+1,...,n+1); por tanto será factible si y sólo si el tiempo de llegada a i+1 no es mayor que $S^*_{i+1,n+1}$.

3.1.3.1.- Criterios de inserción y selección

Entre los más utilizados cabe destacar los siguientes:

(i) Sean

$$\iota_1(u, j) = c_{i,u} + c_{u,i+1} + \mu \cdot c_{i,i+1}.$$

y

$$\iota_2(u, i) = D^n_{i+1} - D_{i+1},$$

donde D^n_{i+1} es el nuevo tiempo de llegada al cliente i+1, después de insertar u en la ruta; entonces se define:

$$\iota(u,i) = \alpha_1 \cdot \iota_1(u,i) + \alpha_2 \cdot \iota_2(u,i), \qquad \text{donde } \alpha_1 + \alpha_2 = 1, \text{ y } \alpha_1, \alpha_2 \geq 0;$$

y

$$\sigma(u,i) = \lambda \cdot c_{1u} - \iota(u,i), \qquad \text{donde } \lambda \geq 0.$$

Según estos criterios, la mejor inserción factible para un cliente que no esté en ninguna ruta, es la que minimiza la combinación de coste de viaje extra y tiempo extra requerido para la visita de dicho cliente. De esta forma, cuando $\mu = \alpha_1 = \lambda = 1$ y $\alpha_2 = 0$, $\sigma(u,i)$ va a ser el ahorro de costes que se produce cuando se sirve al cliente u entre los clientes i e i+1, frente a servirlo de forma individual.

(ii) Sea $\iota(u,i)$ definido como en (i);

y

$$\sigma(u,i) = \beta_1 \cdot R_c(u) + \beta_2 \cdot R(u), \quad \text{donde } \beta_1 + \beta_2 = 1, \ \beta_1, \beta_2 \geq 0;$$

$R_c(u)$ y $R_t(u)$ son, respectivamente, el coste total y el tiempo total cuando u se inserta en la ruta actual.

(iii) Sean $\iota_1(u,i)$ y $\iota_2(u,i)$ definidas como antes, $\iota_1(u,i) = l_u - D_u$;se define

$$\iota(u,i) = \alpha_1 \cdot \iota_1(u,i) + \alpha_2 \cdot \iota_2(u,i) + \alpha_3 \cdot \iota_3(u,i), \quad \text{donde } \alpha_1 + \alpha_2 + \alpha_3 = 1, \text{ y } \alpha_1, \alpha_2, \alpha_3 \geq 0.$$

y

$$\sigma(u,i) = \iota(u,i).$$

En esta tercera aproximación, a los aspectos temporales usados en los anteriores criterios de inserción, se incorpora la *urgencia* de servicio a los clientes.

3.1.4.- ADAPTACIONES DE METODOS CONSTRUCTIVOS PARA EL DARP

Jaw, Odoni, Psaraftis & Wilson, (1.986), describen un algoritmo constructivo para una variante del DARP con ventanas de tiempo.

Para la identificación de inserciones factibles se introduce el concepto de *periodos activos*, en los que se realiza el transporte de clientes, situados entre *periodos ociosos*, en los que no se transportan clientes. (Se omiten los superíndices + y -; genéricamente el tiempo de visita a i se denota por V_i). Para cada visita a i, situada en un determinado periodo activo, se definen las siguientes cuatro variantes de los saltos hacia adelante y hacia atrás:

$$\Sigma_i^- = \min\{\min_{j\le i}\{V_j - e_j\}, \Lambda\},$$

$$\Sigma_i^+ = \min_{j\le i}\{l_j - V_j\},$$

$$S_i^- = \min_{j\ge i}\{V_j - e_j\},$$

$$S_i^+ = \min\{\min_{j\ge i}\{l_i - V_i\}, L\},$$

donde Λ y L son las duraciones de los tiempos de ocio inmediatamente precedente y siguiente de el periodo activo que se está considerando.

Los mínimos se toman para las localizaciones visitadas en ese periodo activo; Σ_i^- (Σ_i^+) indica la máxima cantidad de tiempo que se pueden adelantar, (retrasar), cada parada anterior a i+1 sin violar las restricciones de las ventanas de tiempo. Análogamente, S_i^- (S_i^+) indica la máxima cantidad de tiempo que se pueden adelantar, (retrasar), las paradas posteriores a i-1 sin violar dichas restricciones. Estas cantidades indican como se debe *desplazar* cada segmento de un periodo activo para acomodar clientes adicionales, como se muestra en la figura 21.

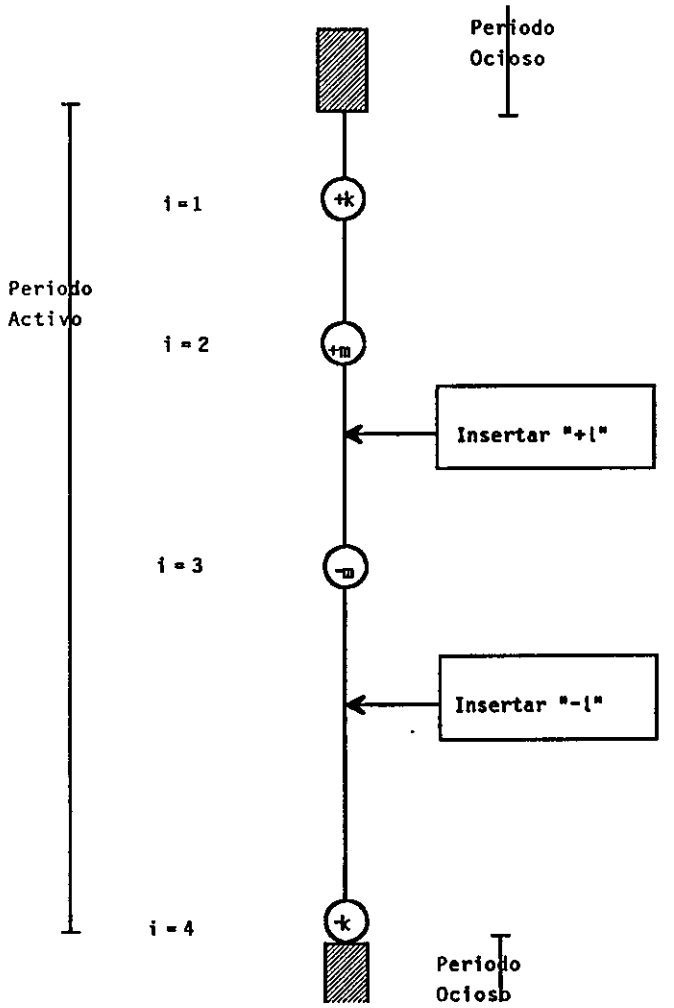


Figura 21. Inserción de un cliente l en un bloque activo existente

Si se quiere añadir la carga de un cliente l entre la segunda y la tercera parada del bloque activo, el tiempo extra requerido para esta visita viene dado por $t_{2,l^+} + t_{l^+,3} - t_{23}$. Si esta cantidad es menor que $\Sigma^-_2 + S^+_3$ entonces la inserción es factible. De igual forma se debe chequear la factibilidad de la inserción de la descarga del cliente l.

3.2.- ADAPTACION DE LOS METODOS DE MEJORA ITERATIVOS

En esta sección se extienden los procedimientos de k-intercambios para el TSPTW y el VRPTW. En el caso del TSTPW, una modificación en el tiempo de llegada a un punto afecta a los tiempos de partida en toda la ruta; por tanto se ha de tener en cuenta la factibilidad de estos k-intercambios. Desrochers y otros, (1.988) incorporan métodos para adaptar el algoritmo de Or, (1.976), a las ventanas de tiempo; análogamente, Solomon, Baker & Schaffer, (1.988a), proponen métodos para reducir los k-intercambios que deben ser considerados.

3.2.1.- ADAPTACION DEL ALGORITMO DE OR

Or, (1.976), 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. 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.

Para establecer una estrategia de búsqueda y método para el chequeo de la factibilidad de los intercambios Desrochers y otros, (1.988), definen un conjunto de variables globales como se muestra a continuación.

3.2.1.1.- Estrategias de búsqueda

Supóngase que el cliente i se recoloca entre j y j+1; esto hace que los arcos (i-1,i), (i,i+1) y (j,j+1) sean sustituidos por (i-1,i+1), (j,i) e (i,j+1). Los casos de recolocación *hacia atrás* (j < i) y *hacia adelante* (j > i) son tratados separadamente. En el primer caso, j se elige sucesivamente igual a i-1, i-2, ..., 1. En el segundo caso, j toma los valores i+1, i+2, ..., n en este orden. La figura 22 muestra este proceso.

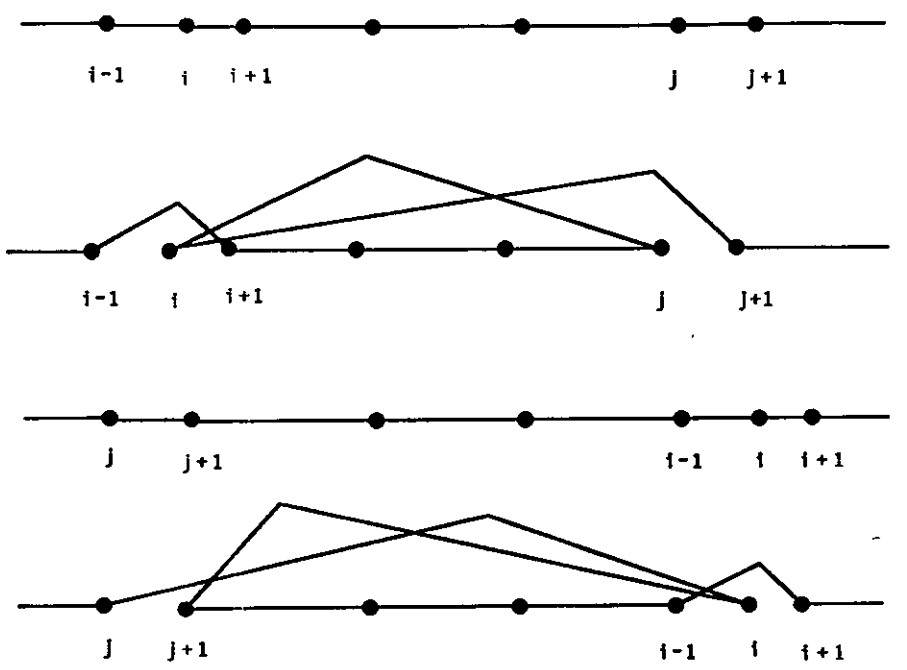


Figura 22. Recolocación hacia adelante y hacia atrás

Las variables globales que se definen son las siguientes:

(1) El posible *salto hacia adelante* S^+, que es igual a $S^+_{j+1,i-1}$; (tal como se definió previamente).
(2) El posible *salto hacia detrás* S^-, que es igual a $S^-_{i+1,j-1}$.
(3) La ganancia de tiempo G que se obtiene por ir directamente desde i-1 a i+1:

$$G = A_{i+1} - (D_{i-1} + t_{i-1,i+1}).$$

(4) La pérdida de tiempo L por ir desde j a j+1 atravesando i:

$$L = \max\{D_j + t_{ji}, e_i\} + t_{i,j+1} - A_{j+1}.$$

(5) El tiempo de espera W que se produce en el camino (j+1,...,i-1):

$$W = \Sigma_{j+1 \le k \le i-1} W_k.$$

3.2.1.2.- Estudio de la factibilidad

Durante la búsqueda *hacia atrás*, un intercambio será factible si $D^n_k \leq l_k$ para k = j+1, ..., i-1, y *potencialmente deseable*, si $D^n_{i+1} < D_{i+1}$; siendo D^n el nuevo tiempo de partida en cada punto si el intercambio tiene lugar. Obsérvese que una disminución en el tiempo de llegada a i+1 no garantiza que se llegue antes al depósito, sin embargo puede ser un buen criterio para aceptar o no un intercambio. En términos globales, la factibilidad y la *deseabilidad en potencia* son equivalentes a:

$$L < \min\{S^+, G+W\}.$$

Una vez realizado el intercambio las variables globales se redefinen:

$$S^+ = W_{j+1} + \min\{l_{j+1}-D_{j+1}, S^+\};$$

$$W = W+W_{j+1}.$$

Durante la búsqueda *hacia adelante*, análogamente, un intercambio es factible si $D^n_i \leq l_i$, y *potencialmente deseable* si $D^n_{j+1} < D_{j+1}$. Esto es equivalente a:

$$L < \min\{S^-, G\}.$$

Las variables globales se redefinen:

$$S^+ = \min\{D_j-e_j, S^-\}.$$

3.2.2.- ADAPTACION DE SOLOMON Y OTROS

Solomon, Baker & Schaffer, (1.988), proponen métodos de chequeo rápidos para eliminar intercambios no factibles. Concretamente describen un método de chequeo para los 2-intercambios y otro para los 3-intercambios.

3.2.2.1.- Definición de valores de precedencia

Estos métodos se basan en el establecimiento de relaciones de precedencia entre los diferentes clientes que deben ser visitados. Más concretamente, para todo par de clientes i,j se examina la siguiente desigualdad

$$e_i + t_{ij} > l_j;$$

en caso de verificarse indicaría que, necesariamente, el cliente j debe preceder al cliente i en la ruta para que ésta sea factible. Como resultado se obtiene una matriz, llamada *matriz de precedencia del vehículo* (VP_{ij}) definida como

$$VP_{ij} = \begin{cases} +1, \text{ si el cliente i debe preceder al cliente j;} \\ 0, \text{ si no existe relación de precedencia;} \\ -1, \text{ si el cliente j debe preceder al cliente i.} \end{cases}$$

Para cada cliente i en la ruta, se define un *valor de precedencia del nodo* NP_i, igual al número del primer cliente situado más allá del cliente i+1, que tiene un predecesor en algún cliente posterior al i en la ruta. Más formalmente se define:

$$NP_i = \min \{k / k > i+1, \text{ y existe un } j \geq i+1 \text{ verificando que } VP_{jk} = +1\};$$

si el anterior conjunto es vacío entonces se define NP_i igual a n+1.

3.2.2.2.- Chequeo de los 2-Intercambios

En cada ruta que se va obteniendo, el vector (NP_i) puede calcularse utilizando $\theta(n^2)$ operaciones a partir de la matriz (VP_{ij}). Una vez definido este vector se establece la siguiente

Condición 3.1.-

Una condición necesaria para la factibilidad del 2-intercambio de los arcos (i,i+1) y (j,j+1) por los arcos (i,j) y (i+1,j+1) es que $j < NP_i$.

Demostración.- Por reducción al absurdo. Si, para algún 2-intercambio $j \geq NP_i$, entonces el segmento de ruta desde j a i+1 que se recorre en orden inverso, contiene una relación de precedencia que se viola al realizar el intercambio.

En la siguiente figura se observa el cambio de orientación.

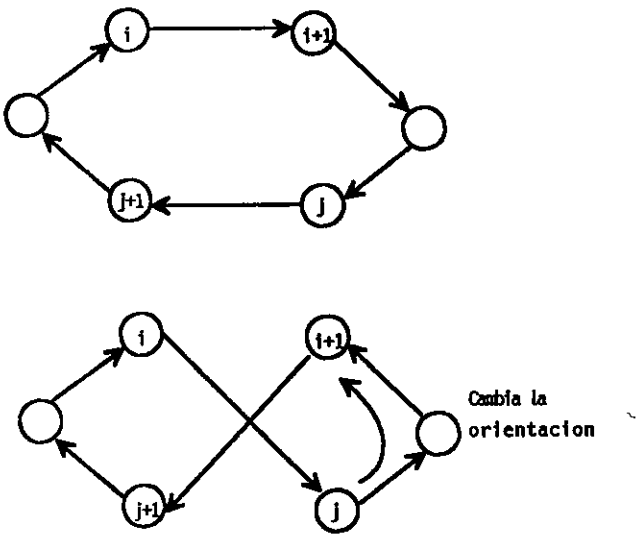


Figura 23. Intercambio 2-óptimo

3.2.2.3.- Chequeo de los 3-Intercambios

De forma análoga se opera para los 3-intercambios, en los que se reemplazan los arcos (i,i+1), (j,j+1) y (k,k+1). El reemplazamiento de estos arcos se puede realizar de varias formas. Por ejemplo, si al realizar un intercambio, los segmentos de ruta desde i+1 a j, y desde j+1 a k, cambian de orden al ser recorridos, entonces es necesario usar la matriz (VP_{ij}) para chequear si hay algún predecesor de alguno de los clientes del segundo segmento desde j+1 a k, en el primero

desde i+1 a j, lo que haría infactible el intercambio. Más concretamente, se define una matriz de predecesores de segmentos SP_{ij} como

$$SP_{ij} = \min\{\ r/\ j+1 \leq r, \text{ y } VP_{qr} = +1 \text{ para algún } q,\ i+1 \leq q \leq j\};$$

si el anterior conjunto es vacío se redefine $SP_{ij} = n+1$.

De esta forma se establece la siguiente

Condición 3.2.-

Una condición necesaria para el intercambio de los arcos (i,i+1), (j,j+1), y (k,k+1) es que se verifique $k < SP_{ij}$.

Demostración.- Al realizar un 3-intercambio se cambia el orden en que se recorren los segmentos desde i+1 a j, y desde j+1 a k; si $k \geq SP_{ij}$ entonces algún cliente en el segmento desde j+1 a k tiene algún predecesor obligatorio en el segmento desde i+1 a j, lo que hace infactible el intercambio.

La figura 24 muestra como un 3-intercambio puede cambiar el orden de visita de los segmentos

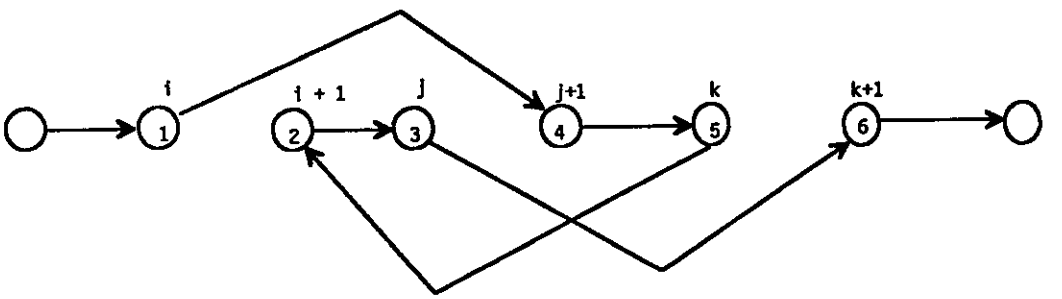


Figura 24. Ejemplo de Intercambio 3-óptimo

4.- CONCLUSIONES

Los problemas de rutas con ventanas de tiempo han sido tradicionalmente poco estudiados en la literatura de la programación matemática; ha sido a principio de los años ochenta cuando se han empezado a desarrollar técnicas de solución para estos problemas. Gran parte de estas técnicas son adaptaciones de los algoritmos ya existentes para los correspondientes problemas estandard. Alguna de las conclusiones que se pueden extraer tanto de la experiencia propia como de la diversos autores, como Haouri, (1.990), Desrochers, (1.988), o Assad, (1.988), sobre estas técnicas son las siguientes:

- Los algoritmos heurísticos son preferibles a los exactos, y por consiguiente han sido mucho más estudiados. Entre las técnicas exactas los métodos más eficaces son, según Assad, (1.988), las técnicas basadas en la relajación del espacio de estados de Programación Dinámica de Christofides y otros, (1.981). En cuanto a los métodos de generación de columnas, en Desrochers y otros, (1.992), se describen algunas mejoras en el método anteriormente propuesto por Desrosiers y otros, (1.984).

- En cuanto a la eficacia de los algoritmos heurísticos las conclusiones son parecidas a las obtenidas en los anteriores capítulos. El método de Fisher & Jaikumar ha demostrado mantener la eficacia cuando se ha adaptado al VRPTW; ejemplo de estas adaptaciones se pueden ver en el trabajo de Savelsbergh, (1.989), y en el de Nygard y otros, (1.988).

- Los métodos r-óptimos, según varios autores como Desrochers y otros, (1.988), siguen siendo los más eficaces para mejorar una ruta factible en estos problemas. Los modificaciones para chequear la factibilidad de los intercambios que se describieron en la sección 3.3. reducen mucho el tiempo de computación. Así en las experiencias computacionales descritas en el trabajo de Solomon y otros, (1.988a), para un conjunto de problemas test el tiempo de computación se reduce entre un 21% y un 57 % en el algoritmo 2-óptimo; entre un 55% y un 84% en el algoritmo 3-óptimo, y entre un 27% y un 64 % en el método de Or.

CAPITULO 4: DESCRIPCION DE UN ALGORITMO PARA EL PROBLEMA DE RUTAS DE VEHICULOS CON VENTANAS DE TIEMPO

1.- INTRODUCCION

En este capítulo se presenta un nuevo algoritmo exacto para el VRPTW asimétrico, así como algunas variantes heurísticas. Se comienza describiendo un algoritmo exacto para el TSPTW, y a continuación se señalan las modificaciones realizadas para adaptarlo al VRPTW. Más adelante, se describirá de que forma este procedimiento exacto puede dar lugar a técnicas heurísticas llamadas de *búsqueda incompleta*, análogas a las descritas por Aragón y Pacheco, (1.992). Finalmente se compararán estos algoritmos entre sí y con otros algoritmos heurísticos a través de un conjunto de problemas tests y con un problema real.

El algoritmo exacto para el TSPTW es de tipo Branch & Bound en el que en los diferentes vértices del árbol de búsqueda se van añadiendo arcos sucesivamente al inicio y al final de la ruta que se va formando. El cálculo de las cotas inferiores en los vértices está basado en procedimientos de relajación del espacio de estados a partir de la formulación del problema en términos de Programación Dinámica, propuesto por Christofides y descrito con detalle en capítulos 1, 2 y 3. Para resolver el problema relajado se adaptará el algoritmo de Dijkstra para el problema del camino mínimo. El valor de la solución determinará la cota inferior en este vértice. Además dicho camino mínimo resultante en el espacio de estados relajado, que se va a denominar *seudoruta*, determinará los arcos que se añaden a la rutas en las siguientes ramificaciones.

Para mejorar el cálculo de las cotas inferiores, se propone un método de penalización lagrangiana, basado en la aplicación a las *seudorutas* de los resultados obtenidos por Held & Karp

para los 1-árboles de expansión, según se vio en la descripción del algoritmo de estos autores para el TSP simétrico en el capítulo 1.

2.-UN ALGORITMO EXACTO PARA EL TSPTW

El algoritmo para el TSPTW es un proceso recursivo de acotación y ramificación a través del árbol de búsqueda. El cálculo de una cota inferior para cada vértice del árbol de búsqueda y la determinación de las ramas en las que se va a dividir el conjunto de soluciones correspondientes a cada vértice dependen de la *seudoruta* asociada al conjunto de soluciones de cada vértice.

2.1.- PROCESO GENERAL DE ACOTACION Y RAMIFICACION

Sea N={1,2,...,n} el conjunto de nodos donde el nodo 1 es el origen y final, y el resto los clientes que deben ser visitados. El cliente i debe ser visitado dentro de un intervalo de tiempo $[a_i,b_i]$. El tiempo de viaje y la distancia entre los nodos i y j vienen dados respectivamente por t_{ij} y d_{ij} (suponemos que el tiempo de visita v_i es cero). Se trata de realizar el trayecto minimizando la distancia total recorrida comenzando en el instante a_1.

A cada vértice α en el árbol de búsqueda, le corresponde un conjunto de arcos que van a constituir una ruta parcial $R(\alpha)$ que se va formando tanto por el comienzo como por el final; más concretamente, $R(\alpha)$ consta del conjunto $R_1(\alpha)$ de arcos que se añaden de forma secuencial al comienzo y el conjunto $R_2(\alpha)$ de arcos que se añaden de forma secuencial desde el final de dicha ruta parcial. Es decir, $R_1(\alpha)$ será de la forma $R_1(\alpha)=\{1\text{-}i_1\text{-}i_2\text{-}\ldots\text{-}i_t\}$, y $R_2(\alpha)=\{i_r\text{-}i_{r+1}\text{-}\ldots\text{-}i_n\text{-}1\}$. Además, cada vértice lleva asociado un conjunto $F(\alpha)$ de arcos que no pueden formar parte de las rutas de ese conjunto de soluciones. Para el tratamiento de estos arcos, se redefine en cada vértice $d_{ij} = \infty, \forall (i,j) \in F(\alpha)$. Inicialmente $R_1(\alpha) = \{1\}$, y $R_2(\alpha) = \{1\}$. De igual forma $F(\alpha) = \varnothing$.

En cada proceso de ramificación se seleciona el conjunto de arcos que se van a añadir a la ruta parcial $R(\alpha)$. La elección de estos arcos y su ordenamiento deben asegurar que se respeta la secuencialidad en la formación de las nuevas rutas parciales. Sean (i_f, i_{f+1}), (i_{f+1}, i_{f+2}), ..., (i_{f+s}, i_{f+s+1}), los arcos que se añaden a la ruta parcial. El conjunto de soluciones correspondientes al vértice α se ramifica en los siguientes nuevos vértices α', α'', ..., α''', α'''', con los siguientes conjuntos asociados $R(\alpha') = R(\alpha) \cup \{(i_f, i_{f+1}), (i_{f+1}, i_{f+2}), ..., (i_{f+s}, i_{f+s+1})\}$ y $F(\alpha') = F(\alpha)$.

Análogamente:

$$R(\alpha'') = R(\alpha) \cup \{(i_f, i_{f+1}), (i_{f+1}, i_{f+2}), ..., (i_{f+s-1}, i_{f+s})\}, \text{ y } F(\alpha'') = F(\alpha) \cup \{(i_{f+s}, i_{f+s+1})\};$$

.....

.....

$$R(\alpha''') = R(\alpha) \cup \{(i_f, i_{f+1})\}, \text{ y } F(\alpha''') = F(\alpha) \cup \{(i_{f+1}, i_{f+2})\};$$

$$R(\alpha'''') = R(\alpha), \text{ y } F(\alpha'''') = F(\alpha) \cup \{(i_f, i_{f+1})\},$$

es decir, en cada ruta parcial $R(\alpha^i)$ se añaden unos arcos y se prohiben los demás.

Una vez realizada la ramificación, se continúa la exploración en el vértice α' hasta que la ruta parcial sea completada. Si el coste de la ruta obtenida en este proceso es inferior al de la mejor obtenida hasta ese momento se registra la nueva solución, y se prosigue analizando las ramas correspondientes a otros vértices.

Simultáneamente con el proceso de cálculo de los arcos que se van a añadir a $R(\alpha)$ se calcula también una cota inferior para el coste óptimo del conjunto de soluciones asociadas a ese vértice α. Si dicha cota inferior es superior al coste de la solución obtenida hasta ese momento se corta la exploración de dicho vértice.

2.2.- CALCULO DE LAS SEUDORUTAS EN CADA VERTICE

Según se ha comentado anteriormente, tanto la selección del conjunto de arcos que se han de añadir en cada vértice del árbol de búsqueda, como el cálculo de la cota inferior asociada a cada vértice está basada en el cálculo de la *seudoruta* en cada vértice.

Sean: $R_1(\alpha) = 1 - i_2 - i_3 - .... - i_f$ $R_2(\alpha) = i_r - i_{r+1} - .... - i_n - 1$

X = Conjunto de nodos que no aparecen ni en $R_1(\alpha)$ ni en $R_2(\alpha)$;

La seudoruta va a estar formada por los arcos de $R_1(\alpha)$ al principio y $R_2(\alpha)$ al final, más un conjunto de arcos de la forma $i_f - j_1 - j_2 - ... - j_K - i_r$, donde $\{j_1, j_2, ..., j_K\} \subset X$ y se añaden entre ambos conjuntos, de forma que en total haya n arcos.

Para el cálculo de este *tramo intermedio* se va a hacer uso de los procedimientos de relajación del espacio de estados propuestos por Christofides y otros, (1.981a).

2.2.1.- PLANTEAMIENTO MEDIANTE PROGRAMACION DINAMICA

El problema consiste en hallar la ruta más corta desde i_f hasta i_r pasando una vez por cada nodo de X. Para $S \subseteq X$, y $j \in S$ se define f(S,j) como el coste mínimo de visitar los nodos de S, respetando las ventanas de tiempo, comenzando en el vértice i_f en el instante T_{i_f} y finalizando en j; siendo T_{i_f} el tiempo de salida del punto i_f después de recorrer los nodos de $R_1(\alpha)$. Sea u(S,j) el momento de llegada a j según el recorrido determinado por f(S,j).

Se verifica: $f(S,j) = \min_{i \in S-\{j\}} \{f(S-\{j\}, i) + d_{ij}\}$;

Si i[j] es el índice correspondiente a este mínimo, entonces $u(S,j)=u(S-\{j\},i[j])+t_{i[j]j}$;

y se redefine:

$u(S,j) = a_j$ si $u(S,j) < a_j$, $\quad f(S,j) = \infty$ si $u(S,j) > b_j$;

con la condición inicial $\quad f(\{j\},j) = d_{1,j}$, $\quad u(\{j\},j) = T_{1,f} + t_{1,j}$.

El problema consiste en determinar $\min_{j \in X} \{f(X,j) + d_{j,1}\}$.

2.2.2.- RELAJACION DEL ESPACIO DE ESTADOS

Este problema, así formulado, utiliza muchas operaciones para su realización, ya que el número de estados (S,j), determinados por esta formulación recursiva, es exponencial en el cardinal de X. Por tanto, se utilizará una proyección o *relajamiento* de este espacio de estados en otro espacio de estados de menor cardinalidad de la siguiente forma:

$f(k,j) = \min_{i \in X-\{j\}} \{f(k-1,i) + d_{ij}\}$; $\quad$ siendo k = |S|.

Si i[j] el índice correspondiente a este mínimo entonces

$u(k,j) = u(k-1, i[j]) + t_{i[j],j}$, para k = 1,2,...,K;

siendo K = |X|, y donde se redefine:

$u(k,j) = a_j$, si $u(k,j) < a_j$, $\quad f(k,j) = \infty$, si $u(k,j) > b_j$;

En este caso, ahora hay que determinar $\min_{j \in X} \{f(K,j) + d_{j,1}\}$;

con la condición inicial: $\quad f(1,j) = d_{1,j}$, $\quad$ y $\quad u(1,j) = T_{1,f} + t_{1,j}$.

La solución obtenida de esta forma no tiene por que ser óptima con respecto al problema recursivo original con los estados iniciales, ya que puede contener nodos de X que aparezcan más de una vez, y nodos que no aparezcan. Sin embargo el cálculo de la solución es mucho más

rápido, ya que el número de estados es polinomial en el cardinal de X, y el valor de la solución proporciona una cota inferior ajustada del valor óptimo del problema con los estados originales. Si $SR(\alpha) = i_f - j_1 - j_2 - \ldots - j_K - i_r$ es el camino obtenido con la formulación relajada, el coste del camino formado por los arcos de $R_1(\alpha)$, $SR(\alpha)$, $R_2(\alpha)$, proporciona una buena cota inferior del valor óptimo del conjunto de soluciones asociadas al vértice α.

Si en el tramo intermedio $SR(\alpha)$ no hay nodos repetidos, la seudoruta hallada constituye una ruta factible y, por tanto, es una solución óptima en el conjunto de soluciones asociadas al vértice α; en consecuencia no es necesario continuar la exploración en este vértice.

La seudoruta hallada en el vértice α será $1 - i_2 - i_3 - \ldots - i_f - j_1 - j_2 - \ldots - j_K - i_r - i_{r+1} - \ldots - i_n - 1$.

A continuación se muestra como se ha adaptado el algoritmo de Djikstra, o de *etiquetado* para el cálculo del tramo $SR(\alpha)$ de las seudorutas.

2.2.3.- ADAPTACION DEL ALGORITMO DE ETIQUETADO

Considérese la red constituida por los nodos de la forma (k,j), para k=1,..,K, y $j \in X$, correspondientes a los estados relajados de la formulación anterior, junto con el nodo inicial $(0,i_f)$, y el nodo final $(K+1,i_r)$. Los arcos de esta red y sus longitudes vienen determinados por las ecuaciones recursivas anteriores. La resolución del problema recursivo es equivalente al cálculo del camino mínimo, en la red que acabamos de definir, entre los nodos $(0,i_f)$ y $(K+1,i_r)$.

El número de nodos de esta red, es K^2+2. El algoritmo de Djkstra para el problema del camino mínimo utiliza $\theta(m^2)$ operaciones, siendo m el número de nodos.

Para cada nodo (k,j) se definen las siguientes variables:

u(k,j) : tiempo de llegada al nodo (k,j), en el camino más corto desde $(0,i_f)$ hasta ese nodo, obtenido hasta ese momento;

v(k,j) : longitud de dicho camino;

θ(k,j) : nodo inmediatamente anterior en dicho camino.

defin(k,j) : Variable booleana que indica si el nodo (k,j) ha tomado la etiqueta permanente o no.

además se utilizarán la siguientes variables auxiliares:

estado, índice: primera y segunda componente respectivamente del nodo que es candidato a ser etiquetado permanentemente.

Procedimiento CALCULOSEUDORUTA

Paso 1.- Inicializar las variables:

$\forall k = 1,..,K$, y $\forall j \in X$ hacer:

$u(k,j) = T_{if}$; $v(k,j) = \infty$; $\theta(k,j) = (0,i_f)$, $defin(k,j)$ = FALSE,.

Paso 2.- Definir etiquetas iniciales:

$\forall j \in X$ hacer:

$u(1,j) = T_{if} + t_{ifj}$; $v(1,j) = d_{ifj}$;

redefiniendo $u(1,j) = a_j$ si $u(1,j) < a_j$; $v(1,j) = \infty$ si $u(1,j) > b_j$.

Paso 3.- Elegir el nodo que va a ser etiquetado definitivamente:

Sea $v(1,j^*) = \min\{v(1,j) / j \in X\}$ poner: $estado = 1$, $índice = j^*$.

Poner $defin(estado,indice)$ = TRUE.

Paso 4.- Cambiar etiquetas:

Poner $i = estado+1$;

Si $i \leq K$, entonces $\forall j \in X - \{indice\}$ poner:

$v' = v(estado,indice) + d_{indice,j}$

$u' = u(estado,indice) + t_{indice,j}$;

redefinir $u' = a_i$ si $uw < a_i$, y $w' = \infty$ si $u' > b_i$;;

si $v' < v(i,j)$ poner $v(i,j) = v'$, $u(i,j) = u'$, $\theta(i,j) = (estado,indice)$.

Paso 5.- Elegir nuevo nodo para ser etiquetado definitivamente:

Sea $v(i^*,j^*) = \min\{v(i,j) / i = 1,..,K; j \in X; defin(i,j) = FALSE\}$,

poner $estado = i^*$, $indice = j^*$, $defin(estado,indice) = TRUE$.

Paso 6.- Verificar si es necesario etiquetar más nodos:

Sea i = K, comprobar si $\forall j \in X$, $defin(i,j) = TRUE$, si es así ir al paso 7;

en caso contrario volver al paso 4.

Paso 7.- Elegir el camino más corto hasta el nodo $(K+1,i_t)$:

$\forall j \in X$, hacer:

$v^*(j) = v(K,j) + d_{ji_t}$

redefiniendo $v^*(j) = \infty$ si $u(K,j) + t_{ji_t} > b_{i_t}$;

Sea $v^*(j^*) = \min\{v^*(j) / j \in X\}$, y poner $estado = K$, $indice = j^*$.

Paso 8.- Registrar dicho camino mínimo desde (0,1) hasta $(K+1,i_t)$:

Inicializar $SR(\alpha) = \varnothing$,

Poner $(indice,i_j) \in SR(\alpha)$;

Repetir:

$estado1 = estado$, $indice1 = indice$,

$(estado,indice) = \theta(estado1,indice1)$,

$(indice,indice1) \in SR(\alpha)$,

hasta que $(estado,indice) = (0,i_p)$.

Poner $wtotal$ = coste del camino formado por $R_1(\alpha)$, $SR(\alpha)$ y $R_2(\alpha)$.

Registrar $SR(\alpha)$, y $wtotal$.

Parar.

2.3.- ELECCION Y ORDENACION DE LOS ARCOS QUE SE AÑADEN A LA RUTA PARCIAL

Los arcos que se añaden a los conjuntos $R(\alpha)$ y $F(\alpha)$ en las nuevas ramificaciones del vértice que se está explorando son los obtenidos en el camino $SR(\alpha)$ y el orden de entrada en las correspondientes ramificaciones dependerá del orden en que aparecen en $SR(\alpha)$.

2.3.1.- ELECCION DE ARCOS

Si $SR(\alpha) = i_f - j_1 - j_2 - \ldots - j_K - i_r$ es el camino obtenido en el cálculo de la *seudoruta,* los nuevos arcos que se van a añadir están en los conjuntos S_1 y S_2, correspondientes respectivamente a los arcos que se añaden a i_f secuencialmente al final de $R_1(\alpha)$ y a i_r al principio de $R_2(\alpha)$. El siguiente procedimiento establece el criterio usado para la elección y ordenación de los nuevos arcos en S_1 y S_2:

Procedimiento SELECCIONARCOS

Paso 1.- Poner $S_1 = S_2 = \varnothing$.

Paso 2.- Añadir el arco (i_f, j_1) a S_1; poner $j^* = j_1$ y $k = 1$;

Paso 3.- Si $k = K$, ir al paso 4;
si no, mientras $k < K$ repetir :
Poner $k = k+1$;
si j_k no está en ninguno de los arcos de S_1 entonces:
añadir (j^*, j_k) a S_1,
Poner $j^* = j_k$.
en caso contrario, poner $k = k+1$ hasta que $j_k = j^*$.

Paso 4.- Repetir el mismo proceso comenzando por i_r:
Si j_K ya está presente en S_1 parar;
si no añadir (j_K, i_r) a S_2, poner $k = K$ y $j^* = j_K$.

Paso 5.- Si $k = 1$, parar;
si no, mientras $k > 1$, repetir:

Poner k = k-1;

si j_k no está presente ni en S_1 ni en S_2 entonces:

añadir (j_k , j*) a S_2,

poner k = k-1;

en caso contrario poner k = k-1 hasta que $j_k = j^*$.

Una segunda alternativa de este proceso consiste en ejecutar los pasos 4 y 5 antes que los pasos 2 y 3, es decir, comenzando por el conjunto S_2 antes que por el S_1. En el siguiente ejemplo ilustrativo se ejecutan las dos alternativas de este procedimiento, eligiéndose la opción que más arcos vaya a seleccionar para incorporarlos a la ruta parcial.

Ejemplo 4. 1.-

Sea un problema definido por n = 7; $a_i = 0$, $b_i = 60$, i= 1,...,7; **T = D**, y la matriz de distancias **D** definida de la siguiente forma:

	1	2	3	4	5	6	7
1	inf.	48	9	83	98	28	0
2	91	inf.	69	5	0	32	79
3	63	1	inf.	18	65	59	8
4	60	15	43	inf.	58	32	79
5	7	88	30	9	inf.	59	48
6	70	3	9	2	99	inf.	35
7	75	47	7	6	92	64	inf.

Supóngase el vértice inicial α del árbol de búsqueda, con, $R_1(\alpha) = R_2(\alpha) = \{1\}$, y $F(\alpha) = \varnothing$. Sea la seudoruta obtenida: 1 - 7 - 3 - 7 - 3 - 2 - 5 - 1;

en esta situación, utilizando la primera alternativa se obtendrían los conjuntos: $S_1 = \{1 - 7 - 3 - 2 - 5\}$ y $S_2 = \{\varnothing\}$;

mientras que utilizando la segunda S_1 y S_2 serían: $S_1 = \{\varnothing\}$ y $S_2 = \{7 - 3 - 2 - 5 - 1\}$;

en ambos casos el número de arcos seleccionados en total es 4 por tanto se puede elegir cualquiera de las dos opciones.

2.3.2.- ORDENACION DE ARCOS

Una vez seleccionados los conjuntos de arcos que se van a incorporar a la ruta parcial en las siguientes ramificaciones de α, el siguiente paso es determinar el orden de entrada y de salida de estos arcos en estas nuevas ramificaciones, de tal forma que se asegure la secuencialidad, es decir, que los nuevos conjuntos de arcos que se añaden a R_1 y R_2 formen rutas parciales.

Por ejemplo, si se ha elegido $S_1 = \{\varnothing\}$ y $S_2 = \{7 - 3 - 2 - 5 - 1\}$ el arco (5, 1) ha de entrar antes que el (2, 5), (3, 2),..., en las siguientes ramificaciones. Es decir, el primer arco que va a salir de las rutas parciales de las siguientes ramas es (7,3); el segundo es (3,2) y así sucesivamente, de tal forma que se mantenga la secuencialidad de los arcos que forman la ruta parcial en cada rama. Si S_1 no fuera vacío para elegir cual de los arcos va a salir primero, el de S_1 o el de S_2, se calcula el coste de las seudorutas óptimas que se obtendrían sin cada uno de esos arcos, para valorar la importancia o el peso de esos arcos en la seudoruta obtenida; evidentemente cuanto más coste tenga la seudoruta correspondiente más tarde deberá salir el arco.

Sean las siguientes variables:

ordensalida : vector de arcos donde se guarda el orden de salida de los arcos;

m1 y *m2* : número arcos de S_1 y S_2 respectivamente;

valorS1 y *valorS2*: variables lógicas que señalan si se ha valorado en S_1 y S_2, respectivamente, el arco seleccionado para salir a continuación;

costeS1 y *costeS2*: costes de las seudorutas óptimas sin el último arco considerado en S_1 y S_2, respectivamente;

el siguiente algoritmo formaliza la idea anterior para establecer el orden de salida de los arcos que se han seleccionado:

Procedimiento ORDENSALIDA

Paso 1.- Poner k = 1 , k1 = *m1*, k2 = *m2*, *costeS1* = FALSE; *costeS2* = FALSE.

Paso 2.- Si k1 = 0 ó k2 = 0 ir al paso 5;

en caso contrario:

si *valorS1* es FALSE hallar el coste de la seudoruta óptima suponiendo infinito la distancia del arco de S_1 que va a salir a continuación; poner valorS1 = TRUE;

si valorS2 es FALSE hallar el coste de la seudoruta óptima suponiendo infinito la distancia del arco de S_2 que va a salir a continuación; poner *valorS2* = TRUE.

Paso 3.- Si *costeS1* < costeS2:

definir *ordensalida(k)* = siguiente arco de S_2 para salir,

poner k2 = k2-1, *valorS2* = FALSE;

en caso contrario:

definir *ordensalida(k)* = siguiente arco de S_1 para salir,

poner k1 = k1-1, *valorS1* = FALSE.

Paso 4.- Poner k = k+1 e ir al paso 2.

Paso 5.- Si k1 > 0, repetir el siguiente proceso hasta que k1 = 0:

Poner *ordensalida*(k) = siguiente arco de S_1 para salir; k = k+1; k1 = k1-1.

Paso 6.- Si k2 > 0, repetir el siguiente proceso hasta que k2 = 0:

poner *ordensalida*(k) = siguiente arco de S_2 para salir; k = k+1; k2 = k2-1.

Aplicando este procedimiento al ejemplo propuesto en el apartado 2.3.1., el orden de salida obtenido fue el siguiente:

ordensalida(1) = (7,3);

ordensalida(2) = (3,2);

ordensalida(3) = (2,5);

ordensalida(4) = (5,1);

Por tanto las nuevas ramas que se obtienen, en este proceso de selección y ordenación, a partir de α, dan lugar a los siguientes vértices de soluciones con las siguientes rutas parciales R_1 y R_2, y conjuntos F de arcos prohibidos:

$$\alpha' : R_1(\alpha') = \{1\}, R_2(\alpha') = \{7-3-2-5-1\}, F(\alpha') = \varnothing;$$

$$\alpha'' : R_1(\alpha'') = \{1\}, R_2(\alpha'') = \{3-2-5-1\}, F(\alpha'') = \{(7,3)\};$$

$$\alpha''' : R_1(\alpha''') = \{1\}, R_2(\alpha''') = \{2-5-1\}, F(\alpha''') = \{(3,2)\};$$

$$\alpha^{iv} : R_1(\alpha^{iv}) = \{1\}, R_2(\alpha^{iv}) = \{5-1\}, F(\alpha^{iv}) = \{(2,5)\};$$

$$\alpha^{v} : R_1(\alpha^{v}) = \{1\}, R_2(\alpha^{v}) = \{1\}, F(\alpha^{v}) = \{(5,1)\};$$

2.4.- DESCRIPCION DEL ALGORITMO COMPLETO EN SEUDOCODIGO

El algoritmo es un proceso recursivo de exploración y ramificación. En cada iteración se explora un determinado vértice del árbol de búsqueda y se calcula la seudoruta correspondiente. El coste de esta seudoruta es una cota inferior del coste de la ruta óptima que se obtendría para el conjunto de soluciones correspondientes a ese vértice. Si esa cota inferior es mayor que el coste de la solución obtenida se corta la exploración de esa rama, y se prueba con otras; en caso contrario, a partir de la seudoruta, se seleccionan los conjuntos de arcos que se van a añadir a los conjuntos de soluciones, y se procede a la ramificación. En el momento en que $R(\alpha) = R_1(\alpha) \cup R_2(\alpha)$, contenga n arcos es decir, cuando $R(\alpha)$ sea una ruta, se calcula el coste de esta ruta y se registra si mejora la mejor solución obtenida hasta ese momento.

A este proceso recursivo le llamaremos RECURSIONGENERAL, y tiene como parámetros de entrada $R_1(\alpha)$, $R_2(\alpha)$ y $F(\alpha)$. Para su descripción se definen las siguientes variables:

ruta1 : Conjunto de arcos que constituyen la mejor ruta obtenida hasta ese momento;

costemínimo1 : Coste de la mejor ruta obtenida hasta ese momento.

Procedimiento RECURSIONGENERAL($R_1(\alpha)$, $R_2(\alpha)$, $F(\alpha)$):

Paso 1.- Chequear la factibilidad de la ruta parcial formada por $R_1(\alpha)$ y $R_2(\alpha)$,:
Si no es factible parar; si es factible
Poner n1 = cardinal ($R_1(\alpha)$) + cardinal ($R_2(\alpha)$);
si n1 < n, ir al paso 2, en caso contrario ir al paso 6.

Paso 2.- Ejecutar CALCULOSEUDORUTA y obtener la seudoruta correspondiente;
poner *w1* = coste de esta seudoruta:
si *w1* < *costeminimo1* ir al paso 3, en caso contrario ir al paso 7.

Paso 3.- Si los nodos de $SR(\alpha)$ no están repetidos:
poner *costeruta* = *w1*, *ruta1* = *seudoruta*, ir al paso 7.

Paso 4.- Ejecutar SELECCIONARCOS y ORDENSALIDA;
poner *numarcos* = cardinal(S_1) + cardinal(S_2).

Paso 5.- Realizar para i = 0 hasta *numarcos* el siguiente proceso:
Poner $R_1(\alpha^*) = (R_1(\alpha) \cup S_1) - (\cup_{k=1}^{i}$ *ordensalida*$(k))$;
poner $R_2(\alpha^*) = (R_2(\alpha) \cup S_2)) - (\cup_{k=1}^{i}$ *ordensalida*$(k))$;
$F(\alpha^*) = F(\alpha)$ *U ordensalida*(i);
Efectuar el procedimiento RECURSIONGENERAL($R_1(\alpha)$,$R_2(\alpha)$,$F(\alpha)$).
(Se considera ordensalida(0) = $\varnothing$).
Ir al paso 7.

Paso 6.- Poner *costeruta* = Coste de la ruta formada por $R_1(\alpha)$ y $R_2(\alpha)$;

Si *costeruta* < *costeminimo1* entonces:

Poner *costeminimo1* = *costeruta*;

ruta1 = Ruta formada por $R_1(\alpha)$ y $R_2(\alpha)$.

Paso 7.- Parar.

Integrando este procedimiento en el algoritmo principal, éste tendría los siguientes pasos:

Procedimiento PRINCIPAL

Paso 1.- Leer los datos siguientes:

número de nodos *n*; ventanas de tiempo $[a_i, b_i]$ i = 1,...,n;

matriz de distancias (d_{ij}) y matriz de tiempos (t_{ij}).

Paso 2.- Inicializar las variables:

costeminimo1 = ∞; $R_1(\alpha) = R_2(\alpha) = \{1\}$, $R(\alpha) = \varnothing$.

Paso 3.- Ejectutar el procedimiento RECURSIONGENERAL($R_1(\alpha)$,$R_2(\alpha)$,$F(\alpha)$).

Paso 4.- Escribir los resultados registrados: ruta1 y costeminimo1.

Paso 5.- Parar.

Ejemplo 4. 2.-

Sea el TSPTW definido por:

n = 7, **a** = (**0**), b = (60, 30, 50, 30, 50, 30, 50), **T** = **D** definida de la siguiente forma

-	1	2	3	4	5	6	7
1	inf.	20	23	15	20	22	15
2	9	inf.	3	7	0	13	7
3	6	9	inf.	4	9	11	4
4	2	5	8	inf.	5	7	0
5	16	15	10	14	inf.	13	14
6	11	2	5	9	2	inf.	9
7	2	16	19	11	16	18	inf.

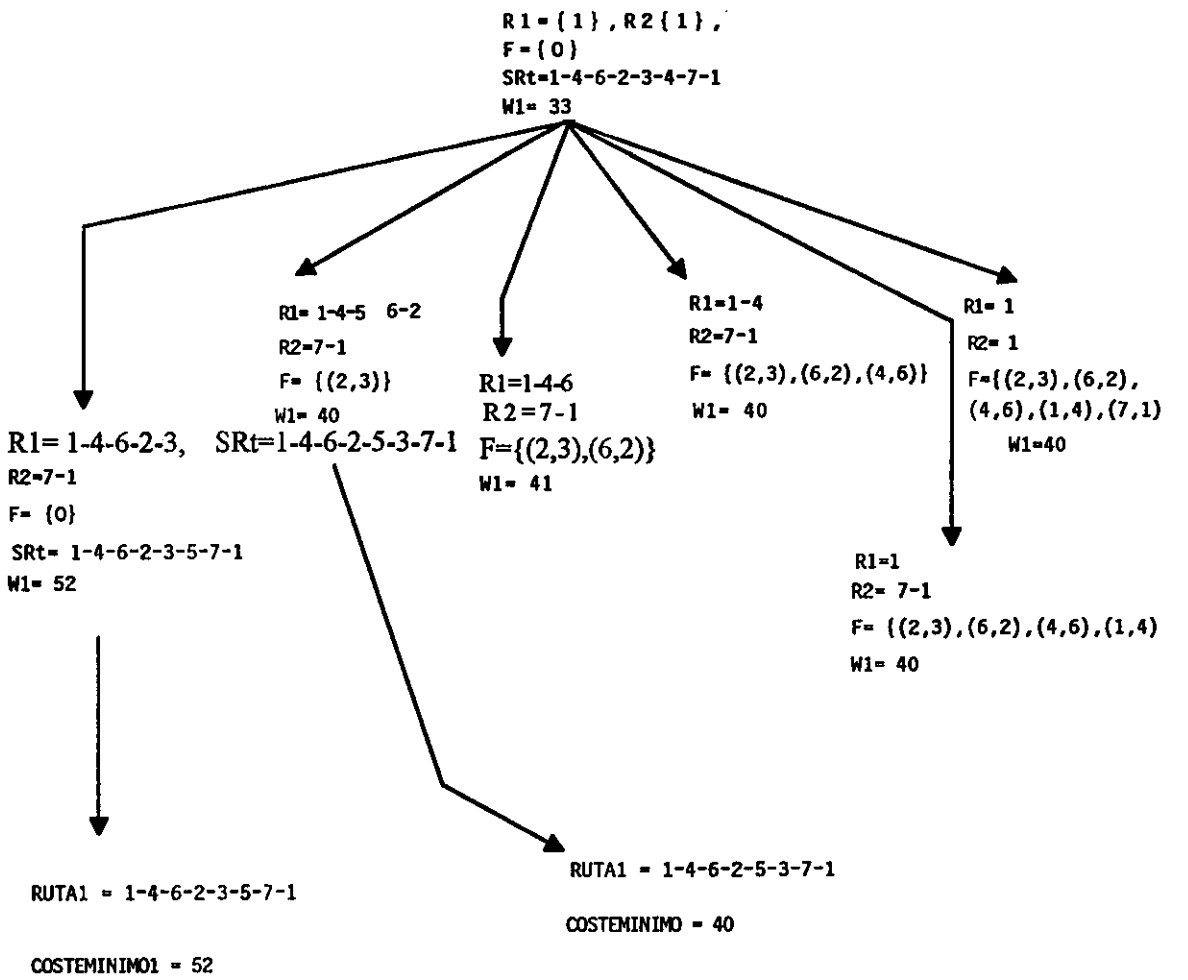


Figura 25. Gráfica del árbol de soluciones utilizando el algoritmo descrito. En cada vértice se señalan los conjuntos $R_1(\alpha)$, (R1), $R_2(\alpha)$, (R2) y $F(\alpha)$, (F); el coste de la seudoruta obtenida, W1, y dicha seudoruta cuando continúa la exploración.

2.5.- MEJORA EN EL CALCULO DE LAS SEUDORUTAS MEDIANTE METODOS DE PENALIZACION

En este apartado se aplican de los métodos de penalización lagrangiana al cálculo de las seudorutas en cada vértice del árbol de búsqueda de las soluciones, con el fin de conseguir cotas inferiores *más ajustadas* al valor del óptimo en ese vértice y seudorutas con un menor número de nodos repetidos. Se pretende seleccionar más arcos en las siguientes ramificaciones y llegar a la ruta óptima con menor número de exploraciones.

Para describir estos métodos de penalización consideramos el problema que se planteó en el apartado 2.2.1.: Calcular la ruta más corta que, saliendo del nodo i_f recorre todos los nodos de X exactamente una vez, hasta llegar al nodo i_r, donde i_f es el último nodo que aparece en $R_1(\alpha)$, i_r es el primer nodo que aparece en $R_2(\alpha)$, y X el conjunto de nodos que no aparecen en $R(\alpha)$.

El problema relajado correspondiente es calcular el camino más corto que partiendo de i_f llega a i_r visitando un número total de |X| veces algunos de los nodos de X, sin pasar dos veces consecutivas por el mismo.

Se verifica el siguiente resultado:

Lema 4. 1.-

Sea $(\pi_j)_{j\in X}$, donde $\pi_j \in R$; si R* es una ruta óptima del problema definido en el apartado 2.2.1. para la matriz de distancias (d_{ij}), entonces también es óptima para la matriz $(d_{ij}+\pi_i+\pi_j)$.

Demostración.-

Sea R una ruta factible, su distancia total con respecto a (d_{ij}) es $\sum_{(i,j)\in R} d_{ij}$ y con respecto a $(d_{ij}+\pi_i+\pi_j)$ es $\sum_{(i,j)\in R}(d_{ij}+\pi_i+\pi_j)=\sum_{(i,j)\in R} d_{ij}+2\cdot\sum_{j\in S}\pi_j$, ya que cada nodo tiene exactamente dos arcos incidentes.

Se observa que la factibilidad con respecto a las ventanas de tiempo no se ve tampoco alterada, ya que la matriz de tiempos no queda afectada.-

Por tanto, la transformación de (d_{ij}) a $(d_{ij}+\pi_i+\pi_j)$ no cambia la ruta óptima del problema; sin embargo si puede cambiar el cálculo del tramo $SR(\alpha)$ en el problema relajado y la cota inferior obtenida a partir de este cálculo. Así, una buena estrategia es elegir un vector π de forma que esta cota inferior sea lo mayor posible.

Más concretamente; sea $S^*(\pi)$ una solución óptima del problema relajado con respecto a la matriz de distancias $(d_{ij}+\pi_i+\pi_j)$ y R^* una ruta óptima del problema original, que no depende de π. Entonces con respecto a $(d_{ij}+\pi_i+\pi_j)$, el coste de $S^*(\pi)$ es siempre menor o igual que el de R^*, por consiguiente:

$$\sum_{(i,j)\in S^*(\pi)}(d_{ij}+\pi_i+\pi_j)\le\sum_{(i,j)\in R^*}(d_{ij}+\pi_i+\pi_j),$$

$$\sum_{(i,j)\in S^*(\pi)}d_{ij}+\sum_{i\in X}n_i\cdot\pi_i\le\sum_{(i,j)\in R^*}d_{ij}+2\cdot\sum_{i\in X}\pi_i,$$

$$\sum_{(i,j)\in S^*(\pi)}d_{ij}+\sum_{i\in X}(n_i-2)\cdot\pi_i\le\sum_{(i,j)\in R^*}d_{ij};$$

donde

n_i = número de arcos de $S^*(\pi)$ incidentes en el nodo i:

Si se define $v_i = n_i - 2$, $\mathbf{v}_\pi = (v_i)$, y $f(\pi)=\sum_{(i,j)\in S^*(\pi)}d_{ij}+\mathbf{v}_\pi\cdot\pi$, se tiene que $f(\pi)$, $\forall\pi$, es una cota inferior del óptimo del problema anterior, y mejor aún $\max_\pi f(\pi)$. Para el cálculo de $SR(\alpha)$, se propone un proceso iterativo que se aproxime a este máximo iterativamente $\pi^{m+1}=\pi^m+t\cdot\mathbf{v}_{\pi^m}$, siendo t una constante de penalización arbitraria positiva.

Sean:

ws1 : máximo valor de $f(\pi)$

π^1 : vector de penalizaciones correspondientes al mayor $f(\pi)$ hallado hasta el momento;

π : vector de penalizaciones de partida:

π^2 : vector de penalizaciones que se tiene en cada momento:

niter : número iteraciones en las que no ha habido mejora en el cálculo de $f(\pi)$;

maximoiter : Número máximo de iteraciones sin mejora permitido, a partir del cual se para el proceso;

Procedimiento CALCULOSEUDORUTAMEJORADO

Paso 1.- Poner $\pi^2 = \pi$, *niter* = 0, y *ws1* = $-\infty$.

Paso 2.- Poner $d_{ij} = (d_{ij} + \pi_i^2 + \pi_j^2)$, min1 = $\min_{(i,j)} d_{ij}$, $d_{ij} = d_{ij}$ - min1,

ejecutar CALCULOSEUDORUTA, poner $f(\pi^2) = f(\pi^2)$ - (n-n1)min1.

Si $f(\pi^2)$ + coste de los arcos de $R_1(\alpha)$ y $R_2(\alpha)$ > costeminimo1

entonces:

poner $\pi^1 = \pi^2$, y *ws1* = $f(\pi^2)$, ir al paso 7;

en otro caso ir al paso 3.

Paso 3.- Si en $S^*(\pi^2)$ no aparecen nodos repetidos:

poner $\pi^1 = \pi^2$, *ws1* = $f(\pi^2)$ ir al paso 7;

en otro caso ir al paso 4.

Paso 4.- Si $f(\pi^2)$ > *ws1* entonces:

poner *ws1* = $f(\pi^2)$, $\pi^1 = \pi^2$, *niter* = 0;

en otro caso poner *niter* = *niter*+1.

Paso 5.- Calcular v_{π^2}, poner $\pi^2 = \pi^2 + t \cdot v_{\pi^2}$

Paso 6.- Si *niter* < *maximoiter* ir al paso 2; si no ir al paso 7.

Paso 7.- Poner $SR(\alpha) = S^*(\pi^1)$, restaurar los valores originales de d_{ij}, parar.

El cálculo de las $S^*(\pi^2)$ se realiza utilizando la adaptación del algoritmo de etiquetado con la matriz $(d_{ij} + \pi_i^2 + \pi_j^2)$ en vez de (d_{ij}), y registrando en el último paso los arcos de la solución obtenida en $S^*(\pi^2)$. El objetivo de restar *min1* es evitar que las matrices de distancia resultantes tengan valores negativos.

Este proceso de cálculo de los vectores π se incorpora al algoritmo general. El vector inicial π va a formar parte de los parámetros de entrada de cada vértice α. (A partir de ahora se denotará por $\pi(\alpha)$). Además del proceso descrito para el cálculo del camino $SR(\alpha)$ se realizan cambios en los procedimientos RECURSIONGENERAL y PRINCIPAL.

Cambios en el Procedimiento RECURSIONGENERAL($R_1(\alpha)$, $R_2(\alpha)$, $F(\alpha)$, $\pi(\alpha)$):

Paso 2.- Poner $\pi = \pi(\alpha)$ y ejecutar CALCULOSEUDORUTAMEJORADO;
poner w1 = coste del camino formado por $R_1(\alpha)$, $R_2(\alpha)$ + *ws1*;
si w1 < *costeminimo1* ir al paso 3; sino ir al paso 8.

Paso 5.- Realizar para i = 0 hasta *numarcos* el siguiente proceso:
Hacer:
$R_1(\alpha^*) = (R_1(\alpha) \cup S_1) - (\cup_{k=1}^{i}$ *ordensalida*$(k))$,
$R_2(\alpha^*) = (R_2(\alpha) \cup S_2) - (\cup_{k=1}^{i}$ *ordensalida*$(k))$,
$F(\alpha^*) = F(\alpha) \cup$ *ordensalida*(i),
$\pi(\alpha^*) = \pi^1$;
Ejecutar RECURSIONGENERAL($R_1(\alpha^*)$, $R_2(\alpha^*)$, $F(\alpha^*)$, $\pi(\alpha^*)$).

Cambios en el Procedimiento PRINCIPAL:

Paso 2.- Inicializar las variables:

$costeminimo1 = \infty$, $R_1(\alpha) = R_2(\alpha) = \{1\}$, $F(\alpha) = \varnothing$ y $\pi(\alpha) = \mathbf{0}$.

Paso 3.- Ejectutar el procedimiento RECURSIONGENERAL($R_1(\alpha)$, $R_2(\alpha)$, $F(\alpha)$, $\pi(\alpha)$).

En el ejemplo 4.1. en el origen es $R_1(\alpha) = R_2(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, y $\pi(\alpha) = \mathbf{0}$. Para t = 0.5 se tiene la siguiente secuencia de iteraciones:

	Seudoruta	**Vector de Penalizaciones π**	**c.inf. f**
Iteración 1	1-7-3-7-3-2-5-1	(0,0,0,0,0,0,0)	30
Iteración 2	1-7-3-2-4-2-5-1	(0,0,1.0,-1.0,0,-1.0,1.0)	37
Iteración 3	1-7-3-7-3-2-5-1	(0,1.0,1.0,-1.0,0,-2.0,1.0)	40
Iteración 4	1-7-3-2-4-2-5-1	(0,1.0,2.0,-2.0,0,-3.0,2.0)	43
Iteración 5	1-7-3-2-4-2-5-1	(0,2.0,2.0,-2.0,0,-4.0,2.0)	47
Iteración 6	1-7-3-2-4-2-5-1	(0,3.0,2.0,-2.0,0,-5.0,2.0)	51
Iteración 7	1-7-4-6-4-2-5-1	(0,4.0,2.0,-2.0,0,-6.0,2.0)	54
Iteración 8	1-7-3-7-3-2-5-1	(0,4.0,1.0,-1.0,0,-6.0,2.0)	50
Iteración 9	1-7-4-6-4-2-5-1	(0,4.0,2.0,-2.0,0,-7.0,3.0)	54
Iteración 10	1-7-3-7-3-2-5-1	(0,4.0,1.0,-1.0,0,-7.0,3.0)	54
Iteración 11	1-7-4-6-4-2-5-1	(0,4.0,2.0,-2.0,0,-8.0,4.0)	54
Iteración 12	1-7-4-6-3-2-5-1	(0,4.0,1.0,-1.0,0,-8.0,4.0)	55

En este punto termina al proceso, ya que el $S^*(\pi)$ encontrado no va a tener ningún nodo de X repetido; por tanto $S^*(\pi) = R^*$ y la seudoruta encontrada SR = 1 - 7 - 4 - 6 - 3 - 2 - 5 - 1 es una solución óptima para ese vértice. Como el vértice α que se estaba analizando era el origen del árbol de búsqueda, el algoritmo termina aquí, con SR como solución y con un coste óptimo de 55.

3.- UN ALGORITMO EXACTO PARA EL VRPTW

En este apartado se describe un algoritmo exacto para el VRPTW consistente básicamente en la adaptación del algoritmo para el TSPTW descrito en el apartado 2. Sean m: número de vehículos, (q_i), i=2,...,n, vector de demandas de los clientes, y capac la capacidad total que puede transportar cada vehículo.

Para adaptar el algoritmo anterior al VRPTW, se incorpora un conjunto de m-1 nodos ficticios: n+1, n+2, n+m-1 que representan la llegada al depósito central de cada vehículo y la salida del siguiente, de tal forma que el conjunto de arcos entre cada dos nodos ficticios consecutivos, (o entre el nodo 1 y un nodo ficticio), representa una ruta. Estos nodos ficticios, deben estar situados en la misma posición que el nodo 1 y ha de asegurarse que haya algún nodo entre cada dos nodos ficticios consecutivos, para que no haya rutas vacías. Por tanto, las matrices de distancias y tiempo se amplían de la siguiente forma:

$d_{ij} = d_{1j}$, para $i = n+1, n+2,..., n+m-1; \quad j = 2, 3,..., n;$
$d_{ji} = d_{j1}$, para $i = n+1, n+2,..., n+m-1; \quad j = 2, 3,..., n;$
$d_{ij} = \infty$, para $i, j = n+1, n+2,..., n+m-1;$
$d_{1i} = d_{i1} = \infty$, para $i = n+1, n+2, ..., n+m-1;$

análogamente para la matriz de tiempos:

$t_{ij} = t_{1j}$, para $i = n+1, n+2,..., n+m-1; \quad j = 2, 3,..., n;$
$t_{ji} = t_{j1}$, para $i = n+1, n+2,..., n+m-1; \quad j = 2, 3,..., n;$
$t_{ij} = \infty$, para $i, j = n+1, n+2,..., n+m-1;$
$t_{1i} = t_{i1} = \infty$, para $i = n+1, n+2, ..., n+m-1.$

Así mismo:

$a_i = a_1, \quad b_i = b_1, \quad$ y $\quad q_i = 0$, para $i = n+1, n+2,..., n+m-1.$

3.1.- ADAPTACION DEL PROCESO GENERAL DE ACOTACION Y RAMIFICACION

Considérese ahora, además del conjunto de nodos que hay que visitar, los nodos ficticios y las matrices de tiempos y distancias ampliadas. Cada vértice α del árbol de búsqueda va a tener asociado el conjunto de arcos que aparecen en el conjunto de soluciones asociadas a este vértice, y el conjunto de arcos *prohibidos* $F(\alpha)$ para formar parte en las soluciones de este vértice. Opcionalmente, se puede incorporar en cada vértice el vector $\pi(\alpha)$ de penalizaciones iniciales.

Sin embargo, en este caso el conjunto $R(\alpha)$ sólo va a estar formado por los arcos que se van añadiendo de forma secuencial al origen; es decir $R(\alpha) = R_1(\alpha)$, o bien $R_2(\alpha) = \{1\}$ en todos los vértices. El propósito de esta nueva definición del conjunto $R(\alpha)$ es controlar mejor la factibilidad de las rutas que se van formando con respecto a las demandas y capacidades.

En la exploración de cada vértice se calcula una seudoruta que proporciona el conjunto de arcos a añadir a $R(\alpha)$ y $F(\alpha)$ en las nuevas ramificaciones, de forma análoga a como se hace en el algoritmo para el TSPTW. Es decir, sea $R(\alpha) = 1 - i_1 - i_2 - \ldots - i_f$, los arcos que se seleccionan serán de la forma (i_f, i_{f+1}), $(i_{f+1}, i_{f+2}), \ldots, (i_{r-2}, i_{r-1})$, (i_{r-1}, i_r), y dan lugar a las siguientes ramificaciones:

$$R(\alpha') = R(\alpha) \cup \{(i_f, i_{f+1}), (i_{f+1}, i_{f+2}), \ldots, (i_{r-2}, i_{r-1}), (i_{r-1}, i_r)\}, \text{ y } F(\alpha') = F(\alpha);$$

$$R(\alpha'') = R(\alpha) \cup \{(i_f, i_{f+1}), (i_{f+1}, i_{f+2}), \ldots, (i_{r-2}, i_{r-1})\}, \text{ y } F(\alpha'') = F(\alpha) \cup \{(i_{r-1}, i_r)\};$$

.....

.....

$$R(\alpha''') = R(\alpha) \cup \{(i_f, i_{f+1})\}, \text{ y } F(\alpha''') = F(\alpha) \cup \{(i_{f+1}, i_{f+2})\};$$

$$R(\alpha'''') = R(\alpha), \text{ y } F(\alpha'''') = F(\alpha) \cup \{(i_f, i_{f+1})\}.$$

Para todas estas ramificaciones $\pi(\alpha') = \pi(\alpha'') = \ldots = \pi(\alpha''') = \pi(\alpha'''') = \pi^1$, siendo π^1 vector de penalizaciones correspondiente a la seudoruta obtenida en ese vértice.

Cuando $(i,j) \in F(\alpha)$ y uno de los nodos i o j es ficticio o es el nodo inicial 1 (supóngase que este es el nodo i), para impedir en ese conjunto de soluciones el arco que une el nodo origen con el nodo j, se debe considerar que en el conjunto $F(\alpha)$ están incluidos implícitamente los arcos (1,j), (n+1,j), (n+2,j),..., (n+m-1,j). En el caso de ser j el nodo ficticio, en el conjunto $F(\alpha)$ también están los arcos (i,1), (i,n+1), (i,n+2),..., (i,n+m-1) aunque no aparezcan de forma explícita.

3.2.- ADAPTACION DEL CALCULO DE LA SEUDORUTA EN CADA VERTICE

El proceso de exploración también se basa el cálculo de las seudorutas en cada vértice. La seudoruta va a estar formada por los arcos de $R(\alpha)$ más un conjunto de arcos de la forma i_f-j_1-j_2 -...-j_K-1, donde j_1, j_2, ..., j_K son nodos que no aparecen en $R(\alpha)$, de tal forma que en total haya n arcos. Ha de tenerse en cuenta que cada ruta formada ha de ser factible con respecto a las demandas y las capacidades. Además las capacidades de los vehículos de las rutas que no se han formado deben ser mayores que las demandas de los nodos que quedan por visitar en esas rutas.

Para ello se plantea el problema siguiente: Hallar las m rutas de menor distancia total que visitan los n clientes comenzando y finalizando en 1, recorriendo obligatoriamente los arcos de $R(\alpha)$, y respetando las restricciones de ventanas de tiempo y capacidad. Para determinar el conjunto de arcos que hay que añadir a los de $R(\alpha)$, sea:

X = Conjunto de nodos, ficticios y no ficticios, que no aparecen en $R(\alpha)$;
T_{i_f} = Tiempo de salida del nodo i_f después de recorrer los arcos de $R(\alpha)$, si i_f es ficticio $T_{i_f} = a_1$;

3.2.1.- PLANTEAMIENTO MEDIANTE PROGRAMACION DINAMICA

$\forall S \subseteq X$, y $j \in S$, se definen:

f(S,j) = Distancia del camino más corto que inicialmente recorre los arcos de $R(\alpha)$, a continuación los nodos de S y finaliza en j, de forma que se respeten las ventanas de tiempo y las capacidades de los vehículos;

u(S,j) = Tiempo de llegada a j en ese camino mínimo;

nrut(S,j) = Número de rutas completas (o visitas a nodos ficticios) en ese camino mínimo;

qtot(S,j) = Demanda total acumulada de los clientes visitados en ese camino mínimo;

qpar(S,j) = Demanda total del conjunto de nodos visitados tras la última llegada a un nodo ficticio.

La recursión queda de la siguiente forma:

$$f(S,j) = \min_{i \in S-\{j\}} \{f(S-\{j\}, i) + d_{ij}\},$$

sea i(j) el índice correspondiente a este mínimo

$$u(S,j) = u(S-\{j\}, i(j)) + t_{i(j)j};$$

$$qtot(S,j) = qtot(S-\{j\}, i(j)) + q_j;$$

$$qpar(S,j) = qpar(S-\{j\}, i(j)) + q_j;$$

nrut(S,j) = nrut(S-{j},i(j)), si j = 1, 2,..., n;

nrut(S,j) = nrut(S-{j},i(j))+1, si j = n+1,..., n+m-1;

redefiniéndose:

$u(S,j) = a_j$, si $u(S,j) < a_j$;

$f(S,j) = \infty$, si $u(S,j) > b_j$;

$u(S,j) = a_0$, si j = n+1,...,n+m-1;

qpar(S,j) = 0, si j = n+1,...,n+m-1;

$f(S,j) = \infty$, si qpar(S,j) > capac;

$f(S,j) = \infty$, si qtotal- (qtot(S,j) - qpar(S,j)) > (m-nrut(S,j))·capac;

siendo capac la capacidad de cada vehículo, y qtotal la suma de las demandas de todos los clientes.

Valores iniciales:

$f(\varnothing, i_f)$ =Coste de $R(\alpha)$;
$u(\varnothing, i_f) = T_{i_f}$;
$nrut(\varnothing, i_f)$ = Número de visitas a nodos ficticios en $R(\alpha)$;
$qtot(\varnothing, i_f)$ = Demanda total de los nodos visitados en $R(\alpha)$;
$qacum(\varnothing, i_f)$ = Demanda de los nodos visitados en $R(\alpha)$ desde la última llegada a un nodo ficticio.

El problema, así definido, trata de determinar el $\min_{j \in X} \{f(X,j) + d_{j1}\}$.

3.2.2.- RELAJACION DEL ESPACIO DE ESTADOS

Aplicando la relajación basada en el cardinal k = |S|, la recursión anterior queda de la siguiente forma:

$$f(k,j) = \min_{i \neq j} \{f(k-1,i) + d_{ij}\};$$

sea i(j) el índice correspondiente a este mínimo

$u(k,j) = u(k-\{j\}, i(j)) + t_{i(j)j}$;
$qtot(k,j) = qtot(k-1, i(j)) + q_j$;
$qpar(k,j) = qpar(k-1, i(j)) + q_j$;
nrut(k,j) = nrut(k-1,i(j)), si j = 1, 2,..., n;
nrut(k,j) = nrut(k-1,i(j))+1, si j = n+1,..., n+m-1;

para k = 1, 2,..., K; siendo K = |X|;

redefiniéndose además

$u(k,j) = a_j$, si $u(k,j) < a_j$;
$f(k,j) = \infty$, si $u(k,j) > b_j$;
$u(k,j) = a_0$, si $j = n+1,...,n+m-1$;
$qpar(k,j) = 0$, si $j = n+1,...,n+m-1$;
$f(k,j) = \infty$, si $qpar(k,j) > capac$;
$f(k,j) = \infty$, si $qtotalrel - (qtot(k,j) - qpar(k,j)) > (m-nrut(k,j))\cdot capac$;

siendo *qtotalrel* la menor suma que se puede formar con las demandas correspondientes a los clientes que aparecen en los diferentes caminos desde $(0,i_f)$ hacia (K+1,1).en el espacio de estados relajados.

Valores iniciales:

$f(0,i_f)$ = Coste de $R(\alpha)$;
$u(0,i_f) = T_{i_f}$;
$nrut(0,i_f)$ = Número de visitas a nodos ficticios en $R(\alpha)$;
$qtot(0,i_f)$ = Demanda total de los nodos visitados en $R(\alpha)$;
$qpar(0,i_f)$ = Demanda de los nodos visitados en $R(\alpha)$ desde la última llegada a un nodo ficticio.

El problema, así definido, trata de determinar el $\min_{j \in X}\{f(K,j) + d_{j1}\}$. El coste del camino resultante proporciona una cota inferior al valor del problema anterior, usando un tiempo polinomial en el cardinal de X. Análogamente al caso del TSPTW se puede mejorar el cálculo de la cota inferior si se utiliza el vector de penalizaciones $\pi(\alpha)$, con la nueva matriz de distancias $d'_{ij} = d'_{ij} + \pi_i + \pi_j$.

3.2.3.- ADAPTACION DEL ALGORITMO DE ETIQUETADO

Considérese la red constituida por los nodos de la forma (k,j), para k=1,..,K, y $j \in X$, correspondientes a los estados relajados de la formulación anterior, junto con el nodo o vértice

inicial $(0,i_f)$ y el final (K+1,1). Los arcos y sus longitudes vienen definidos por el problema recursivo anterior.

En su resolución se utiliza una nueva adaptación del algoritmo de Djikstra que halla el coste mínimo entre los vértices $(0,i_f)$ y (K+1,1). Para ello se definen las siguientes constantes:

numrutas = Número de rutas completas formadas en $R(\alpha)$;
qtotalini = Demanda total de los nodos visitados en $R(\alpha)$;
qparini = Demanda total de los nodos visitados en $R(\alpha)$ desde la última llegada a un nodo ficticio;
$SR(\alpha)$ = Conjunto de arcos que se van a añadir a $R(\alpha)$ para formar la seudoruta;

además de las etiquetas definidas para el algoritmo del TSPTW, en la adaptación se definen las siguientes:

qt(k,j): demanda total de la segunda componente de los vértices de la red visitados en el camino mínimo de $(0,i_f)$ a (k,j) hallado hasta ese momento;
qp(k,j): demanda total de la segunda componente de los vértices de la red visitados, desde la última llegada a un vértice cuya segunda componente era un nodo ficticio, en ese camino mínimo;
nr(k,j): número de visitas a vértices cuya segunda componente es un nodo ficticio;

El algoritmo para hallar la seudoruta queda de la forma siguiente:

Procedimiento CALCULOSEUDORUTA

Paso 1.- Inicializar las variables.
$\forall$ k = 1,..K, y $\forall j \in X$ hacer:
u(k,j) = ∞; *v(k,j)* = ∞; *qt(k,j)* = qtotalini; *qp(k,j)* = qparini;
nr(k,j) = numrutas; $\theta(k,j) = (0,i_f)$; *defin(k,j)* = FALSE,.

Paso 2.- Definir etiquetas iniciales.

$\forall j \in X$, hacer:

$u(1,j) = T_{if} + t_{ifj}$, $v(1,j) = d_{ifj}$ + Coste de $R(\alpha)$,

$qt(1,j) = qtotalini + q_j$;

$qp(1,j) = qparini + q_j$ si j = 2, 3,...,n;

$qp(1,j) = 0$ y $nr(1,j)$ = numrutas+1 si j = n+1, n+2,..., n+m-1:

Se redefine:

$u(1,j) = a_j$ *si* $u(1,j) < a_j$;

$v(1,j) = \infty$ si $u(1,j) > b_j$;

$v(1,j) = \infty$ si $qpar(1,j) >$ capac;

$v(1,j) = \infty$ si qtotalrel-$(qt(1,j)-qp(1,j))$>(m-$nr(1,j)$)·capac.

Paso 3.- Elegir el nodo que va a ser etiquetado definitivamente:

Sea $v(1,j^*) = \min\{v(1,j) / j \in X\}$. Poner *estado* = 1, *índice* = j*.

Poner *defin(estado,indice)* = TRUE.

Paso 4.- Cambiar etiquetas:

Poner i = *estado*+1;

Si i ≤ K, entonces $\forall j \in X - \{indice\}$:

poner v' = $v(estado,indice) + d_{indice,j}$, u' = $u(estado,indice) + t_{indice,j}$;

redefiniendo

u' = a_j si uw < a_j,

v' = ∞ si uw > b_i,

v' = ∞ si $qp(estado,indice)+q_j >$ capac;

v' = ∞ si (qtotalrel-$qt(i,j)$) > capac·(m-($nr(i,j)$+1)) si j = n+1, n+2,..., n+m-1;

si v' < $v(i,j)$ entonces poner $v(i,j)$ = v', $u(i,j)$ = u', $\theta(i,j)$ = (*estado,indice*).

Paso 5.- Elegir nuevo nodo para ser etiquetado definitivamente:

Sea $v(i^*,j^*) = \min\{v(i,j) / i = 1,..,K; j \in X; defin(i,j) = FALSE\}$,

poner *estado* = i*; *indice* = j*, *defin*(*estado,indice*) = TRUE.

Paso 6.- Verificar si es necesario etiquetar más nodos:

Sea i = K; si *estado(i,j)* = TRUE, ir al paso 7;
si $\exists j \in X$ / *defin(i,j)* = FALSE volver al paso 4.

Paso 7.- Elegir el camino más corto hasta el nodo (K+1,1):

Poner $v^*(j) = v(K,j) + d_{j1}$, redefiniendo $v^*(j) = \infty$ si $u(K,j) + t_{j1} > a_1$, $\forall j \in X$;
Sea $v^*(j^*) = \min\{v^*(j) / j \in X\}$; poner *estado* = K, *indice* = j*.

Paso 8.- Registrar dicho camino mínimo desde (0,1) hasta (K+1,i_1):

Inicializar $SR(\alpha) = \varnothing$,
poner (*indice*,1) $\in SR(\alpha)$;
Repetir el siguiente proceso:

estado1 = *estado*, *indice1* = *indice*,

(*estado*,*indice*) = θ(*estado1*,*indice1*),

(*indice*,*indice1*) $\in SR(\alpha)$,

hasta que (*estado*,*indice*) = (0,i_1).

Registrar $SR(\alpha)$, poner *wtotal* = coste del camino formado por $R(\alpha)$ y $SR(\alpha)$.

Parar.

Como en el algoritmo para el TSPTW, en cada rama se puede repetir este proceso de cálculo varias veces cambiando la matriz de distancias, $d'_{ij} = d_{ij} + \pi_i + \pi_j$, mediante un vector de penalizaciones π que va cambiando, a partir de un vector inicial $\pi(\alpha)$ en cada rama, como se explicó en el procedimiento CALCULOSEUDORUTAMEJORADO para el TSPTW.

En la ruta parcial $R(\alpha)$, la visita a los nodos ficticios representa siempre la finalización de una ruta completa de un vehículo, y por ello es indiferente cambiar cualquier nodo ficticio por otro. Por tanto, para evitar duplicar la exploración de ramas que aparentemente tengan distintas rutas parciales, es necesario establecer un criterio para ordenar los nodos ficticios que aparecen en la seudoruta obtenida en cada rama. Para ello se propone no parar en al paso 8, y añadir un paso nuevo al proceso anterior:

Paso 9.- Sea 1 - i_1 - ... - i_f - j_1 - j_2 - ... - j_t - 1 la seudoruta obtenida con los arcos de $R(\alpha)$ y $SR(\alpha)$. Cambiar en esta seudoruta el primer nodo ficticio que aparece por n+1; el siguiente nodo que aparece por n+2,... así sucesivamente hasta llegar al final; si antes de llegar al final ya se ha incorporado el nodo n+m-1, volver a repetir el proceso cambiando el siguiente nodo ficticio que se encuentre por n+1, ...;

Parar.

3.3.- SELECCION Y ORDENACION DE LOS ARCOS OBTENIDOS EN LA SEUDORUTA

El proceso de selección de los arcos hallados en $R(\alpha)$ para incorporarse a las rutas parciales en las siguientes ramificaciones es más sencillo que en el algoritmo para el TSPTW: solamente se seleccionan arcos de forma secuencial comenzando por i_f hasta que se repite algún nodo.

Procedimiento ORDENSALIDA

Paso 1.- Poner $S = \varnothing$.

Paso 2.- Añadir el arco (i_f, j_1) a S; poner $j^* = j_1$, $k = 1$;

Paso 3.- Si $k = K$, ir al paso 4, en caso contrario:

Repetir mientras $k < K$ y j_{k+1} no esté presente en ningún arco de S:

Poner $k = k+1$ y añadir (j^*, j_k) a S, poner $j^* = j_k$.

Paso 4.- Parar.

De esta forma se controla la factibilidad de las rutas parciales que se van obteniendo con respecto a la capacidad. Además, el orden de salida de los arcos elegidos en las siguientes ramificaciones es trivial.

Ejemplo 4. 3.-

Supóngase el vértice inicial, $R(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, $\pi(\alpha) = (0)$, en un problema con 8 clientes, con una demanda de 2 unidades cada uno y 4 vehículos con una capacidad de 4 unidades cada uno de ellos. El nodo 1 indica el depósito central, los nodos del 2 al 9 indican los clientes y se añaden 3 nodos ficticios 10, 11 y 12. Sea la siguiente seudoruta obtenida:

1 - 7 - 8 - 10 - 7 - 8 - 11 - 7 - 8 - 12 - 7 - 8 - 1;

y el vector π^1 correspondiente a esta seudoruta utilizando t = 0'5

$$\pi^1 = (0, 46, -18, -10, 122, 86, -110, -126, 10, 0, 0, 0);$$

los arcos seleccionados son los siguientes: S = {(1,7), (7,8) y (8,10)} y el orden de salida obviamente será: ordensalida(1) = (8,10), ordensalida(2) = (7,8), ordensalida(3) = (1,7). Por tanto los vértices correspondientes a la ramificación de α son:

$$\alpha' : R(\alpha') = \{1-7-8-10\}, F(\alpha') = \varnothing \text{ y } \pi(\alpha') = \pi^1;$$

$$\alpha'' : R(\alpha'') = \{1-7-8\}, F(\alpha'') = \{(8,10)\} \text{ y } \pi(\alpha'') = \pi^1;$$

$$\alpha''' : R(\alpha''') = \{1-7\}, F(\alpha''') = \{(7,8)\} \text{ y } \pi(\alpha''') = \pi^1;$$

$$\alpha'''' : R(\alpha'''') = \{1\}, F(\alpha'''') = \{(1,7)\} \text{ y } \pi(\alpha'''') = \pi^1.$$

3.4.- DESCRIPCION DEL ALGORITMO EN SEUDOCODIGO

Análogamente al algoritmo para el TSPTW, se trata de un procedimiento recursivo de exploración y ramificación de cada vértice α, que se va a denominar RECURSIONGENERAL, con los parámetros de entrada en este caso $R(\alpha)$, $F(\alpha)$ y $\pi(\alpha)$. Se definen las siguientes variables:

ruta1 : Conjunto de arcos que constituyen la mejor solución obtenida hasta ese momento;

costeminimo1 : Coste de la mejor solución obtenida hasta ese momento.

Procedimiento RECURSIONGENERAL($R(\alpha)$,$F(\alpha)$,$\pi(\alpha)$):

Paso 1.- Chequear la factibilidad de las rutas definidas por $R(\alpha)$:
Si no son factibles parar.
Poner n1 = cardinal de $R(\alpha)$;
si n1 < n+m-1, ir al paso 2, en caso contrario ir al paso 6.

Paso 2.- Ejecutar CALCULOSEUDORUTAMEJORADO y obtener la seudoruta correspondiente;
definir w1 como la cota inferior mínima asociada a esta seudoruta,
y π^1 como el vector de penalizaciones correspondientes;
si w1 < *costeminimo1* ir al paso 3, en caso contrario ir al paso 7.

Paso 3.- Si los nodos de $SR(\alpha)$ no están repetidos:
poner *costeminimo1* = w1, *ruta1* = seudoruta, ir al paso 7;
si no ir al paso 4.

Paso 4.- Hallar el conjunto de arcos S, a partir de $SR(\alpha)$, y el correspondiente vector *ordensalida*; poner *numarcos* = cardinal (S).

Paso 5.- Realizar para i = 0 hasta *numarcos* el siguiente proceso:

Poner $R(\alpha^*) = (R(\alpha) \cup S) - (\cup_{k=1}^{i} ordensalida(k))$
poner $F(\alpha^*) = F(\alpha) \cup ordensalida(i)$, $\pi(\alpha^*) = \pi^1$;

Ejecutar el procedimiento RECURSIONGENERAL($R(\alpha^*)$, $F(\alpha^*)$, $\pi(\alpha^*)$).

(Se considera ordensalida(0) = $\varnothing$).

Ir al paso 7.

Paso 6.- Poner *costeruta* = coste de los arcos de $R(\alpha)$;

Si *costeruta* < *costeminimo1* entonces:

Poner *costeminimo1* = *costeruta*; *ruta1* = $R(\alpha)$.

Paso 7.- Parar.

Integrando este procedimiento en el algoritmo principal éste tendría los siguientes pasos:

Procedimiento PRINCIPAL

Paso 1.- Leer los datos:

número de nodos n; número de vehículos m; ventanas de tiempo $[a_i, b_i]$ i = 1,...,n;

matriz de distancias (d_{ij}); matriz de tiempos (t_{ij}).

Paso 2.- Redefinir la matriz de distancias y tiempos:

$t_{ij} = t_{1j}$, $t_{ji} = t_{j1}$, $d_{ij} = d_{1j}$, $d_{ji} = d_{j1}$, para i = n+1,..., n+m-1, j = 1,..., n;
$t_{ij} = \infty$, $d_{ij} = \infty$, para i,j = 1, n+1,..., n+m-1.

Paso 3.- Inicializar las variables:

costeminimo1 = ∞; $R(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, $\pi(\alpha)=(0)$.

Paso 4.- Ejecutar el procedimiento RECURSIONGENERAL($R(\alpha)$,$F(\alpha)$,$\pi(\alpha)$).

Paso 5.- Escribir los resultados registrados: ruta1 y costeminimo1.

Ejemplo 4. 4.-

A continuación, se muestra un ejemplo para ilustrar el funcionamiento de este algoritmo para un problema con 4 clientes, cada uno de los cuales demanda 2 unidades de mercancía, la capacidad de cada vehículo es 6, y son 2 las rutas generadas. El número total de nodos será por tanto, 6. La matriz de distancias **D** es la siguiente

	1	2	3	4	5
1	inf.	49	190	248	286
2	49	inf.	144	278	237
3	190	144	inf	155	233
4	248	278	155	inf.	323
5	286	237	233	323	inf

así mismo se considera **T** = **D**, **a** = **(0)**, **b**= (960, 120, 420, 360, 720),

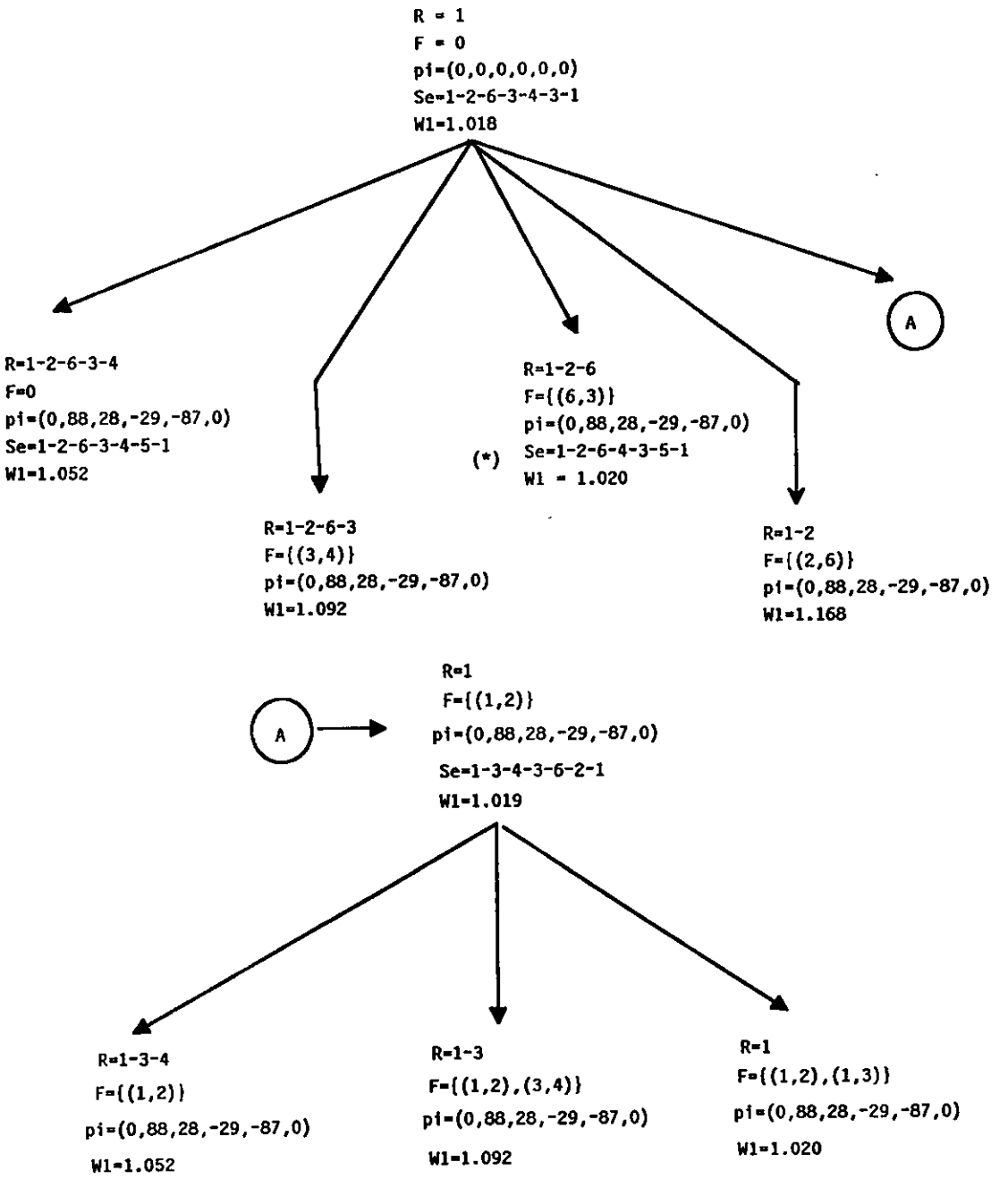


Figura 26. Gráfica del árbol de soluciones utilizando el algoritmo descrito. En cada vértice α se señalan los conjuntos R(α),(R), y F(α), (F); el vector de penalizaciones iniciales π(α); el coste de la seudoruta obtenida, W1, y dicha seudoruta, Se, cuando se continúe la exploración en esa rama. La solución óptima (*) se encuentra en la tercera rama.

3.5.- MODIFICACIONES EN EL ALGORITMO

Existen dos modificaciones que sin cambiar el carácter óptimo del algoritmo reducen en muchas ocasiones el tiempo de computación; estas son las siguientes:

1.- Utilizar como valor de entrada de la variable *ruta*1 la solución obtenida por algún algoritmo heurístico rápido, y como valor de la variable COSTEMINIMO1 el coste de esta solución, en vez de COSTEMINIMO1 = ∞.

2.- Cuando el número de nodos asignados a cada ruta no sea muy grande (no más de 8 ó 9 nodos), cada vez que se llega a una nueva solución en el algoritmo, antes de comprobar si mejora la solución obtenida hasta el momento, hallar la solución del TSPTW a cada conjunto de puntos asignados a las rutas de esa solución a la que se ha llegado.

A continuación, se explica como se incorporan estas modificaciones con más detalle.

3.5.1.- OBTENCION DE LA SOLUCION DE UN HEURISTICO COMO PUNTO DE PARTIDA

El uso de una solución de partida obtenida mediante un algoritmo heurístico pretende evitar la exploración de las ramas que den lugar a soluciones peores o igual que ésta al iniciar el algoritmo.

En este sentido dos algoritmos eficaces y rápidos, que dan soluciones en poco tiempo, son las adaptaciones al VRPTW, de los algoritmos de Fisher & Jaikumar, (1.981) y Beasley, (1.983). Una vez obtenida la solución heurística, si está cercana a la óptima, al obtenerse en el algoritmo exacto buenas cotas inferiores en las ramas de exploración se consigue evitar muchas exploraciones innecesarias y obtener el óptimo en menos tiempo.

La incorporación de este heurístico al procedimiento general se hace, sencillamente modificando el paso 3 del procedimiento PRINCIPAL de la forma siguiente:

MODIFICACION 1.-

Paso 3.- Ejecutar algoritmo heurístico;

Inicializar las variables:

definir *ruta*1 como la Solución obtenida por este algoritmo,

definir *Costeminimo* como el coste de *ruta*1,

$R(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, $\pi(\alpha) = (0)$.

3.5.2.- OPTIMIZACION DE LAS RUTAS AL LLEGAR A UNA SOLUCION

Cuando el algoritmo exacto encuentra una solución intermedia, no siempre ocurre que cada una de las rutas formadas sea óptima; para asegurar esto, a cada uno de los conjuntos de puntos asignados a cada ruta se le aplica el TSPTW. Esto sólo es operativo cuando el número nodos asignados a cada ruta no es grande, no mayor de 9, (para este número los algoritmos exactos para el TSPTW apenas emplean tiempo), aunque el número de rutas sea grande. Con esto se consigue en muchos casos ahorrar tiempo en exploraciones posteriores.

En el ejemplo 4.4., la primera solución a la que se llega es 1 - 2 - 6 - 3 - 4 - 5 - 1; es decir dos rutas 1 - 2 - 1, y 1 - 3 - 4 - 5 - 1, con un coste total de 1.052. Optimizando la segunda ruta (la primera no haría falta), se obtendría una nueva solución formada por las rutas 1 - 2 - 1 y 1 - 4 - 3 - 5 - 1 con un coste total de 1.020, que coincide con la solución óptima que se obtendría en pasos posteriores.

La introducción de esta modificación afecta al paso 6 del procedimiento recursivo:

MODIFICACION 2.- Procedimiento RECURSIONGENERAL

Paso 6.- Determinar los conjuntos de nodos asignados a cada una de las rutas formada por los arcos de $R(\alpha)$.

Para cada uno de estos conjuntos:

si el número de nodos es pequeño, entonces:

Hallar la ruta óptima para ese conjunto

en caso contrario dejar la ruta invariante.

Poner *costeruta* = coste de todas las rutas.

Si *costeruta* < *costeminimo1* entonces:

costeminimo1 = *costeruta*;

poner en *ruta1* las Rutas que se han formado.

3.6.- DETERMINACION DEL MINIMO NUMERO DE VEHICULOS

Cuando además hay que determinar el mínimo número de vehículos para utilizar se propone el siguiente criterio:

1.- Considérese solamente el conjunto de clientes cuya demanda puede ser transportada junto con otras en un mismo camión.

2.- Calcular qtotal1* la cantidad total demandada por este conjunto de clientes. Calcular $m1 = \left\lceil \frac{qtotal1^*}{capacidad} \right\rceil$.

3.- Definir el número de rutas inicial m como m1 más el número de clientes cuya carga ha de ser transportada sola.

4.- Ejecutar el algoritmo, con el número de rutas m, sin vector de penalizaciones

5.- Si no se obtiene solución factible, poner m = m+1, y volver al paso anterior.

Este criterio apenas añade tiempo de computación ya que el *reconocimiento* de un problema infactible para un determinado número de vehículos m, suele ser muy rápido comparándolo con la ejecución de un problema factible.

4.- VARIANTES HEURISTICAS

Diversas experiencias indican que en el proceso de ramificación y acotación que utilizan los algoritmos Branch & Bound, la solución óptima se suele encontrar mucho antes de finalizar el proceso recursivo de búsqueda. Por otra parte, alguna de las ramas que se exploran se adivinan innecesarias ya que la cota inferior está muy próxima al mejor valor encontrado hasta ese momento y quedan muchos arcos por seleccionar. Así cabe esperar obtener un rápido algoritmo heurístico a partir de un exacto estableciendo criterios de parada en la exploración de determinadas ramas. Entre otros, consideramos:

1.- Rama Izquierda: Solamente se explora una rama, cada vez que se llama al proceso recursivo. Dicha rama es la que añade todos los arcos seleccionados a $R(\alpha)$.

2.- Aumento de la cota inferior: No se explora solamente la rama izquierda, pero se suman a las cotas inferiores obtenidas, una determinada cantidad función del número de arcos que quedan por seleccionar en esa rama.

3.- Tiempo máximo de computación que lleva funcionando el proceso recursivo. Este tiempo suele tomarse polinomial en el número de nodos.

4.- Parada interactiva por el propio usuario del sistema donde esté incorporado el algoritmo.

4.1.- RAMA IZQUIERDA

Los arcos elegidos para añadir a la solución, pertenecerán a la seudoruta que determina la cota inferior a la solución óptima en esa rama. Cabe esperar que la solución a la que se llegue selecionando recursivamente estos arcos estará cercana al óptimo. Así sólo se considera la búsqueda en la rama izquierda, es decir, en la que sólo se añade el conjunto de arcos que forman la ruta sin aumentar el conjunto de arcos prohibidos, que siempre permanecerá vacío. La incorporación de este criterio afecta al paso 5 del proceso recursivo que queda de la forma siguiente:

VARIANTE RAMA IZQUIERDA.- Procedimiento RECURSION GENERAL($R(\alpha)$, $\pi(\alpha)$)

Paso 5.- Poner $R(\alpha^*) = R(\alpha) \cup S$;
y $\pi(\alpha^*) = \pi^1$;
Ejecutar el procedimiento RECURSIONGENERAL($R(\alpha^*)$,$\pi(\alpha^*)$).

Obviamente no hace falta considerar el conjunto $R(\alpha)$, ni el vector de arcos ordensalida. Además se puede incorporar la modificación 2, propuesta anteriormente, optimizando las rutas obtenidas en la solución final.

4.2.- AUMENTO DE LAS COTAS INFERIORES

En el cálculo de la cota inferior en la exploración de cada rama, se observa que la diferencia entre esta cota y el óptimo de dicha rama, difieren más cuantos más arcos quedan por seleccionar, incluso aunque se utilicen los métodos de penalización descritos que mejoran el cálculo de dichas cotas inferiores. Es decir, se espera que el valor óptimo en esa rama, sea al menos el valor de la cota inferior más una cantidad proporcional al número de arcos que quedan por seleccionar.

El heurístico que se propone consiste en aumentar la cota inferior obtenida sumándole una cantidad proporcional al número de arcos que queden por seleccionar, y de los valores de la matriz de distancias.

VARIANTE DE LA COTA INFERIOR.- Procedimiento RECURSIONGENERAL

Paso 2.- Obtener la seudoruta correspondiente mediante el cálculo de $SR(\alpha)$;

poner:

w1 = cota inferior mínima asociada a esta seudoruta,

π^1 el vector de penalizaciones correspondientes;

si w1 + f(n+m-1-n1) < costeminimo1 ir al paso 3; si no ir al paso 7.

La función f que se añade a la cota inferior es habitualmente de la forma f = c·(n+m-1-n1). En la elección de la constante c se considera el valor medio de las distancias del problema y si se utiliza vector de penalizaciones para hallar la cota inferior, el valor de c se elige mucho menor.

4.3.- LIMITACION DEL TIEMPO DE COMPUTACION

Esta variante heurística consiste en fijar el máximo tiempo de computación o permitir la interrupción del algoritmo en cualquier momento una vez obtenida una solución.

VARIANTE MAXIMO TIEMPO DE COMPUTACION:

Se incorporan las siguientes variables: TSALIDA, TACT, TMAX y SOL, variable lógica que indica si se ha hallado una solución.

Procedimiento PRINCIPAL

Paso 1.- Leer todos los datos:

número de nodos n; número de vehículos m; ventanas de tiempo $[a_i , b_i]$ i = 1,...,n;
matriz de distancias (d_{ij}) matriz de tiempos (t_{ij});
Tiempo máximo de computación TMAX, Momento de inicio del algoritmo
TSALIDA.

Paso 3.- Inicializar las variables:
costeminimo1 = ∞; $R(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, $\pi(\alpha) = (0)$, SOL = FALSE.

Procedimiento RECURSIONGENERAL.-

Paso 0.- Tomar TACT, tiempo actual,
Si (TACT-TSAL >= TMAX) y SOL, entonces ir al paso 7, si no ir al paso 1.

Paso 3.- Si los nodos de $SR(\alpha)$ no están repetidos:
poner *costeminimo1* = w1,
ruta1 = *seudoruta*,
poner SOL = TRUE;
Ir al paso 7;
si no ir al paso 6.

Paso 6.- Poner *costeruta* = Coste de los arcos de $R(\alpha)$;
Si *costeruta* < *costeminimo1* entonces:
costeminimo1 = *costeruta*;
ruta1 = $R(\alpha)$;
poner SOL = TRUE.

El tiempo inicial TSALIDA puede tomarse en otro momento del algoritmo, preferiblemente después de ejecutarse el procedimiento CALCULOSEUDORUTAMEJORADO por primera vez.

VARIANTE INTERRUPCION INTERACTIVA:

Se incorporan las siguientes variables: SOL, igual que en la variante anterior y TECPUL, variable lógica que indica si una tecla ha sido pulsada, (algunos compiladores de lenguajes de alto nivel tienen incorporada dicha función, por ejemplo en el caso del TURBOPASCAL de Borland, esta función se denomina *Keypressed*).

Procedimiento PRINCIPAL.-

Paso 3.- Inicializar las variables:
costeminimo1 = ∞; $R(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, $\pi(\alpha) = (0)$, TECPUL, SOL = FALSE.

Procedimiento RECURSIONGENERAL.-

Paso 0.- Hacer TECPUL = TRUE si se ha pulsado el teclado;
Si TECPUL y SOL, entonces ir al paso 7, si no seguir ir al paso 1.

Paso 3.- Si los nodos de $SR(\alpha)$ no están repetidos:
Poner *costeruta* = w1, *ruta1* = *seudoruta*, OBT = TRUE, Ir al paso 7;
en caso contrario ir al paso 4.

Paso 6.- Poner *costeruta* = coste de los arcos de $R(\alpha)$;
Si *costeruta* < *costeminimo1* :
Poner *costeminimo1* = *costeruta*, *ruta1* = $R(\alpha)$, hacer SOL = TRUE.

CAPITULO 5: RESULTADOS EN PROBLEMAS SIMULADOS Y DESARROLLO DE UN EJEMPLO REAL

En este capítulo se pondrán a prueba los algoritmos heurísticos desarrollados en el capítulo anterior para comprobar tanto su operatividad y eficacia como sus limitaciones. Concretamente estos heurísticos son: VARIANTE RAMA IZQUIERDA y VARIANTE CONSIDERANDO EL TIEMPO DE COMPUTACION.

En la primera sección se simularán varias matrices de distancias para diferentes números de nodos. Cada problema simulado se resolverá utilizando, además de los algoritmos mencionados, el algoritmo exacto a partir del cual se desarrollan y los algoritmos de Beasley, (1.983), y Fisher & Jaikumar, (1.981).

En la segunda sección se desarrollará un ejemplo práctico en el que se mostrarán los resultados obtenidos por estos dos heurísticos comparándolos con los obtenidos por los heurísticos de Beasley y Fisher & Jaikumar.

El método de asignación lineal generalizada de Fisher y Jaikumar, y el algoritmo propuesto por Beasley, son considerados como dos de los algoritmos más eficaces desarrollados en los últimos años, (Haouari y otros, (1.990), Assad, (1.988), Benavent y otros, (1.985)). El primero es un representante del tipo de algoritmos basados en modelos de Programación Matemática, y en técnicas *Clusters 1º-Rutas 2º*. Por su parte el algoritmo de Beasley, es un representante claro de los algoritmos de tipo *Ruta 1º-Cluster 2º*.

1. RESULTADOS PARA PROBLEMAS SIMULADOS

Esta sección recoge los resultados proporcionados por los algoritmos VARIANTE RAMA IZQUIERDA y VARIANTE CONSIDERANDO EL TIEMPO MAXIMO DE COMPUTACION, en la resolución de un conjunto de problemas asociados a matrices de distancias obtenidas por simulación. Se han realizado 10 simulaciones para cada número de nodos: 10, 15, 20, 25, 30, 35 y 40, tanto para matrices simétricas como no simétricas. Se compararán dichos resultados con los obtenidos por el algoritmo de Beasley y el algoritmo exacto desarrollado en el capítulo anterior.

1.1.- ALGORITMO DE BEASLEY

El algoritmo de Beasley ha sido explicado en el capítulo 2. En esta sección se describen las modificaciones y variantes introducidas para la ejecución en los problemas simulados.

Algoritmo de BEASLEY ADAPTADO

Paso 0: (Formar una *gran ruta* inicial).

Resolver el TSP con todos los clientes sin incluir el depósito inicial 1.

Supóngase, sin pérdida de generalidad, que el primer cliente visitado en la ruta obtenida tiene asignado el índice 2, el segundo el 3, ... y así sucesivamente.

Poner k = 2, *costeminimo* = ∞, *granruta*[1]=1.

Paso 1: Para i = 2,..., n-k+2: poner *granruta1*[i] = i+k-2,

Para i = n-k+3,...,n: poner *granruta1*[i] = i-n+k-1.

Paso 2: (Construcción de la matriz de distancias c)

Definir la matriz $c = (c_{ij})$ de la forma siguiente:

si i < j y la ruta R que parte del depósito, visita los clientes *granruta1*[i+1], *granruta1*[i+2],.. y *granruta1*[j], y regresa al depósito es factible poner c_{ij} = longitud de la ruta R;
en caso contrario (i ≥ j ó R no factible) poner $c_{ij} = \infty$.

Paso 3: (Determinar la *partición óptima*).

Definir la red completa con N = {1,2,...,n} y con longitud del arco (i,j) c_{ij} .
Determinar el camino más corto del nodo 1 hasta el n en esta red.
Para cada arco (i,j) de este camino formar un grupo con los clientes *granruta1*[i+1], *granruta1*[i+2], ..., *granruta1*[j].

Paso 4: Para cada uno de los grupos definidos en el paso 3, determinar la ruta óptima que saliendo del nodo 1 visita los nodos de dicha partición.
Poner *costeminimo1* = suma de los costes de las rutas asociadas a cada grupo.

Paso 5: Si *costeminimo1* < *costeminimo* entonces:

Poner *costeminimo* = *costeminimo1*,
Registrar la solución hallada.

Poner k = k+1; si k ≤ n ir al paso 1, en caso contrario ir al paso 6.

Paso 6: Parar.

En el paso 0 para formar la gran ruta inicial se emplea el algoritmo de Held & Karp, (1.970) y (1.971), en el caso de matrices simétricas, y el algoritmo RECHAZO HEURISTICO, propuesto por Aragón y Pacheco, (1.992), para las matrices asimétricas. La incorporación de las variables auxiliares k y *granruta1* tiene como objeto, manteniendo el orden establecido en la gran ruta inicial, repetir los siguientes pasos comenzando cada vez con un nodo diferente y registrar la mejor solución.

En el paso 2, la factibilidad de cada ruta al determinar la matriz (c_{ij}) se determina considerando, además de las cargas, las correspondientes ventanas de tiempo.

En el paso 3, el camino mínimo en la red se calcula utilizando el algoritmo de Djikstra.

En el paso 4, las rutas óptimas para cada uno de los grupos se determinan utilizando el algoritmo exacto para el TSPTW desarrollado en el capítulo anterior. En el caso que haya pocos nodos por grupo se recomienda ejecutar este algoritmo sin utilizar vectores de penalizaciones.

1.2- PROGRAMACION DEL ALGORITMO DE FISHER Y JAIKUMAR

Según se vio en el capítulo 2, el algoritmo de Fisher y Jaikumar se basa en el tratamiento del VRP como un problema de Asignación Lineal Generalizado. Para ello se han de definir previamente una serie de puntos semilla, tantos como el máximo número de rutas que se desean generar. A continuación se describe el método utilizado para determinar dichos puntos semilla.

El proceso de selección de clientes o puntos semilla $w_1, w_2, \ldots, w_K$, propuesto por los propios autores del algoritmo, es el siguiente:

Procedimiento DETERMINARPUNTOSSEMILLA

Paso 1: (Formar *conos* de clientes)

Para i = 2,...,n, determinar las semirectas r_i que unen el depósito con el cliente i.

Determinar las bisectrices de los ángulos formados por las semirectas r_i.

Cada par de bisectrices consecutivas forman el cono C(i) asociado al cliente i.

Paso 2: Asociar un peso q_i a cada cono C(i), para i = 2,..,,n.

Definir $\alpha = \frac{\sum_{i=2}^{n} q_i}{m \cdot capac}$.

Paso 3: (Formar conos mayores asignados a cada vehículo)

Poner qtotal = Suma de las demandas de todos los clientes.

Comenzando con C(2) unir conos o *fracciones* de conos de clientes adyacentes, en el sentido de las agujas del reloj, de forma que el peso total del cono formado sea exactamente α·qtotal. Las *fracciones* de cono que se unen deben ser proporcionales a la cantidad de peso que se quiere añadir al cono grande.

Paso 4: (Determinación de los puntos-semilla)

El punto semilla w_k se coloca en la bisectriz de los conos grandes de forma que el *peso* correspondiente a ese punto sea 0'75·α·capac. Este peso se define como la suma de las demandas q_i de los clientes que quedan *por debajo* del arco que desde ese punto barre el cono, más una fracción de la demanda q_i del cliente en el cono más próximo *por fuera* de este arco; esta fracción va a ser de la forma A/(A+B), donde A es la distancia del origen al cliente más cercano al arco *por dentro*, y B la distancia del origen a este cliente más cercano *por fuera*.

La figura 27 muestra este proceso.

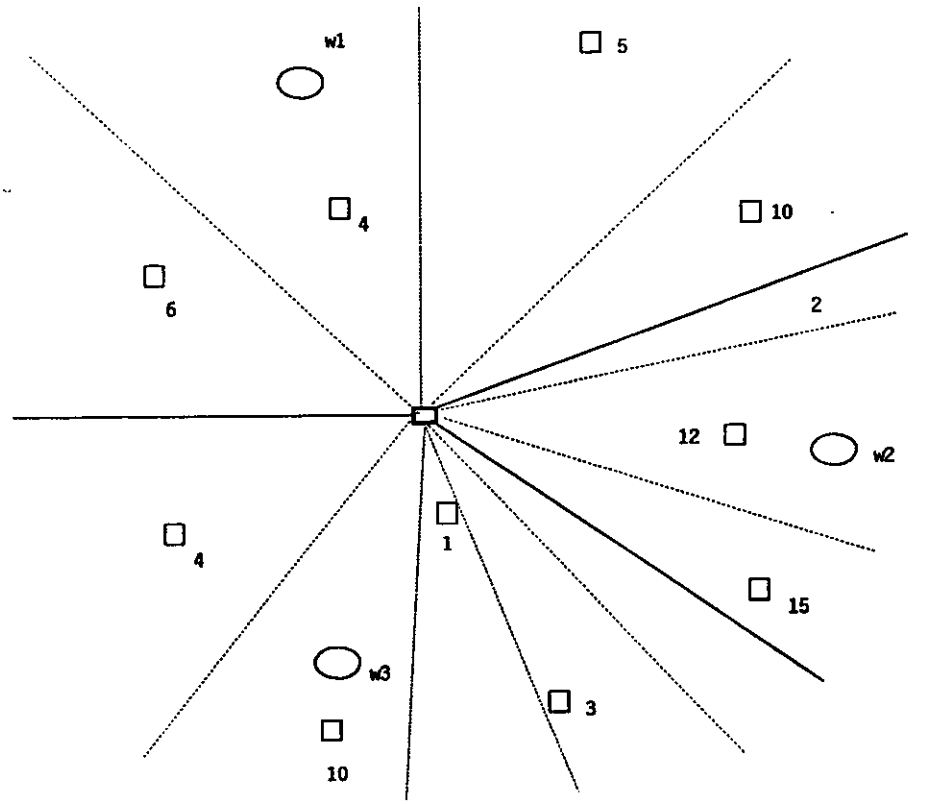


Figura 27. Selección de los puntos-semilla. Cada punto tiene indicado su demanda.

Una vez obtenidos los puntos semilla y formulado el problema como un modelo de Asignación Lineal Generalizada, éste se resuelve utilizando un heurístico propio, diseñado a partir de una modificación del propuesto por Raft, (1.982), para el GAP. Para explicar dicho heurístico se define

e_{ik}: coste de asignar el cliente i a la ruta que va del origen al punto semilla w_k y vuelve;

Δ_i: factor de penalización, igual al coste de no asignar al cliente i al punto semilla más cercano sino al segundo más cercano, es decir

$$\Delta_i = -e_{ij*} + \min_{j \neq j*} e_{ij},$$

siendo $e_{ij*} = \min_j e_{ij}$. Este factor de penalización Δ_i se ha de calcular considerando sólo las asignaciones posibles en cada caso.

Procedimiento GAPHEURISTICO

Paso 1.- Calcular los factores de penalización Δ_i para todos los clientes.

Paso 2.- Ordenar los clientes sin asignar en orden decreciente de Δ_i.

Paso 3.- Tomar el primer cliente en esta ordenación y asignarlo al punto semilla más cercano.

Paso 4.- Si no quedan clientes sin asignar parar;
en caso contrario:
chequear las asignaciones posibles para los clientes que restan,
Recalcular los factores de penalización Δ_i para estos clientes,
Ir al paso 2.

Una vez asignados todos los puntos a los diferentes puntos-semilla se determina la ruta óptima para cada conjunto de puntos obtenido. Para ello se ejecuta el algoritmo exacto para el TSPTW desarrollado en el capítulo anterior.

Se repite todo el proceso de determinación de los puntos semilla y asignación de clientes a éstos, cambiando en cada iteración el cono C(i) a partir del cual se forman conos mayores. El proceso finaliza cuando se han probado todos los conos.

1.3.- PROGRAMACION DE LOS ALGORITMOS HEURISTICOS Y EXACTO

Los algoritmos heurísticos que se van a examinar son, en esencia, el mismo algoritmo exacto en el que se toman soluciones intermedias sin esperar a la finalización de su ejecución. Por consiguiente, para la realización de la simulación y la posterior comparación de resultados, se ha

creído conveniente ejecutar solamente el algoritmo exacto incorporando una serie de modificaciones con objeto de registrar estas soluciones intermedias y, en el caso del algoritmo VARIANTE RAMA IZQUIERDA, el tiempo en el que se han obtenido.

Para ello se incorporan las siguientes nuevas variables:

TSAL : Tiempo de comienzo de la ejecución del algoritmo para cada problema;
TACTUAL : Variable donde se guarda el tiempo actual;
TMAX : Tiempo máximo establecido para el algoritmo VARIANTE TIEMPO MAXIMO DE COMPUTACION;
OBT : Variable lógica que indica si la primera solución se ha hallado;
costeminimo2: Valor de la primera solución hallada;
costeminimo3: Valor de la mejor solución hallada sin rebasar TMAX de tiempo de ejecución.

Las modificaciones realizadas son las siguientes:

Procedimiento PRINCIPAL

Paso 1.- Leer los datos:

n; m; $[a_i , b_i]$ i = 1,...,n; las matrices (d_{ij}) y (t_{ij}); TMAX, TSAL.

Paso 3.- Inicializar las variables:

costeminimo1 = ∞, *costeminimo2* = ∞, *costeminimo3* = ∞, $R(\alpha) = \{1\}$, $F(\alpha) = \varnothing$, $\pi(\alpha) = (0)$, OBT = FALSE.

Procedimiento RECURSIONGENERAL.-

Paso 3.- Si los nodos de $SR(\alpha)$ no están repetidos:

Poner *costeminimo1* = w1, *ruta1* = *seudoruta*,

Definir TACT como el tiempo actual,

si (TACT - TSAL < TMAX) o (no OBT) poner *costeminimo3* = w1;

si (no OBT) poner *costeminimo2* = w1, OBT = TRUE;

Ir al paso 7.

Paso 6.- Poner *costeruta* = coste de las rutas formadas por los arcos de $R(\alpha)$;

Si *costeruta* < *costeminimo1* hacer:

costeminimo1 = *costeruta*;

ruta1 = Rutas formadas por $R(\alpha)$;

Tomar el tiempo actual TACT;

si (TACT - TSAL < TMAX) o (no OBT)

poner *costeminimo3* = w1;.

si (no OBT) poner *costeminimo2* = w1, OBT = TRUE.

1.4.- SIMULACION DE LOS PROBLEMAS

Según se ha comentado anteriormente, se han establecido cuatro niveles correspondientes a diferentes números de nodos: 10, 15, 20, 25, 30 35 y 40; para cada nivel se han simulado 10 matrices de distancias diferentes, tanto simétricas como asimétricas, y los correspondientes problemas, de la siguiente forma para cada una de ellas:

- A cada cliente i se le asigna un punto p_i en una retícula cuadrada de lado 100: las coordenadas de este punto toman valores enteros uniformemente entre 0 y 99.

- Los diferentes valores de la matriz de distancias, expresados en Kms., se obtienen como $d_{ij} = \lfloor \|p_i - p_j\|_2 \rfloor$ para las matrices simétricas. Para obtener la matrices asimétricas se

perturba los valores de cada una de las matrices simétricas multiplicándolos por un coeficiente aleatorio entre 0.9 y 1.1.

- Las demandas q_i se obtienen generando valores enteros con distribución uniforme entre 1 y *capac*. Se toma la capacidad de los vehículos como 8 unidades de carga, las ventanas de tiempo de la forma [0,240], y la velocidad de todos los vehículos de 60 Kms./hora. No se ha considerado tiempo de descarga, de esta forma los valores de la matriz de tiempos se van a definir como $t_{ij} = d_{ij}$ expresado en minutos.

- En cuanto al tiempo máximo de computación TMAX, se considera que este debe ser función polinómica del número de nodos; en este caso se ha definido TMAX = n·15, expresado en segundos, siendo n el número de nodos.

1.5.- RESULTADOS

1.5.1.- MATRICES SIMETRICAS

A continuación se muestra un cuadro en el que se resumen los resultados obtenidos para los 10 problemas de matrices simétricas para cada número de nodos considerado. El cuadro incluye los costes medios, la desviación porcentual del óptimo, y los tiempos de computación en segundos (medio, máximo y mínimo):

			10 nodos	15 nodos	20 nodos	25 nodos	30 nodos	35 nodos	40 nodos
Beasley	*Coste*		766.4	1091.7	1389.3	1748.7	2414.1	2234.9	3032.0
	Desvia.	*Medio*	8.37	7.41	4.95	9.33	7.22	-	-
		Mínimo	0	0	0	0.43	0	-	-
		Máximo	16.89	17.65	14.65	16.85	14.22	-	-
	Tiempo	*Medio*	0.158	0.660	1.489	8.206	25.957	137.068	489.03
		Mínimo	0	0.22	0.06	1.27	2.58	4.67	21.86
		Máximo	0.27	1.38	2.64	35.26	94.04	482.30	976.53
Fisher & Jaikumar	*Coste*		725.3	1071.1	1371.0	1622.1	2296.3	2221.3	2822.3
	Desvia.	*Medio*	2.56	5.38	3.57	1.42	1.99	-	-
		Mínimo	0	0	0	0	0	-	-
		Máximo	16.89	17.97	12.12	5.64	7.77	-	-
	Tiempo	*Medio*	0.138	0.412	0.841	1.516	2.263	3.376	4.982
		Mínimo	0.11	0.28	0.49	0.99	1.76	2.63	3.90
		Máximo	0.17	0.66	1.81	3.02	2.91	4.39	6.15
Rama Izda.	*Coste*		736.3	1053.0	1384.7	1681.8	2295.4	2132.4	2867.2
	Desvia.	*Medio*	4.11	3.60	4.60	5.15	1.95	-	-
		Mínimo	0	0	0	0	0	-	-
		Máximo	16.14	14.59	12.31	18.16	7.03	-	-
	Tiempo	*Medio*	0.126	3.996	8.385	19.602	49.324	163.511	198.156
		Mínimo	0.10	2.43	4.49	6.45	8.99	52.57	85.25
		Máximo	0.25	6.09	14.95	50.98	156.22	322.55	389.43
T.Máximo	*Coste*		707.2	1016.4	1323.8	1599.4	2251.9	2075.7	2741.6
	Desvia.	*Medio*	0	0	0	0.01	0.01	-	-
		Mínimo	0	0	0	0	0	-	-
		Máximo	0	0	0	0.1	0.09	-	-
	Tiempo	*Medio*	52.633	135.726	180.084	329.03	450	525	600
		Mínimo	29.41	130.64	38.20	60.01	450	525	600
		Máximo	79.39	148.48	228.71	375	450	525	600
Exacto	*Coste*		707.2	1016.4	1323.8	1599.3	2251.6	-	-
	Tiempo	*Medio*	52.633	135.726	180.084	339.02	958.361	-	-
		Mínimo	29.41	130.64	38.20	60.1	477.52	-	-
		Máximo	79.39	148.48	228.71	450.22	2135.99	-	-

La figura 28 muestra gráficamente las desviaciones respecto del óptimo

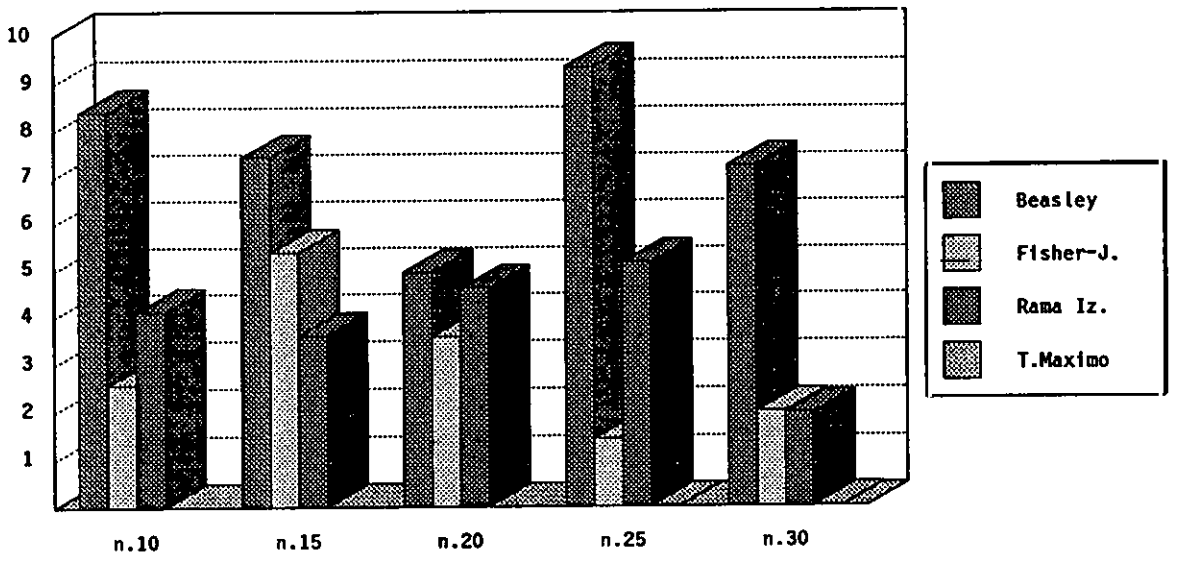


Figura 28. Desviaciones porcentuales medias del valor óptimo para los algoritmos de Beasley, Fisher y Jaikumar, Rama Izquierda y Tiempo Máximo.

1.5.2.- MATRICES ASIMETRICAS

El cuadro siguiente y la figura 29 muestran los resultados obtenidos para los 10 problemas de matrices asimétricas

			10 nodos	15 nodos	20 nodos	25 nodos	30 nodos	35 nodos	40 nodos
Beasley	*Coste*		758.9	1100.4	1368.9	1708.7	2342.0	2269.4	2991.4
	Desvia.	*Medio*	9.04	9.60	4.85	9.45	7.48	-	-
		Mínimo	0	3.09	0	1.86	0	-	-
		Máximo	16.49	28.31	8.48	17.71	16.56	-	-
	Tiempo	*Medio*	0.034	0.089	0.144	0.223	0.357	0.648	0.778
		Mínimo	0.00	0.05	0.11	0.11	0.27	0.55	0.65
		Máximo	0.06	0.49	0.22	0.24	0.49	0.82	1.10
Fisher & Jaikumar	*Coste*		711.5	1057.4	1330.2	1583.0	2209.7	2181.8	2764.1
	Desvia.	*Medio*	2.23	5.32	1.88	1.40	1.41	-	-
		Mínimo	0	0	0	0	0	-	-
		Máximo	16.12	15.63	6.55	6.71	8.34	-	-
	Tiempo	*Medio*	0.148	0.402	0.834	1.472	2.286	3.352	4.939
		Mínimo	0.11	0.27	0.44	0.99	1.81	2.69	4.61
		Máximo	0.22	0.66	1.81	2.53	3.13	4.23	5.82
Rama Izda.	*Coste*		733.5	1047.0	1385.1	1636.5	2224.1	2097.7	2834.1
	Desvia.	*Medio*	5.39	4.28	6.09	4.82	2.07	-	-
		Mínimo	0	0	0	0	0	-	-
		Máximo	15.54	20.42	12.31	18.99	18.99	-	-
	Tiempo	*Medio*	0.126	3.795	8.539	19.122	43.584	162.112	191.024
		Mínimo	0.11	2.54	4.85	5.99	11.25	49.32	83.25
		Máximo	0.27	5.89	13.89	51.02	137.87	316.87	392.55
T.Máximo	*Coste*		696.0	1004.0	1305.6	1561.2	2179.2	2026.3	2689.2
	Desvia.	*Medio*	0	0	0	0.01	0.01	-	-
		Mínimo	0	0	0	0	0	-	-
		Máximo	0	0	0	0.1	0.09	-	-
	Tiempo	*Medio*	50.585	134.965	175.668	320.387	450	525	600
		Mínimo	27.22	130.25	30.06	45.88	450	525	600
		Máximo	79.37	149.26	227.34	375	450	525	600
Exacto	*Coste*		696.0	1004.0	1305.6	1561.1	2179.0	-	-
	Tiempo	*Medio*	50.585	134.965	175.668	360.254	948.335	-	-
		Mínimo	27.22	130.25	30.06	45.88	439.25	-	-
		Máximo	79.37	149.26	227.34	628.20	2098.54	-	-

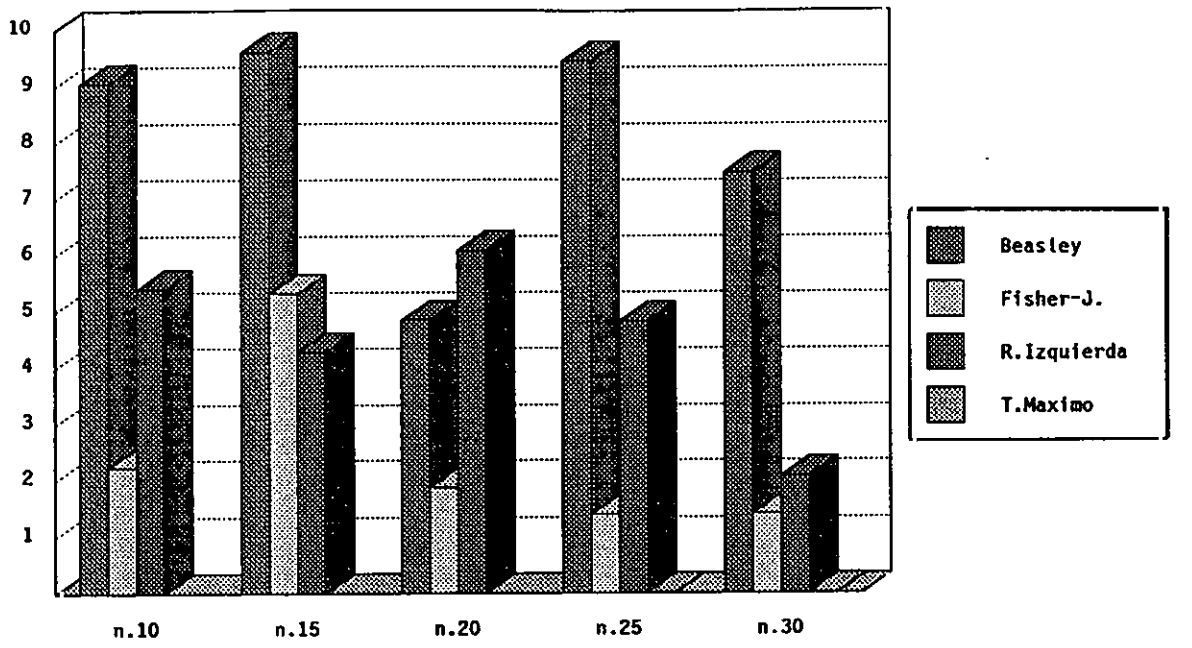


Figura 29. Desviaciones porcentuales medias del valor óptimo para los algoritmos de Beasley, Fisher y Jaikumar, Rama Izquierda y Tiempo Máximo.

2.- RESOLUCION DE UN PROBLEMA REAL

Al igual que en la sección anterior, en ésta se examinarán los dos heurísticos anteriores, VARIANTE RAMA IZQUIERDA y VARIANTE TIEMPO MAXIMO, a través de un problema real comparando sus resultados con los obtenidos por los algoritmos de Beasley y el de Fisher y Jaikumar.

La programación del algoritmo VARIANTE RAMA IZQUIERDA y del algoritmo de Beasley es igual que la que se utilizó en la sección anterior. En la programación del algoritmo VARIANTE TIEMPO MAXIMO se introduce una pequeña variación: la variante TSAL no toma como valor el tiempo de comienzo del algoritmo, sino el tiempo de finalización del procedimiento

CALCULOSEUDORUTAMEJORADO cuando se ejecuta por primera vez el procedimiento RECURSIONGENERAL.

Para el algoritmo de Fisher & Jaikumar se considera el mapa de España como el plano donde están situados los puntos que intervienen en el problema; más concretamente, las coordenadas de dichos puntos van a ser las coordenadas gráficas de la representación de ese mapa en ordenador obtenida a partir de información digitalizada. Se considera la distancia euclídea entre los diferentes puntos, obtenida a partir de dichas coordenadas.

2.1.- DESCRIPCION DEL PROBLEMA

Este problema se le plantea todos los días a una empresa fabricante de productos de pastelería, para distribuir sus productos a las delegaciones desde la fábrica central situada en Briviesca (Burgos).

Las demandas de las delegaciones se distribuyen en forma de *Carros* o *Palés*, por tanto ésta será la unidad de carga a considerar. El objetivo principal del problema es minimizar el número de vehículos, mientras que los kilómetros recorridos es un objetivo secundario.

La entrega en los horarios establecidos es fundamental para esta empresa y no se puede violar bajo ningún concepto. Esta no es una cuestión tan trivial ya que existen empresas que prefieren una demora relativa en las entregas si esto les supone una reducción sustancial en el coste de la distribución propiamente dicha; sin embargo en este caso no es así.

Los tiempos de conducción y descanso de los conductores vienen determinados por la ley. En cualquier caso, al trabajarse simultáneamente con conductores en nómina y conductores autónomos, la flexibilidad al tratar estos tiempos no es igual para todos, y es difícil establecer un criterio claro homogéneo. Por consiguiente se optó por la siguiente solución para este ejemplo:

establecer una velocidad media teórica constante, en la que se reflejan la velocidad real y estos tiempo de descanso. Si la velocidad real es de 80-85 kms/hora, y los intervalos de conducción y descanso son de 4 conducir-1 descansar 4-conducir-6 descansar, se tomó una velocidad media de 45 kms/hora. Así se asegura que la solución obtenida por este criterio puede llevarse a la realidad.

El problema corresponde a las demandas que debían ser satisfechas el Lunes 21 de Junio de 1.993.

Fábrica Central:

1.-Briviesca (Burgos).

Delegaciones y demanda (en número de carros):

Delegación	*Demanda*	*Delegación*	*Demanda*	*Delegación*	*Demanda*
2.- Orense	6	11.- Pamplona	7	20.- Tarragona	17
3.- Pontevedra	22	12.- Salamanca	8	21.- Lérida	14
4.- Santiago	4	13.- Valladolid	4	22.- Santiga-Bna.	12
5.- La Coruña	28	14.- Palencia	15	23.- S.Quirce	12
6.- Lugo	8	15.- Burgos	14	24.- Villanova	8
7.- Santander	28	16.- Logroño	3	25.- Madrid	28
8.- Bilbao	26	17.- Quel	3	26.- Cáceres	8
9.- Vitoria	17	18.- Huesca	7	27.- Mérida	14
10.- S.Sebastian	4	19.- Zaragoza	8		

Capacidad de los vehículos: 28 carros.

Tamaño de la Flota: 20 vehículos.

Horarios y Tiempos Establecidos:

Tiempo Mínimo de salida: 12. 00 del día anterior al de la entrega

Tiempo Máximo de vuelta a Briviesca: 18. 45 del día siguiente al de la entrega

Hora mínima de recogida en las delegaciones: 9. 00.

Hora máxima de recogida: 21.30.

Tiempo medio de descarga: 12 minutos

2.2. RESULTADOS

Los resultados obtenidos por los algoritmos son los siguientes:

	Beasley	Fisher & Jaikumar	Rama Izquierda	Tiempo Máximo
Nº de Rutas	16	14	13	13
Nº de Kilómetros	11.678	12.011	10.741	10.637
Tiempo de Computación	55"	17"	220"	502"

En esta tabla se ve claramente la eficacia de estos nuevos algoritmos a la hora de minimizar el número de vehículos frente a los de Beasley y el de Fisher & Jaikumar; el tiempo de computación sin embargo es mucho mayor, aunque siempre va controlado por el propio carácter de los algoritmos.

A continuación se muestra con detalle las soluciones obtenidas por los algoritmos.

2.2.1.-SOLUCION DE BEASLEY

Nº de Rutas: 16.

Nº de Kilómetros: 11.678.

Itinerario:

	Puntos Visitados	Distancia Recorrida (Kms.)
Ruta 1	1 - 12 - 2 - 1	1.101
Ruta 2	1 - 27 - 26 - 1	1.086
Ruta 3	1 - 25 -1	572
Ruta 4	1 - 14 - 13 - 1	344
Ruta 5	1 - 15 - 1	98
Ruta 6	1 - 17 - 1	114
Ruta 7	1 - 16 - 17 - 19 - 24 - 1	1.196
Ruta 8	1 - 22 - 23 - 1	1.084
Ruta 9	1 - 20 - 1	958
Ruta 10	1 - 11 - 18 - 21 - 1	843
Ruta 11	1 - 10 - 1	350
Ruta 12	1 - 8 - 1	228
Ruta 13	1 - 7 - 1	326
Ruta 14	1 - 6 - 1	968
Ruta 15	1 - 28 - 1	1.168
Ruta 16	1 - 3 - 4 - 1	1.242

2.2.2.-SOLUCION DE FISHER & JAIKUMAR

Número de Rutas: 14

Número de Kilómetros: 12.011

Itinerario.

	Puntos Visitados	Distancia Recorrida (Kms.)
Ruta 1	1 - 3 - 1	1.178
Ruta 2	1 - 27 - 15 - 1	1.086

	Puntos Visitados	Distancia Recorrida (Kms.)
Ruta 3	1 - 14 - 6 - 4 - 1	1.190
Ruta 4	1 - 7 - 1	326
Ruta 5	1 - 9 - 1	114
Ruta 6	1 - 21 - 1	774
Ruta 7	1 - 24- 23 - 1	1.184
Ruta 8	1 - 17 - 19 - 20 - 1	970
Ruta 9	1 - 25 - 1	572
Ruta 10	1 - 26 - 12 - 13 - 2 - 1	1.672
Ruta 11	1 - 5 - 1	1.168
Ruta 12	1 - 8 - 1	228
Ruta 13	1 - 10 - 11 - 1	419
Ruta 14	1 - 16 - 18 - 22 - 1	1.130

2.2.3.- SOLUCION DE 'RAMA IZQUIERDA'

Número de rutas generadas: 13.

Número de Kilómetros: 10.741.

Itinerario:

	Puntos Visitados	Distancia Recorrida (Kms.)
Ruta 1	1 - 15 - 11 - 1	400
Ruta 2	1 - 21 - 19 - 1	774
Ruta 3	1 - 17 - 20 - 24 - 1	1.231
Ruta 4	1 - 27 - 26 - 1	1.086
Ruta 5	1 - 6 - 4 - 2 - 1	1.207
Ruta 6	1 - 10 - 3 - 1	1.528

	Puntos Visitados	Distancia Recorrida (Kms.)
Ruta 7	1 - 18 - 9 - 1	661
Ruta 8	1 - 16 - 23 - 22 - 1	1.084
Ruta 9	1 - 14 - 13 - 12 - 1	574
Ruta 10	1 - 5 - 1	1.168
Ruta 11	1 - 7 - 1	228
Ruta 12	1 - 8 - 1	228
Ruta 13	1 - 25 - 1	572

2.2.4.- SOLUCION DE 'TIEMPO MAXIMO DE COMPUTACION'

Número de rutas generadas: 13.

Número de Kilómetros: 10.637

Itinerario:

	Puntos Visitados	Distancia Recorrida
Ruta 1	1 - 26 - 27 - 1	1.086
Ruta 2	1 - 16 - 17 - 19 - 13 - 12 - 1	1.002
Ruta 3	1 - 24 - 21 - 1	1.184
Ruta 4	1 - 22 - 23 - 1	1.084
Ruta 5	1 - 18 - 20 - 1	1.004
Ruta 6	1 - 9 - 11 - 10 - 1	419
Ruta 7	1 - 15 - 1	98
Ruta 8	1 - 14 - 6 - 4 - 1	1.190
Ruta 9	1 - 2 - 3 - 1	1.178
Ruta 10	1 - 5 - 1	1.168
Ruta 11	1 - 7 - 1	326
Ruta 12	1 - 8 - 1	326
Ruta 13	1 - 25 - 1	572

3.- CONCLUSIONES

Los resultados obtenidos por estos algoritmos en la resolución de los problemas simulados y el problema real permiten extraer las siguientes conclusiones:

- El algoritmo exacto desarrollado para el VRPTW sólo es operativo para un número de nodos pequeño (10, 15, 20, a lo sumo 25 en algunos casos), o cuando se dispone de mucho tiempo para la obtención de los resultados.

- El algoritmo heurístico TIEMPO MAXIMO DE COMPUTACION, desarrollado a partir del exacto, se desvía de la solución óptima una cantidad insignificante siendo operativo para problemas de mayor tamaño. Se observa que, en general, cuando se conoce una solución cercana al óptimo en un problema combinatorio, conseguir una mejora pequeña suele llevar mucho tiempo de cálculo.

- El algoritmo RAMA IZQUIERDA también da buenas soluciones y en menos tiempo; éstas son similares a las obtenidas por el algoritmo de Fisher & Jaikumar y mejores que las del algoritmo de Beasley.

- Aunque otros algoritmos heurísticos, como el de Beasley o el de Fisher y Jaikumar dan en conjunto soluciones más desviadas del óptimo, sin embargo usan un tiempo de computación menor.

- El tiempo de computación para los heurísticos propuestos es excesivo si se necesita la solución de forma *inmediata*. Para estos casos son más recomendables otros como los de Beasley o los de Fisher & Jaikumar. Sin embargo, si se dispone de un tiempo moderado

para la obtención de la solución los algoritmos RAMA IZQUIERDA y TIEMPO MAXIMO DE COMPUTACION son más recomendables, obteniendo soluciones 10 % mejores.

- Se realizaron las mismas pruebas para matrices arbitrarias resultando que las diferencias entre los algoritmos propuestos y los de Beasley y Fisher & Jaikumar eran aún mayores, y el tiempo de computación era en general también mayor para todos los algoritmos.

- El algoritmo de Fisher & Jaikumar se comporta ligeramente mejor en las simulaciones que en el problema real, si se le compara con el resto de los algoritmos. Esto es debido a que en los problemas simulados la distancia entre los puntos coincide con su distancia euclídea. Sin embargo en los puntos del problema real esto no es así. A diferencia de lo que ocurre con este algoritmo, los heurísticos propuestos operan sin que sea necesario una representación planar de los puntos que intervienen en el problema.

En resumen, para problemas con hasta 30 nodos, y disponiendo de un tiempo de computación moderado, 10 o 15 minutos, estos algoritmos ofrecen soluciones muy poco desviadas del óptimo, en el caso de la variante TIEMPO MAXIMO DE COMPUTACION insignificante. Es estos casos, su utilidad está por tanto fuera de toda duda. Estas consecuencias pueden ser más optimistas si se tiene en cuenta que se ha trabajado con un ordenador personal y, por tanto, se podría aumentar el tamaño de los problemas para los que son operativos o disminuir el tiempo requerido para obtener la solución si se ejecutaran en *Miniordenadores*, *Mainframes*, *Macroordenadores*...; o incluso en un ordenador personal de arquitectura superior como los *Pentium* a 100 Mhz.

CAPITULO 6: EXTENSIONES Y NUEVAS DIRECCIONES DE TRABAJO

En este capítulo se presentan brevemente las principales líneas de investigación que se derivan de este trabajo.

La primera de ellas consiste en la aplicación de los algoritmos exactos para el TSP y TSPTW, convenientemente modificados, a otros problemas de rutas. El algoritmo básico para ello es el descrito en el capítulo 4 y entre los modelos considerados destacan los de sistema de descarga LIFO (se descarga la última mercancía que entró en el vehículo).

La segunda línea de investigación estudia técnicas de solución para la variante del VRP en la que no se considera la restricción de que cada cliente debe ser satisfecho por un solo vehículo. El estudio de este modelo es importante para la resolución de problemas reales en los que hay que minimizar el número de vehículos a usar.

En ambos casos, en este capítulo se describen únicamente los primeros pasos de esas nuevas líneas de investigación. Por ello las técnicas descritas no son definitivas, y se espera mejorar los resultados obtenidos hasta ahora en futuras investigaciones.

1.- EXTENSIONES DE ALGORITMOS EXACTOS PARA EL TSP Y TSPTW A OTROS MODELOS

1.1.- ADAPTACION A UNA VARIANTE DEL TSPTW CON CARGA Y DESCARGA.

El algoritmo utilizado para resolver el TSPTW, descrito en el capítulo 4, puede ser extendido a otros problemas de rutas, con pequeñas modificaciones en la formulación dinámica recursiva inicial.

Supóngase el problema en el que además de repartir unas determinadas cantidades de mercancía en un conjunto de localizaciones hay que recoger otras en otros puntos. Por ejemplo, en el caso práctico expuesto en el capítulo 5, los responsables de la empresa de Repostería plantearon el deseo de rutas que recogieran *carros* con materias primas o *carros* vacíos de envíos anteriores a la vez que realizaban la actividad de reparto habitual.

Para ello sea N={1,2,..., n}, el conjunto de localizaciones. Los puntos a visitar se dividen en dos subconjuntos: {2,3,...,n1} el conjunto de puntos donde hay que repartir y {n1+1, n1+2,...,n} el conjunto de puntos donde hay que recoger, siendo q_i la cantidad de mercancía que hay que enviar, si $i \leq n1$, o que hay que recoger si $i > n1$. El modelo, así planteado, puede considerarse como una extensión del TSPTW.

El problema puede ser descrito en términos de programación dinámica en la siguiente forma. Para $S \subseteq \{N-1\}$ y $j \in S$ se define:

f(S,j): coste mínimo de visitar todos los nodos de S comenzando en el vértice 1 en el instante 0 y finalizando en j;

u(S,j): tiempo de llegada a j en dicho recorrido;

el(S,j): Espacio libre que queda en el vehículo al realizar ese trayecto después de servir a j;

la formulación recursiva queda de la forma siguiente $f(S,j) = \min_{i \in S-\{j\}} \{f(S-\{j\},i) + d_{ij}\}$;

sea i(j) el nodo correspondiente a este mínimo, entonces:

$u(S,j) = u(S-\{j\},i(j)) + t_{i(j)j}$,
$el(S,j) = el(S-\{j\},i(j)) + q_j$, si $j \leq n1$,
$el(S,j) = el(S-\{j\},i(j)) - q_j$, si $j > n1$;

redefiniéndose:

$u(S,j) = a_j$, si $u(S,j) < a_j$,
$f(S,j) = \infty$, si $u(S,j) > b_j$,
$f(S,j) = \infty$, si $el(S,j) < 0$;

inicialmente $\forall i \in N - \{1\}$ se define:

$f(\{i\},i) = d_{1i}$,
$el(\{i\},i) = capac - \Sigma_{j=2}^{n1} q_j + q_i$, si $i \leq n1$,
$el(\{i\},i) = capac - \Sigma_{j=2}^{n1} q_j - q_i$, si $i > n1$,
$u(\{i\},i) = t_{1i}$.

Sin considerar aspectos temporales, las condiciones necesarias y suficientes para la factibilidad de este problema son las siguientes $\Sigma_{i=2}^{n1} q_i \leq capac$ y $\Sigma_{i=n1+1}^{n} q_i \leq capac$.

Además de cumplir los horarios de entrega o recogida en las diferentes localizaciones la solución obtenida ha de tener en cuenta la capacidad o espacio disponible del vehículo en cada momento. Por consiguiente, el algoritmo anterior es fácilmente adaptable a este nuevo modelo, incluyendo la función el(S,j) en la formulación recursiva planteada en la sección 2.2. del capítulo 4.

Esta función el(S,j) en el espacio de estados relajados lleva asociada la etiqueta el(k,j) que se incorpora fácilmente en el procedimiento CALCULOSEUDORUTA de la forma siguiente:

Paso 1.- Inicializar variables

Para k=1,...,K, y $\forall j \in X$ poner el(k,j) = espacio disponible en el vehículo después de recorrer $R_1(\alpha)$.

Paso 4.- Cambio de etiquetas

si $j \leq n1$ poner elw = el(estado,indice)+q_j,

en caso contrario poner elw = el(estado,indice)-q_j ;

si elw < 0 redefinir w = ∞;

si w < v(i,j) poner el(i,j) = elw.

De esta forma se asegura que los procesos de ramificación y acotación consideran también el espacio libre que queda en cada momento en el vehículo y se rechazan las ramas que darían soluciones infactibles por este motivo.

1.2.- ADAPTACION AL PDPTW CON UN SOLO VEHICULO: CASO GENERAL

Considérese la formulación del PDPTW dada en el capítulo 3 con un sólo vehículo. La adaptación del algoritmo para este modelo, es similar a la expuesta en el apartado anterior para la variante del TSPTW con carga y descarga; en este caso se ha de asegurar además para cada cliente i que el punto de recogida i^+ sea siempre anterior al correspondiente punto de descarga i^- en la ruta final obtenida.

Para ello se incorporan dos procedimientos para *filtrar* los arcos que dan soluciones factibles en este modelo, basándonos en el teorema 2 expuesto en el trabajo de Kalantari y otros, (1.985). Estos procedimientos son: FILTROGENERAL que se ejecuta en el procedimiento PRINCIPAL, y FILTROENRAMA que se ejecuta en el procedimiento RECURSIONGENERAL. A continuación se describe cada uno de ellos:

Procedimiento FILTROGENERAL

Para cada cliente i, i = 2, ...,n, redefinir: $d_{1i^-} = \infty, d_{i^+1} = \infty, d_{i^-i^+} = \infty$.

Procedimiento FILTROENRAMA:

Para cada arco (a,b) de R(α):

- Si (a,b) es de la forma (i^+, j^-), con i ≠ j, redefinir: $d_{j^+i^-} = \infty, d_{i^-j^+} = \infty, d_{1i^+} = \infty, d_{j^-1} = \infty$.
- Si (a,b) es de la forma (i^-, j^+), con i ≠ j, redefinir: $d_{i^+j^-} = \infty, d_{j^-i^+} = \infty, d_{j^+i^+} = \infty, d_{j^-i^-} = \infty$.
- Si (a,b) es de la forma (i^+, j^+), con i ≠ j, redefinir: $d_{j^-i^+} = \infty$.
- Si (a,b) es de la forma (i^-, j^-), con i ≠ j, redefinir: $d_{j^-i^+} = \infty$.

Estos procedimientos se incorporan, junto con las adaptaciones señaladas en el apartado anterior, al algoritmo exacto para el TSPTW de la forma siguiente:

Procedimiento PRINCIPAL

Paso 1.: Leer datos iniciales;

Ejecutar FILTROGENERAL.

Procedimiento RECURSIONGENERAL

Paso 0.: Definir $d1_{ij} = d_{ij}$, $\forall (i,j) \in N$.

Ejecutar FILTROENRAMA.

Paso 4.- Cambio de etiquetas

si j ∈ N^- poner elw = el(estado,indice)+q_j, si j ∈ N^+ poner elw = el(estado,indice)-q_j;
si elw < 0 redefinir w = ∞;
si w < v(i,j) poner el(i,j) = elw.

Paso 7.: Definir $d_{ij} = dl_{ij}$, $\forall(i,j) \in N$.
Parar.

En el procedimiento CALCULOSEUDORUTA se añade una pequeña modificación referente a la nomenclatura de los conjuntos: en este caso se sustituye el conjunto de nodos {2,3,...,n1} por N', y el conjunto {n1+1, n1+2,...,n} por N''.

1.3.- ADAPTACIONES AL PDP Y PDPTW CON UN SOLO VEHICULO: SISTEMAS DE DESCARGA LIFO

Supóngase el modelo anterior en el que además se exige que la mercancía que se descarga en cada momento sea la última que ha entrado en el vehículo. Esta condición se impone en muchos modelos *reales* como, por ejemplo, distribución de gases industriales, o en problemas en los que el tiempo de descarga debe ser controlado. Para diseñar filtros eficaces, se han desarrollado unos resultados teóricos que generalizan los obtenidos por Kalantari y otros, (1.985) para este tipo de problemas.

A continuación se describe la adaptación al PDPTW con este sistema de descarga del algoritmo del TSPTW descrito en el capítulo 4, y posteriormente la adaptación al PDP con el mismo sistema del algoritmo de Little para el TSP.

Aunque básicamente los filtros desarrollados son los mismos, en este último caso se añade alguna modificación más como se verá en la sección 1.3.1.3. En ambos casos se han simulado diversos problemas para comparar el efecto de los diferentes filtros y modificaciones.

1.3.1.- ADAPTACION AL PDPTW CON SISTEMA LIFO DEL ALGORITMO EXACTO DEL TSPTW DESCRITO EN EL CAPITULO 4

1.3.1.1.- Aspectos Teóricos

Definición.-

Sea T una ruta factible del PDPTW con un sólo vehículo con sistema de descarga LIFO. Se define la relación de precedencia:

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

Obviamente, la relación ***ant*** va asociada a la ruta T elegida.

Teorema 1.-

Para toda ruta factible T del PDPTW con sistemas de descarga LIFO, la relación ***ant*** verifica las siguientes propiedades:

- (No Reflexiva) $\forall a \in N^+ \cup N^-$, no se verifica a ***ant*** a;

- (No Simétrica) $\forall a, b \in N^+ \cup N^-$, si a ***ant*** b, entonces no se verifica b ***ant*** a;

- Transitiva: $\forall a, b, c \in N^+ \cup N^-$, si a ***ant*** b y b ***ant*** c entonces a ***ant*** c.

Demostración: Trivial.-

El siguiente teorema generaliza el teorema 2 propuesto por Kalantari y otros (1.985):

Teorema 2.-

Para toda ruta factible T del PDPTW con sistemas de descarga LIFO se verifican las siguientes propiedades:

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

b) $\forall a, b, c, d \in N^+ \cup N^-$ tales que a ***ant*** b, c ***ant*** d, con $a \neq c, d$, $b \neq c, d$:
(b1) - Si (a,d) $\in$ T entonces (c,b), (b,c), (1,a) (d,1) $\notin$ T;
(b2) - Si (b,c) $\in$ T entonces (a,d), (d,a), (c,a) (d,b) $\notin$ T;
(b3) - Si (a,c) $\in$ T entonces (d,a) $\notin$ T;
(b4) - Si (b,d) $\in$ T entonces (d,a) $\notin$ T.

c) $\forall a, b, c \in N^+ \cup N^-$ tales que a ***ant*** b, c ***ant*** b, y a $\neq$ c:
- Si (a,b) $\in$ T entonces (1,a) $\notin$ T.

d) $\forall a, b, d \in N^+ \cup N^-$ tales que a ***ant*** b, a ***ant*** d, y b $\neq$ d:
- Si (a,d) $\in$ T entonces (d,1) $\notin$ T

e) $\forall a, b, d \in N^+ \cup N^-$ tales que a ***ant*** b, b ***ant*** d :
(e1) - (a,d) $\notin$ T;
(e2) - Si (a,b) $\in$ T entonces (d,a) $\notin$ T;
(e3) - Si (b,d) $\in$ T entonces (d,a) $\notin$ T.

Demostración.-

La demostración del apartado a) 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 demuestra la no existencia de los arcos (b,a) y (a,1). Para demostrar el apartado *(b1)* se observa que la existencia simultánea del arco (a,d) con cualquiera de los otros: (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 3.-

Sea T una ruta factible entonces:

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

Demostracion.-

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

$$1 - \ldots - i^{+} - \ldots - j^{+} - \ldots - i^{-} - \ldots - j^{-} - \ldots - 1;$$

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

Teorema 4.-

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

Demostración.-

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

Estos cuatro teoremas junto con la definición de la relación ***ant*** son la base del desarrollo de los filtros que se exponen a continuación: El primer filtro se inserta en el procedimiento principal, tiene como objeto impedir la elección de los arcos (i^{+}, j^{-}) y además registra las relaciones de precedencia determinadas por el apartado a) del teorema 3. El segundo filtro, que se inserta en la exploración de cada rama, actúa de la forma siguiente: Inicialmente *detecta* y registra todas las precedencias surgidas a partir de la ruta parcial, $R_1(\alpha)$ y $R_2(\alpha)$, utilizando para ello los apartados b) y c) del teorema 3 y la propiedad transitiva del teorema 1. A continuación se examina si el conjunto de precedencias obtenido es *compatible* con la formación de una ruta factible. Finalmente, con las precedencias registradas, se utiliza el teorema 2 para impedir la elección de arcos que no pueden estar incluidos en una solución factible.

1.3.1.2.- Descripción de los procedimientos

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

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

y 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;

Procedimiento FILTROGENERAL1.

Paso 1.: Para i = 2,3,..,n, poner $d_{1i^-} = \infty, d_{i^+1} = \infty, d_{i^-i^+} = \infty$.
Para i, j = 2,3,...,n, i ≠ j, poner $d_{i^+j^-} = \infty$,

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

Procedimiento FILTROENRAMA1.

Paso 0.: (Inicializar variables)

Poner *nasc* = número de nodos de $R_1(\alpha)$;

Poner *ndes* = número de nodos de $R_2(\alpha)$;

$\forall k, k1 \in N^+ \cup N^-, k \neq k1$, poner *presente*(k,k1) = FALSE, *prefpres*(k,k1) = FALSE;

$\forall k \in N^+ \cup N^-$, poner *numorden*(k) = nasc+1.

Paso 1.: $\forall (i,j) \in R_1(\alpha) \cup R_2(\alpha)$:

Poner *presente*(i,j) = TRUE;

Poner *numorden*(i) = posición que ocupa el nodo i en la ruta parcial.

Paso 2.: (Preferencias iniciales)

Poner *kpref* = n-1;

Para i, j = 2, 3,..., n; i ≠ j:

Si *numorden*(i^+) < *numorden*(j^+) y *numorden*(j^+) < *numorden*(i^-) entonces:

poner *kpref* = *kpref*+1,

poner *vectorpref*(*kpref*) = (j^-, i^-);

Si *numorden*(i^-) < *numorden*(j^-) y *numorden*(j^+) < *numorden*(i^-) entonces:

poner *kpref* = *kpref*+1,

poner *vectorpref*(*kpref*) = (j^+, i^+);

Para k =1,...,kpref, poner *prefpres*(*vectorpref*(k).*inicio*, *vectorpref*(k).*final*) = TRUE.

Paso 3.: (Propiedad transitiva)

Poner *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,

si no ir al paso 4.

Paso 4.: (Compatibilidad de preferencias)

prefcompat = TRUE;

Para $i \in N^+ \cup N^-$, poner *prefcompat* = *prefcompat* and (not *prefpres*(*i*,*i*));

Si *prefcompat* ir al paso 5,

si no parar.

Paso 5.: (Filtrado)

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

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

$d_{1b} = \infty, d_{a1} = \infty, d_{ba} = \infty$,

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

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

Si *presente*(a,d) hacer

$d_{cb} = \infty, d_{bc} = \infty$,

si a ≠ c poner $d_{1a} = \infty$,

si b ≠ d poner $d_{d1} = \infty$,

si b = c poner *prefcompat* = FALSE,

Si *presente*(b,c) poner

$d_{ad} = \infty, d_{da} = \infty, d_{ca} = \infty, d_{db} = \infty$,

Si *presente*(a,c) poner $d_{da} = \infty$,

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

Paso 6.: Parar.

Estos procedimientos se incorporan, al algoritmo exacto para el TSPTW de la forma siguiente:

Procedimiento PRINCIPAL

Paso 1.: Leer datos iniciales.

Ejecutar FILTROGENERAL1.

Procedimiento RECURSIONGENERAL

Paso 0.: $\forall(i,j) \in N$, definir $d1_{ij} = d_{ij}$.

Ejecutar FILTROENRAMA1.

Si no *prefcompat* parar, en caso contrario continuar.

Paso 7.: $\forall(i,j) \in N$, poner $d_{ij} = d1_{ij}$.

Parar.

1.3.1.3.- Resultados Computacionales

A continuación se muestran los resultados de una serie de problemas simulados para comparar el tiempo de computación obtenido por el algoritmo adaptado anterior cuando se usan los dos filtros y cuando sólo se usa el filtro general.

Se han simulado 10 problemas para cada uno de los siguientes niveles: 4 clientes, 5, 6 y 7, (correspondientes a 9 nodos, 11, 13 y 15). Aunque el tamaño de los problemas simulados sea pequeño se aprecia el efecto de los filtros en los tiempos de cálculo.

El siguiente cuadro muestra los estadísticos correspondientes a los tiempos de computación (en segundos) obtenidos:

Número de Clientes		**n = 4**	**n = 5**	**n = 6**	**n = 7**
Todos los Filtros	*Media*	2.660	9.714	106.255	303.555
	Desviación	2.335	8.766	122.133	222.162
	Mínimo	0.66	1.04	5.66	23.34
	Máximo	8.67	28.23	417.32	649.71
Sólo Filtro General	*Media*	2.805	14.612	124.823	508.825
	Desviación	2.459	19.443	134.972	437.263
	Mínimo	0.66	1.10	5.65	24.39
	Máximo	9.12	69.04	472.75	1558.40

Aunque estos resultados, según se comentó en la introducción de este capítulo, indican que aún se debe trabajar bastante en este tipo de problemas, también muestran el efecto positivo en el tiempo de computación de los filtros desarrollados.

1.3.2.- ADAPTACION AL PDP CON SISTEMA LIFO DEL ALGORITMO EXACTO DE LITTLE PARA EL TSP

Las filtros desarrollados en esta sección son los mismos que en el apartado anterior; sin embargo se incorpora algún aspecto teórico más y especialmente una alteración en el proceso de selección de los arcos que se añaden en cada vértice del árbol de búsqueda.

1.3.2.1.- Aspectos Teóricos

Definición y Notaciones.-

Se define una cadena C como una secuencia de arcos de la forma: (a,b), (b,c), (c,d), ...,(m,n). Se denota por C(r) a la cadena que contiene al nodo r. A la concatenación de las cadenas C_1 y C_2 lo denotamos por (C_1, C_2). Si T es una ruta cerrada o tour, se denota T ∋ C si la cadena C forma parte de ella.

Teorema 5.-

Sea T una ruta factible para el PDP con un solo vehículo con sistema LIFO, ***ant*** la relación definida en 1.3.1.1. y a,b dos nodos cualesquiera, con C(a) y C(b) ∈ T. Si se verifica una de las siguientes condiciones entonces a ***ant*** b :

(*i*) - C(a) = C(b), C(a) ≠ C(1) y a aparece antes que b en C(a);
(*ii*) - C(a) = C(1), C(a) ≠ C(b) y a aparece después que 1 en C(a);
(*iii*) - C(b) = C(1), C(a) ≠ C(b) y b aparece antes que 1 en C(b);
(*iv*) - 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.

Demostración.- Trivial

El siguiente teorema generaliza y modifica el teorema 3 desarrollado en el trabajo de Kalantari y otros, (1.985).

Teorema 6.-

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

a) $\forall a, b \in N^+ \cup N^-$ tales que a ***ant*** b:
   (*a1*) - Si C(a), C(b) $\neq$ C(1) entonces (C(b),C(a)) $\notin$ T;
   (*a2*) - Si C(1), C(b) $\neq$ C(a) entonces (C(1),C(b)) $\notin$ T;
   (*a3*) - Si C(1), C(a) $\neq$ C(b) entonces (C(a),C(1)) $\notin$ T.

b) $\forall a, b, c, d \in N^+ \cup N^-$ tales que a ***ant*** b, c ***ant*** d con $a \neq c, d$; $b \neq c, d$:
   - Si C(a) = C(d), C(a),C(b),C(c) $\neq$ C(1), C(a) $\neq$ C(b) y C(c) $\neq$ C(d),
     entonces (C(b),C(c)), (C(c),C(b)) $\notin$ T.

c) a ***ant*** b y b ***ant*** d:
   - Si C(a), C(b) $\neq$ C(d) y C(a) $\neq$ C(b) entonces (C(a), C(d)) $\notin$ T.

Demostración.- Trivial a partir de la definición de la relación ***ant*** y del teorema 2.

1.3.2.2.- Descripción de los Procedimientos de Filtrado

Además de las variables definidas en el apartado 1.3.1. se definen las siguientes:

primero y ultimo : vectores enteros auxiliares; primero(i) y ultimo (i) indican respectivamente el primer y último elemento de la cadena C(i);

D1 y **D*** : Matrices de distancias auxiliares.

A continuación se describen las modificaciones del Procedimiento FILTROENRAMA respecto a la adaptación anterior y los Procedimientos EXPLORACION(R(α), F(α)) y PRINCIPAL respecto del algoritmo de Little descrito en el capítulo 1.

Procedimiento FILTROENRAMA.

Paso 0.: (Inicializar variables)

$\forall k, k1 \in N^+ \cup N^-, k \neq k1$, poner *presente*(k,k1) = FALSE, *prefpres*(k,k1) = FALSE;
$\forall (i,j) \in R(\alpha)$, poner *presente*(i,j) = TRUE .

Paso 1.: $\forall k \in N^+ \cup N^- \cup \{1\}$, poner primero(k) = primer elemento de C(k);
$\forall k \in N^+ \cup N^- \cup \{1\}$, poner ultimo(k) = último elemento de C(k);

Paso 2.: (Preferencias iniciales)

Poner *kpref* = n-1;
Para i, j = 2, 3,..., n; i ≠ j:
Si i^+ ***ant*** j^+ y j^+ ***ant*** i^- entonces:
poner *kpref* = *kpref*+1,
poner *vectorpref*(*kpref*) = (j^-, i^-);
Si i^- ***ant*** j^- y j^+ ***ant*** i^- entonces:
poner *kpref* = *kpref*+1,
poner *vectorpref*(*kpref*) = (j^+, i^+);
Para k =1,..., kpref, poner *prefpres*(*vectorpref*(k).*inicio*, *vectorpref*(k).*final*) = TRUE.

Paso 6.: (Filtrado - Teorema 6)

Para k = 1,2,...,*kpref*, hacer:
a = *vectorpref*(k).*inicio*, b = *vectorpref*(k).*final*,
Si C(a), C(b) ≠ C(1), poner $d1_{ultimo(b),primero(a)} = \infty$;
Si C(1), C(b) ≠ C(a), poner $d1_{ultimo(1),primero(b)} = \infty$;
Si C(1), C(a) ≠ C(b), poner $d1_{ultimo(a),primero(1)} = \infty$;
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
$d1_{ultimo(b),primero(c)} = \infty, d1_{ultimo(c),primero(b)} = \infty$;
Si b = c, C(a),C(b) ≠ C(d), C(a) ≠ C(b), entonces:
$d1_{ultimo(a),primero(d)} = \infty$.

Paso 7.: Parar.

Procedimiento EXPLORACION(R(α), F(α))

Si los arcos de R(α) forman una ruta:

Poner *Disttotal* = suma de los arcos de 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, (los arcos de R(α) no forman una ruta), hacer:

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

Poner en **D1** la matriz **D** sin las filas correspondientes a los nodos iniciales de los arcos de R(α) y sin las columnas de los nodos finales de dichos arcos.

Ejecutar FILTROENRAMA; (*2*)

$\forall(i,j) \in F(\alpha)$ poner $d1_{ij} = \infty$;

Obtener la reducción **D*** de la matriz **D1** (sección 2.1. del capítulo 1).

Poner $cotainf = \Sigma_{(i,j)\in R(\alpha)} d1_{ij} + \Sigma_{i\in filas} p_i + \Sigma_{j\in colum} q_j$ (donde *filas* y *colum* son respectivamente los conjuntos de filas y columnas de la matriz **D1**).

Si *cotainf* < *Distminima* y *prefcompat* entonces hacer:

$\forall(i,j)/d1_{ij} = 0$, poner $p_{ij} = \min_{k\neq j} d^*_{ik} + \min_{k\neq i} d^*_{kj}$.

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

Poner R(α*) = R(α) ∪(i*,j*);

{Prevención de ciclos}

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);

{Fin de Prevención}

Ejecutar EXPLORACION(R(α*), F(α*));

Si *Cotainf* + p_{i*j*} < *Distminima* entonces, hacer

R(α**) = R(α);

F(α**) = F(α) $\cup$ (i*,j*);

Ejecutar EXPLORACION(R(α**), F(α**)).

Parar.

Procedimiento PRINCIPAL

Leer datos iniciales: n y **D**
Poner *Distminima* = ∞.
Ejecutar FILTROGENERAL (*1*)
Ejecutar EXPLORACION($\varnothing$,$\varnothing$)

1.3.2.3. Variación en el criterio de elección del arco que se añade

Además de la incorporación de los procedimientos de filtrado se propone una pequeña modificación en la elección del arco (i*,j*) con objeto de aumentar la eficacia de los filtros en exploraciones posteriores. La idea básica es sencilla: se restringe la búsqueda de este arco (i*,j*) únicamente a los arcos que finalizen en el primer nodo de la cadena donde está el nodo 1 y a los arcos que comienzan en el último nodo de dicha cadena, en vez de buscarlos entre todos los 0 de la matriz reducida **D***.

Utilizando la variable auxiliar

penal : mayor valor de las penalizaciones p_{i*j*} de los ceros encontrados;

el procedimiento de elección del arco (i*,j*) es:

Procedimiento NUEVAELECCION;

Paso 1.: Poner $penal = \infty$.

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

$penal = p_{i,primero(1)}$,

i* = i

j* = primero(i).

Paso 3.: $\forall j \in colum$, si $d^*_{primero(1),j} = 0$ y si $p_{ultimo(1),j} > penal$ entonces hacer:

$penal = p_{ultimo(1),j}$,

j* = j

i* = ultimo(i).

Paso 4.: Parar.

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.

1.3.2.4.- Resultados Computacionales

Este apartado recoge 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 4, 5, 6, 7 y 8, clientes (10 para cada número de clientes). A cada problema se le aplicaron las cuatro variantes siguientes:

VARIANTE A: En esta variante no se incorpora ningún procedimiento. Sencillamente cuando llega a una solución se comprueba si es factible en cuyo caso se registra. Es decir en el

procedimiento EXPLORACION(R(α),F(α)) no se ejecutan las líneas (*1*) y (*2*), y en el procedimiento PRINCIPAL no se ejecuta la línea (*1*).

VARIANTE B: 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 sí se ejecuta la línea (*1*).

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

A continuación se muestra un cuadro que resume los resultados obtenidos (tiempo en segundos) y la gráfica correspondiente:

Nº de clientes		*VARIANTE A*	*VARIANTE B*	*VARIANTE C*	*VARIANTE D*
4	**Mínimo**	0.06	0.00	0.00	0.00
	Media	9.511	0.027	0.056	0.061
	Máximo	26.69	0.06	0.11	0.11
	Desviación	9.018	0.027	0.035	0.030
5	**Mínimo**	39.71	0.00	0.06	0.11
	Media	398.254	0.311	0.504	0.406
	Máximo	1684.07	1.54	2.30	0.98
	Desviación	527.211	0.460	0.704	0.277

Nº de clientes		*VARIANTE A*	*VARIANTE B*	*VARIANTE C*	*VARIANTE D*
6	**Mínimo**	-	0.11	0.17	0.33
	Media	-	2.168	6.044	4.761
	Máximo	-	4.89	14.50	13.51
	Desviación	-	1.616	5.019	4.515
7	**Mínimo**	-	0.28	1.54	2.08
	Media	-	52.390	64.834	17.980
	Máximo	-	392.23	412.87	60.69
	Desviación	-	114.863	118.188	18.514
8	**Mínimo**	-	1.43	5.26	1.59
	Media	-	369.409	570.769	88.773
	Máximo	-	2293.63	2641.89	221.51
	Desviación	-	656.691	728.605	66.695

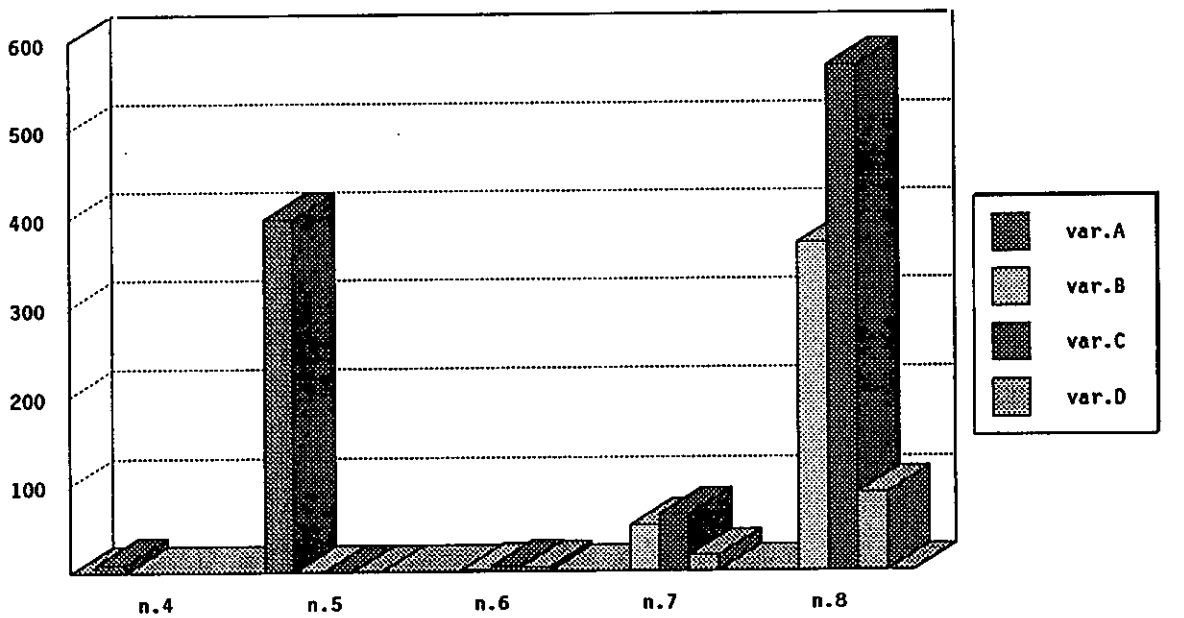


Figura 30. Evolución de los tiempos medios para las variantes A, B, C, D.

1.3.2.5.- Conclusiones, Extensiones y Mejoras

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

- El tiempo de computación cuando se aplica directamente el algoritmo de Little es excesivo incluso para problemas de tamaño pequeño; la aplicación de los procedimientos y modificaciones, como los descritos en el apartado 3, es indispensable para la reducción del tiempo de computación.

- La evolución de los tiempos de computación de las variantes B y C son similares, en cualquier caso son mayores los de la variante C; por consiguiente, aparentemente, el efecto del procedimiento FILTROENRAMA es negativo. La razón es la siguiente: dicho procedimiento detecta y registra relaciones de precedencia a partir de los apartados b) y c) del Teorema 3. Para aplicar este teorema 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 Teorema 5), 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 D, que hace que sólo 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 el teorema 3, con lo que ya se hace *rentable* la utilización del procedimiento FILTROENRAMA.

- Como consecuencia de lo anterior la variante D 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 la variante D es más rápida que las otras.

Estas variantes no consideran las restricciones de carga en el problema; sin embargo, la formación de una sola cadena en torno al nodo inicial 1 hace que esta restricción se pueda incorporar fácilmente a la variante D. En la exploración de cada vértice se añade un pequeño procedimiento que, comenzando en el nodo 1, calcula 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.

Algunas modificaciones, con vistas a bajar el tiempo de computación, que se proponen para estudios posteriores son las siguientes: Partir de una solución inicial obtenida por algún heurístico o bien generar una cota superior en cada exploración.

2.- UN MODELO DE RUTAS CON PARTICION DE DEMANDAS

A veces, en muchas actividades de distribución, no es necesario que la mercancía a distribuir a cada localización tenga que ser enviada por un solo vehículo. Supóngase casos como el planteado en el ejemplo práctico del capítulo 5, en el que la mercancía que se envía desde la fábrica central a cada una de las delegaciones se distribuye en forma de *carros*. Cada uno de estos carros se considera indivisible, pero a una delegación concreta pueden llegar carros distribuidos en varios vehículos. Es decir, en este caso se podría *relajar* el modelo clásico de la restricción de que cada localización debe ser *servida* por un solo vehículo.

El liberar o relajar el modelo de dicha restricción puede ser interesante, sobretodo en aquellas actividades en que se intenta principalmente minimizar el número de vehículos utilizados. Supóngase el siguiente ejemplo sencillo:

Desde una fábrica central O, se deben distribuir a 3 delegaciones A,B y C, 6 carros a cada una de ellas; para ello se dispone de una serie de vehículos de 10 carros de capacidad cada uno. Con los modelos del VRP la única solución factible serían 3 rutas:

O - A- O, O - B - O, y O - C - O;

sin embargo, *relajando* la anterior restricción, existirían soluciones factibles utilizando menos vehículos; por ejemplo:

O - A(6) - B(4) - O, y O - B(2) - C(6) - O;

(entre paréntesis se señalan el número de carros que se envían a cada delegación).

2.1.- FORMULACION

El problema así planteado, ya no corresponde al modelo del VRP clásico. Puede formularse de la siguiente forma;

sean:

N= {1,2,...,n} el conjunto de localizaciones, incluida la inicial;

M : Número máximo de vehículos

capac : Capacidad de cada vehículo expresado en número de carros;

b_j : Número de carros demandados por la delegación j, j = 2,...n;

x_{ij} : Número de carros transportados por el vehículo i a la localización j:

para i = 1,...,M, y j = 2,...n;

y_i : Coste de la ruta óptima que, partiendo del origen, recorre las localizaciones j tales que $x_{ij} > 0$;

$N_i = \{ j \in N\text{-}\{1\} / x_{ij} > 0\} \cup \{1\}$;

se trata de

$$\text{minimizar } \Sigma_{i=1}^{M} y_i$$

sujeto a:

$$\Sigma_{j=2}^{n} x_{ij} \leq capac, \quad i = 1,...,M;$$

$$\Sigma_{i=1}^{M} x_{ij} = b_j, \quad j = 2,...,n;$$

$$x_{ij} \geq 0, \quad i = 1,....,M; \quad j = 2,...,n;$$

donde $y_k = 0$ si $N_k = \{1\}$; en caso contrario

$$y_k = \min \sum_{i \in N_k} \sum_{j \in N_k} d_{ij} z_{ij}$$

sujeto a:

$$\Sigma_{i \in N_k} z_{ij} = 1, \quad j \in N_k;$$

$$\Sigma_{j \in N_k} z_{ij} = 1, \quad i \in N_k;$$

$$u_i - u_j + [N_k] \cdot z_{ij} \leq [N_k] - 1, \quad i,j \in N_k - \{1\}, i \neq j;$$

$$z_{ij} \in \{0,1\}, \quad i,j \in N_k,$$

$$u_i \in Z^+, \quad i \in N_k - \{1\}.$$

para cada k = 1,..,M; en este segundo caso si se desea se puede incorporar en la función objetivo el coste del vehículo.

Se pueden incorporar las ventanas de tiempo a este modelo redefiniendo adecuadamente el problema de obtener y_k.

2.2.- ADAPTACION DEL ALGORITMO DE BEASLEY

Para resolver este tipo de problemas se han probado adaptaciones de algoritmos para el VRP y para el VRPTW; por ejemplo el algoritmo de Beasley descrito en el Capítulo 2 puede ser modificado fácilmente para obtener un heurístico eficaz para el anterior modelo. Esta modificación consiste en lo siguiente:

- Formar una gran ruta inicial con las localizaciones originales;

- Manteniendo el orden establecido por esta gran ruta, considerar cada unidad de mercancía indivisible (carro, palé,...), como puntos individuales independientes situados en la localización correspondiente. Enumerar los puntos según el orden obtenido en la gran ruta inicial. El número de puntos que forman esta nueva gran ruta será $1+\Sigma_{j=2}^{n} b_j$.

- Definir la red completa formada por estos puntos y con matriz de distancias c_{ij} según se definió en el algoritmo original

- Obtener el camino mínimo en esta red desde el nodo 1 hasta el $1+\Sigma_{j=2}^{n} b_j$. Los puntos que forman este camino mínimo determinarán el conjunto de rutas que forman la solución.

2.3.- COMPARACION DE LOS MODELOS MEDIANTE UN EJEMPLO REAL

En esta sección se muestra el resultado de aplicar la adaptación del algoritmo de Beasley descrito en el subapartado anterior al problema real descrito en el capítulo 5

Número de Rutas generadas: 12.

Número Total de Kms. recorridos: 10.444.

	Puntos Visitados (nº de carros transportados)	Distancia Recorrida (Kms.)
Ruta 1	1 - 12 (6) - 2 (6) - 3 (16) - 1	1.309
Ruta 2	1 - 12 (2) - 26 (8) - 27 (14) - 1	1.086
Ruta 3	1 - 25 (28) -1	572
Ruta 4	1 - 15 (9)- 14 (15) - 13 (4) - 1	344
Ruta 5	1 - 9 (17) - 15 (5) - 1	212
Ruta 6	1 - 16 (3) - 17 (3) - 19 (8) - 21 (14)- 1	786
Ruta 7	1 - 23 (13) - 22 (7) - 24 (8) - 1	1.184
Ruta 8	1 - 18 (6) - 20 (17) - 22 (5) - 1	1.165
Ruta 9	1 - 8 (16) - 10 (4) - 11 (7) - 18 (1) - 1	830
Ruta 10	1 - 7 (18) - 8 (10) - 1	387
Ruta 11	1 - 7 (10) - 6 (8) - 5 (10) - 1	1.276
Ruta 12	1 - 5 (18) - 4 (4) - 3 (6) - 1	1.293

Una comparación del modelo de partición de demandas con el modelo VRPTW clásico muestra que la solución para el VRPTW con los mismos datos y con el mismo algoritmo tiene 16

rutas y 11.678 Kms. La mejor solución conseguida por la variante TIEMPO MAXIMO DE COMPUTACION tiene 13 vehículos y 10.637 kms. Por consiguiente es claro que este modelo puede ser considerado cuando menos interesante para el tratamiento de los problemas de rutas.

APENDICE : MODELIZACION DE DIVERSOS PROBLEMAS DE MATEMATICA COMBINATORIA

1.- PROBLEMAS DE ASIGNACION

1.1.- PROBLEMA DE ASIGNACION LINEAL GENERALIZADA

Sea el problema en el que n tareas deben ser realizadas por m máquinas con la condición de que cada tarea debe hacerse en una máquina, y el tiempo máximo de uso de cada máquina es b_i ; sea a_{ij} el tiempo que emplea la máquina i en realizar la tarea j y c_{ij} el coste en que se incurre cuando la máquina i realiza la tarea j, el problema se formula como:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

sujeto a:

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1,...,n;$$

$$\sum_{j=1}^{n} a_{ij} \cdot x_{ij} \leq b_i, \quad i = 1,...,m;$$

$$x_{ij} \in \{0,1\}.$$

Métodos de solución para este problema como el de Ross y Soland, (1.975), y el de Fisher y otros, (1.986), han sido programados para la elaboración del algoritmo de Fisher y Jaikumar para el VRP descrito en el capítulo 2.

1.2.- PROBLEMA DEL EMPAREJAMIENTO PERFECTO CON COSTES

Dado un conjunto de n items (n par) si se emparejan los items i,j se tiene un coste c_{ij}; el problema trata de emparejar los n items de forma que el coste total sea mínimo. Es decir

$$\text{minimizar} \sum_{\substack{i,j=1 \\ i<j}}^{n} c_{ij} \cdot x_{ij}$$

sujeto a:

$$\sum_{\substack{i=1 \\ i<j}}^{n} x_{ij} + \sum_{\substack{i=1 \\ i>j}}^{n} x_{ji} = 1, \quad j = 1,2,...,n;$$

$$x_{ij} \in \{0,1\}, \quad i,j = 1,2,...,n; i<j;$$

donde

$$x_{ij} = \begin{cases} 1, \text{ si i se empareja con j,} \\ \\ 0, \text{ en caso contrario,} \end{cases} \qquad i,j = 1,2,...,n; i<j.$$

2.- PROBLEMAS DE EMPAQUETAMIENTO Y CUBRIMIENTO

2.1.- PROBLEMA DE LA MOCHILA (KP)

Se dispone de n objetos con peso p_i y valor v_i , i = 1,...,n, y una mochila que soporta un peso máximo P. El problema consiste en elegir el subconjunto de objetos de mayor valor sin sobrepasar el peso P. Puede haber restricciones adicionales y varios objetos de cada clase.

2.2.- PROBLEMAS DE CUBRIMIENTO

Los problemas expuestos en esta sección, se formulan en términos de conjuntos. Para ello se define:

M = {1,2,...,m} un conjunto base, y

$\mathbb{P} = \{P_1, P_2, ..., P_n\}$ una colección de subconjuntos de M, es decir

$$P_j \subseteq M, \forall j = 1,2,...,n.$$

2.2.1.- PROBLEMA DE EMPAQUETAMIENTO DE CONJUNTOS (SP)

Se dice que una subfamilia $\mathbb{L} \subseteq \mathbb{P}$ es un empaquetamiento del conjunto M, si todos los elementos de $\mathbb{L}$ son disjuntos dos a dos. Si c_i es el beneficio obtenido por usar el elemento P_i, i = 1, 2, ..., n, se trata de encontrar el empaquetamiento de mayor beneficio.

2.2.2.- PROBLEMA DE PARTICION DE CONJUNTOS (SPP)

Se dice que una subfamilia $\mathbb{L} \subseteq \mathbb{P}$ es una partición de M, si es un empaquetamiento y cada elemento de M está en un único subconjunto de $\mathbb{L}$. Si c_i es el coste de utilizar el elemento P_i, i = 1,2,...,m, se trata de encontrar la partición de mínimo coste.

2.2.3.- PROBLEMA DE CUBRIMIENTO DE CONJUNTOS (SCP)

Se dice que una subfamilia $\mathbb{L} \subseteq \mathbb{P}$ es un cubrimiento de M, si cada elemento de M está al menos en un subconjunto de $\mathbb{L}$. Sea c_i el coste de utilizar el elemento P_i, i = 1,2,...,m, se trata de encontrar el cubrimiento de mínimo coste.

3.- PROBLEMAS DE RUTAS

3.1.- PROBLEMA DEL CAMINO MINIMO

Planteamiento del problema

Dada una red G = (N,S), y unos costes (o longitudes) c_{ij} para cada arco (i,j) en S, se trata de encontrar el camino mínimo (más corto) entre dos nodos de la red G. Pueden considerarse los problemas de encontrar el camino mínimo entre un par de nodos concretos, de un nodo fijo a todos los demás o entre cada par de nodos.

Técnicas de solución

Dada la particularidad de este problema existen algoritmos específicos diseñados para él. Entre otras, cabe citar los siguientes: Algoritmo de Ford, Floyd, Dijkstra....

Por su uso a lo largo de esta memoria a continuación se describe el de Dijkstra también llamado 'Algoritmo de etiquetado' que requiere $c_{ij} \geq 0$, $\forall$ i,j. Este algoritmo encuentra el camino mínimo del nodo 1 a cada uno de los demás nodos.

Para cada nodo i, i = 1,...,n, se define $L(i) = (\theta_i, u_i)$, donde u_i es la longitud de la ruta, formada en cada momento, del nodo 1 al i, y θ_i es el nodo inmediatamente anterior en dicha ruta.

Paso Inicial

Poner P ={1}, T={2,...,n}, $u_1 = 0$; Para j=2,...,n, poner $u_j=c_{1j}$; Para j=1,...,n, poner $\theta_j=1$.

Paso I (Designación de una etiqueta permanente)

Buscar $k \in T$ tal que $u_k = \min_{j \in T} u_j$
Poner T = T-{k}, P = P $\cup$ {k}
Si T = $\varnothing$, parar; si no ir al paso II

Paso II (Revisión de etiquetas tentativas)

$\forall j \in T$ si $u_j > u_k + c_{kj}$ poner $\theta_j = k$, $u_j = u_k + c_{kj}$; Ir al paso I.

La longitud del camino mínimo del nodo 1 al j, j=2,...,n es u_j. Para determinar la ruta mínima se utiliza el vector $\theta = (\theta_1, \theta_2, ..., \theta_n)$: el valor de θ_j indica el predecesor del nodo j en la ruta óptima.

3.2.- VARIANTES DEL TSP

3.2.1.- TSP CON EMBOTELLAMIENTO

El planteamiento es similar al del TSP clásico pero con un 'tiempo' o gasto de estancia en cada ciudad. Como ejemplo de TSP 'con embotellamiento' consideremos una línea de producción con

estaciones de trabajo ordenadas secuencialmente. Hay n trabajos que deben ser realizados, en cualquier orden, uno en cada una de las estaciones. El tiempo requerido para hacer el trabajo j inmediatamente después de hacer el trabajo i es:

$$t_{ij} = d_{ij} + p_j,$$

donde d_{ij} es el tiempo intermedio entre las estaciones i y j, y p_j el tiempo requerido para realizar la tarea j.

Se trata de minimizar el tiempo total de permanencia en la línea de producción, es decir: $\Sigma_{i=1}^{n} \Sigma_{j=1}^{n} \Sigma_{k=1}^{n} t_{ij} \cdot x_{ijk}$ con las restricciones del TSP clásico.

3.2.2.- TSP DEPENDIENTE DEL TIEMPO

Consideremos un TSP en el que hay n períodos de tiempo diferentes y donde d_{ijt} es el coste de ir directamente desde la ciudad i a la j en el período t. Las variables x_{ijt} se definen de forma análoga a las x_{ijk} del problema clásico. El problema se formula como:

$$\text{minimizar } \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} \Sigma_{t=1}^{n} d_{ijt} \cdot x_{ijt}$$

sujeto a las siguientes restricciones:

$$\Sigma_{j=1}^{n} \Sigma_{t=1}^{n} x_{ijt} = 1, \quad i = 1,...,n;$$

$$\Sigma_{i=1}^{n} \Sigma_{j=1}^{n} x_{ijt} = 1, \quad t = 1,...,n;$$

$$\Sigma_{i=1}^{n} \Sigma_{t=1}^{n} x_{ijt} = 1, \quad j = 1,...,n;$$

$$\Sigma_{j=1}^{n} \Sigma_{t=2}^{n} t \cdot x_{ijt} - \Sigma_{j=1}^{n} \Sigma_{t=1}^{n} t \cdot x_{jit} = 1, \quad i = 2,...,n;$$

(el tiempo que hay entre dos llegadas consecutivas es 1)

se supone para simplificar que el viaje comienza y acaba en la ciudad 1.

Otros variantes más recientes del TSP se pueden encontrar en Rote, (1.992), Paletta, (1.992) y especialmente en el libro de Lawler, Lenstra, Rinnoy Kan y Shmoys, (1.985).

3.3.- VARIANTES DEL VRP

3.3.1.- PROBLEMA DE RUTAS CON RESTRICCIONES DE CAPACIDAD

Se define como el VRP con la siguiente característica adicional: cada arco (i,j) tiene asignada una capacidad máxima cap_{ij} de tal forma que el vehículo que recorra este tramo no puede llevar en ese momento una carga superior.

Un amplio tratamiento de este problema puede verse en Benavent, Campos, Corberán y Mota, (1.990) y (1.992).

3.4.- OTROS PROBLEMAS DE RUTAS

3.4.1.- PROBLEMA DEL CARTERO CHINO (CPP)

Sea una red G = (V,E) en la que cada arco $v \in V$ tiene asignado una distancia d_v; sea un subconjunto de arcos $V' \subseteq V$. Se trata de encontrar la ruta de menor distancia que, partiendo de un nodo inicial 1 recorra todos los arcos de V.

Referencias y Bibliografía

1.- ADRABINSKI,A. and SYSLO,M.M.
"Computational Experiments with some Heuristic Algorithms for the Traveling Saleman Problem".
Zastos. Mat., 18, (1.983), 91-95.

2.- AGARWAL,Y.K., MATHUR,K. and SALKIN,M.
"A Set-Partitioning-Based Algorithm for the Vehicle Routing Problem".
Networks, 19, (1.989), 731-749.

3.- AHN,B.H. and SHIN,J.Y.
"Vehicle Roueting with Time Windows and Time-Varying Congestion".
Journ.Operat.Resear.Soci., 42, 5, (1.991), 393-400.

4.- AKINC,U, and SRIKANTH,K.
"Optimal Routing and Process Scheduling for a Mobile Service Facility".
Networks, vol.22, n° 2, (1.992), 163-184.

5.- ALVAREZ,G.
"Racionalización y Optimización del Transporte".
Curso sobre *La Función del Transporte*. Fundación CONFEMETAL. Madrid, 28,29-I-1.992.

6.- ARAGON,A. y PACHECO,J.
"Desarrollo de Algoritmos Heurísticos Mejorados para el TSP".
VI Congreso de ASEPELT-ESPAÑA, GRANADA 1992.

7.- ASSAD,A.A.

"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., (1.988), Nort-Holland, 7-46.

8.- BAGCHI,P.K. and NAG,B.N.

"Dynamic Vehicle Scheduling: An Expert Systems Approach".

Intern.Journal of Physical Distribution & Logistic Management, vol.21, nº 2, (1.991), 10-18.

9.- BAKER,E.K.

"An Exact Algorithm for the Time-Constrained Traveling Salesman Problem".

Oper.Res., 31, (1.983), 938-945.

10.- BAKER,E. and SCAHAFFER,J.

"Computational Experience with Branch Exchange for Vehicle Routing Problems Time Window Constraints".

Am.J.Math.Management Sc., 6, (1.986), 261-300.

11.- BALAS,E. and CHRISTOFIDES,N.

"A Restricted Lagrangean Approach to the Traveling Salesman Problem".

Mathl.Program. 21, (1.981), 19-46.

12.- BALL,M.

"Allocation/Routing: Models and Algorithms".

In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 199-222.

13.- BALL,M. and MAGAZINE,M.

"The Design and Analysis of Heuristics".

Networks, vol.11, nº 2, (1.981), 215-220.

14.- BEASLEY,J.

"Adapting the Savings Algorithm for Varying Inter-Costumer Travel Times".

Omega, vol.9, nº 6, (1.981), 638-639.

15.- BEASLEY.J.

"Route First-Cluster Second Methods for Vehicle Routing".

Omega, vol.11, nº 4, (1.983), 403-408.

16.- BENAVENT,E. CAMPOS,V., CORBERAN,N. y MOTA,E.

" Modelos de Routing".

Seminario sobre Programación Matemática. *I Encuentro Hispano-Luso de I.O.* Barcelona 1.985.

17.- BENAVENT,E. CAMPOS,V., CORBERAN,N. y MOTA,E.

"The Capacitated Arc Routing Problem: A Heuristic Algorithm".

Questiio, 14, 1-3, (1.990), 107-122.

18.- BENAVENT,E. CAMPOS,V., CORBERAN,N. y MOTA,E.

"The Capacitated Arc Routing Problem: Lower Bounds".

Networks, vol.22, nº 7, (1.992), 669-690.

19.- BENTON,W.C. and ROSSETTI,M.D.

"The Vehicle Scheduling Problem with Intermitten Customer Demands".

Computers & Ops.Res. Vol. 19, nº 6, (1.992), 521-532.

20.- BODIN.L.D. and GOLDEN,B.L.

"Classification in Vehicle Routing and Scheduling".

Networks, vol.11, nº 2, (1.981), 97-108.

21.- BODIN,L.D., GOLDEN,B.L., ASSAD,A. and BALL,M.
"Routing and Scheduling of Vehicles and Crews: the State of the Art".
Comput.Oper.Res., 10, (1.983), 63-212.

22.- BODIN,L.D. and KURSH,S.J.
"A Computer-Assisted System for the Routing and Scheduling of Street Sweepers".
Operations Research, vol.26, n° 4, (1.978), 525-537.

23.- BODIN,L.D. and KURSH,S.J.
"A Detailed Description of a Computer System for the Routing and Scheduling of Street Sweepers".
Comput. & Ops.Res., vol., 6, (1.979), 181-198.

24.- BOTT,K. and BOLLOU,R.H.
"Research Perspectives in Vehicle Routing and Scheduling".
Transp.Res.A, vol.20a, n° 3, (1.986), 239-243.

25.- BOYD,S.C. and PULLEYBLANK,W.R.
"Optimizing Over the Subtour polytope of the Traveling Salesman Problem".
Mathematical Programming, 49, (1.991), 163-187.

26.- BOYD,S.C., PULLEYBLANK,W.R. and CORNUEJOLS,G.
"TRAVEL: An Interactive Traveling Salesman Problem Package for the IBM Personal Computer".
Ops.Res.Lett. 6, (1.987), 141-143.

27.- BURKARD,R.E. and VAN DER VEEN,J.A.A.
"Universal Conditions for Algebraic Travelling Salesman Problem to be Efficiently Solvable".
Optimization 22, 5, (1.991), 787-814.

28.- CARLIER,J. and VILLON,P.
"A New Heuristic for the Traveling Salesman Problem".
Recherche Operationnelle/Operations Researsch, vol.24, n° 3, (1.990), 245-253.

29.- CARPENETO,G. and TOTH,P.
"Some new Branching and Bounding Criteria for the Assymmetric Traveling Salesman Problem".
Mgmt.Sci. 26, (1.980), 736-743.

30.- CASCO,D.O., GOLDEN,B.L. and WASIL,E.A.
"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., (1.988), Nort-Holland, 7-46.

31.- CHAPLEAU,L., FERLAND,J.A AND ROUSSEAU,J.M.
"Clustering for Routing in Densely Populated Areas".
European J.Oper.Res., 20, (1.985), 48-57.

32.- CHIEN,T.W.
"Operational Estimators for the Length of a Traveling Salesman Tour".
Computers & Ops.Res. Vol. 19, n° 6, (1.992), 469-78.

33.- CHRISTOFIDES,N.
"Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem".
Management Sciences Researsch Report n° 388, Carnegie-Mellon University, (1.976).

34.- CHRISTOFIDES,N. and EILON,S.
"An Algorithm for the Vehicle-Dispatching Problem".
Operational Researsch Quaterly, vol.20, n° 3, (1.969), 309-318.

35.- CHRISTOFIDES,N., MINGOZZI,A. and TOHT,P.
"State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems".
Networks, vol.11, n° 2, (1.981a), 145-164.

36.- CHRISTOFIDES,N., MINGOZZI,A. and TOTH,P.
"Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations".
Mathematical Programming 20, (1.981b), 255-282.

37.- CHUNG,H.K. and NORBACK,J.P.
"A Clustering and Insertion Heuristic Applied to a Large Routeing Problem in Food Distribution".
J.Opl.Res.Soc., vol.42, n° 7, (1.991), 555-564.

38.- CLARKE,G. and WRIGHT,J.W.
"Scheduling of Vehicles from a Central Depot to a Number of Delevery Points".
Oper.Res., 12, (1.964), 568-581.

39.- COBES,A.M. y COROMINAS,A.
"Affectation des Véhicules aux Routes dans un Système de Transport Multi-Ecole".
Questiio, vol. 14, n°s 1-2-3, (1.990), 181-192.

40.- CRAINIC,T.G. and ROY,J.
"Une Approache de Recouvrement d'Ensembles pour l'Etablissement d'Horaires de Chauffeurs dans le Transport Routier de Charges Partielles".
Recherche Operationnelle/Operations Researsch, vol.24, n° 2, (1.990), 123-158.

41.- CRODWER,H. and PADBERG,M.W.
"Solvimg Large-Scale Symmetric Travelling Salesman Problems to Opmality".
Management Science, vol.26, n° 5, (1.980), 495-509.

42.- CROES,A.
"A Method for Solving Traveling Salesman Problems".
Oper.Res. 5, (1.958), 791-812.

43.- CUADRAS, C.M.
Métodos de Análisis Multivariante
P.P.U., Barcelona 1.991.

44.- CULLEN,F.H., JARVIS.J.J. and RATLIFF,H.D.
"Set Partitioning Based Heuristic for Interactive Routing".
Networks, vol.11, n° 2, (1.981), 125-144.

45.- DAGANZO,C.F.
"Modeling Distribution Problems with Time Windows: Part.I".
Transp.Sci., 21, (1.987a), 171-179.

46.- DAGANZO,C.F.
"Modeling Distribution Problems with Time Windows: Part.II".
Transp.Sci., 21, (1.987b), 180-187.

47.- DESROCHERS,M., DESROSIERS,J. and SOLOMON,M.
"A new Optimization Algorithm for the Vehicle Routing Problem with Time Windows".
Operations Research, vol. 40, n° 2, (1.992), 342-354.

48.- DESROCHERS,M., LENSTRA,J.K. SAVELSBERGH,M.W.P. and SOUMIS,F.
"Vehicle Routing with Time Windows: Optimization and Approximation".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 65-84.

49.-DESROCHERS,M. and SOUMIS,F.
"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, (1.991), 291-310.

50.- DESROSIERS,J., DUMAS,Y. and SOUMIS,F.
"A Dynamic Programming Solution of the large-scale Single-Vehicle Dial-a-Ride Problem with time Windows".
J.Math.Manage.Sci., 6, (1.986a), 301-326.

51.- DESROSIERS,J., PELLETIER,P. and SOUMIS,F.
"Plus Court Chemin avec Constraintes d'Horaires".
RAIRO, Rech.Opér., 17, 4, (1.984), 357-377.

52.- DESROSIERS,J., SAUVE,M. and SOUMIS,F.
"Lagrangen Relaxarion Methods for Solving the Minimun Flet Size Multiple Traveling Salesman Problem with Time Windows".
Manage.Sci., 34, (1.988), 1.005-1.022.

53.- DESROSIERS,J., SOUMIS,F., DESROCHERS,M. and SAUVE,M.
"Méthods for Routing with Time Windows".
Eur.J.Oper.Res., 23, (1.986b), 236-245.

54.- DESROSIERS,J., SOUMIS,F. and DESROCHERS,M.
"Routing with Time Windows by Column Generation".
Networks, vol. 14, (1.984), 545-565.

55.- EATSMAN,W.L.
"Linear programming with Pattern Constraints".
PH.D.Thesis. Harvard University. Cambridge, MA, (1.958).

56.- EGLESE,R.W. and MURDOCK,H.
"Routeing Road Sweepers in a Rural Area".
J.Opl.Res.Soc., vol.24, n° 4, (1.991), 281-288.

57.- EILON,S., WATSON-GANDY,C.D.T. and CHRISTOFIDES,N.
"Distribution Management: Mathematical Modelling and Practical Analysis".
Grifftin, London, (1.971).

58.- FISCHETTI,M. and TOTH,P.
"An Additive Approach for the Optimal Solution of the Prize-Collecting Traveling Salesman Problem".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 319-344.

59.- FISHER,M.L.
"Worst Case Analysis of Algorithms".
Manage.Sci., 26, (1.980), 1-17.

60.- FISHER,M.L., GREENFIELD,A.J., JAIKUMAR,R. and LESTER III,J.T.
"A Computerized Vehicle Routing Application".
Interfaces, vol.12, n° 4, (1.982), 42-52.

61.- FISHER,M.L. and JAIKUMAR,R.
"A Generalized Assignment Heuristic for Vehicle Routing".
Networks, vol.11, n° 2, (1.981), 109-124.

62.- FISHER,M.L., JAIKUMAR,R. and VAN VASSENHOVE,L.N.
"A Multiplier Adjustment Method for the Generalized Assignment Problem".
Management Science, vol. 32, n° 9, (1.986), 1.095-1.103.

63.- FISHER,M.L. and RINOOY KAN,A.H.G.
"The Design, Analysis and Implementation of Heuristics".
Manag.Sci., 34, (1.988), 263-265.

64.- FLOYD,R.
"Algorithm 245: Treesort 3".
Comm. ACM 7, (1.964), 701.

65.- GILLET,B.E. and JOHNSON,J.G.
"Multi-Terminal Vehicle Dispatch Algorithm".
Omega, vol.4, n° 6, (1.976), 711-718.

66.-GILLET,B.E. and MILLER,L.R.
"A Heuristic Algorithm for the Vehicle-Dispatch Problem".
Oper.Res., 22, (1.974), 340-349.

67.- GOLDEN,B.L.
"Route Planning for Coast Guard Ships".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 439-444.

68.- GOLDEN,B.L. and ASSAD,A.A.
"Perspectives on Vehicle Routing: Exciting New Developments".
Operations Research, vol.34, n° 5, (1.986), 803-810.

69.- GOLDEN,B., BODIN,L. DOYLE,T. and STEWART,W.Jr.
"Approximate Traveling Salesman Algorithms".
Operations Researsch, vol.28, n° 3, parte II, (1.980), 694-711.

70.- GOLDEN,B.L. and WASIL,E.A.
"Computerized Vehicle Routing in the Soft Drink Industry".
Operations Research, vol 35, n° 1, (1.987), 6-17.

71.- GROTSCHEL,M.
"On the Symmetric Traveling Salesman Problem: solution of a 120.city Problem".
Mathl.Program.Study 12, (1.980), 61-77.

72.- HAIMOVICH,M. RINNOOY KAN,A.H.G. and STOUGIE,L.
"Analysis of Heuristics for Vehicle Routing Problems".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 47-62.

73.- HAOUARI,M., DEJAX,P. et DESROCHERS,M.
"Les Problems de Tournées avec Contraintes des Fenêtres de Temps: L'Etat de l'Art".
Recherche Operationnelle/Operations Research, vol. 24, n° 3, (1.990), 217-244.

74.- HAYHURST,K.J. and SHIER,D.R.
"A Factoring Approach for the Stochastic Shortest path Problem".
Operations Research Letters 10, (1.991), 329-334.

75.- HELD,M., and KARP,R.M.
"The Traveling Salesman Problem and Minimum Spanning Trees".
Ops.Res. 18, (1.970), 1.138-1.162.

76.- HELD,M., and KARP,R.M.
"The Traveling Salesman Problem and Minimum Spanning Trees: Part II".
Mathematical Programing I, (1.971), 6-25.

77.- HOLT,J.N. and WATTS,A.M.
"Vehicle Routing and Scheduling in the Newspaper Industry".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 347-358.

78.- HOOBAN,J.M.
"Marketing a Vehicle Routing Package".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 447-468.

79.- INFANTE,R.
Métodos de Programación Matemática.
UNED. Madrid, 1.977.

80.- JACOBSEN,S.K. and MADSEN,O.B.G.
"A Comparative Study of Heuristics for Two-Level Routing-Location Problem".
European Journal of Operational Research, 5, (1.980), 378-387.

81.- JAILLET,P. and ODONI,A.R.
"The Probabilistic Vehicle Routing Problem".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 293-318.

82.- JAW,J.J., ODONI,A.R., PSARAFTIS,H.N. and WILSON,N.H.M.
"A Heuristic Algorithm for the Multi-Vehicle Advance Request Dial-a-Ride Problem with Time Windows".
Transp.Res.B, vol.20b, n° 3, (1.986), 243-257.

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

84.- KANELLAKIS,P.C., and PAPADIMITRIU,C.H.
"Local Search for the Asymetric Traveling Salesman Problem".
Ops.Res. 28, (1.980), 1.086-1.099.

85.- KARP,R.M.
"Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane".
Mathematics of Operations Researsch, vol.2, n° 3, (1.977), 209-224.

86.- KIM,C.
"A Minimal Spanning Tree and Aproximate Tours for a Traveling Salesman".
Comp.Sci.Tech.Report. University of Maryland, 1.975.

87.- KOLEN,A.W.J., RINNOOY KAN,A.H.G. and TRIENEKENS,H.W.J.M.
"Vehicle Routing with Time Windows".
Operations Researsch, vol. 35, n° 2, (1.987), 266-273.

88.- LAPORTE,G.
"Location Routing Problems".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 163-198.

89.- LAPORTE,G., and NOBERT,Y.
"Exact Algorithms for the Vehicle Routing problem".
Ann.Discr.Math., 31, (1.987), 147-184.

90.- LARDINOIS,C., CRAINIC,T.G. and DION,J.
"An Interactive Graphic Approach for the Integrated Design of Intercity Transportation Timetables and Vehicle Operations".
Computers & Ops.Res. Vol. 19, n° 2, (1.992), 139-150.

91.- LARSON,R.C. MINKOFF,A. and GREGORY,P.
"Fleet Size and Dispatching for the Marine Division of the New York City Department of Sanitation".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 395-424.

92.- LAWLER,E. LENSTRA,J.K, RINOOY KAN,A.H.G. and SHMOYS,D.
The Traveling Salesman Problem.
John Wiley. New York, 1.985.

93.- LENSTRA,J.K. and RINNOY KAN,A.H.G.
"Complexity of Vehicle Routing and Scheduling Problems".
Networks, vol.11, n° 2, (1.981), 221-228.

94.- LEVI,L. and BODIN,L.
"Scheduling in the Postal Carriers for the United States Postal Service: An Application of Arc Partitioning and Routing".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 359-394.

95.- LIN,S.
"Computer Solutions to the Traveling Salesman Problem".
Bell Syst.Tech.Jou., vol 44, (1.965), 2245-2269.

96.- LIN S. and KERNIGHAN,B.W.
"An Effective Heuristic Algorithm for the Traveling Salesman Problem".
Operations Researsch, vol.20, (1.973), 498-516.

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

98.- LYSGAARD,J.
"Dynamic Transportation Networks in Vehicle Routing and Scheduling".
Interfaces, 22, 3, (1.992), 45-55.

99.- MANGANTI,T.L.
"Combinatorial Optimization and Vehicles Fleet Planning: Perspectives and Prospects".
Networks, vol.11, n° 2, (1.981), 179-214.

100.- MARSTEN,R.E. and SHEPARDSON,F.
"Exact Solution of a crew Scheduling problems Using the Set Partitioning Model: Recent Succesful Aplications".
Networks, vol.11, n° 2, (1.981), 165-178.

101.- MILLER,C. TUCKER,A. and ZEMLIN,R.
"Integer Programming Formulation of Traveling Salesman Problem".
J.Assoc.Comput.Mach., 7, (1.960), 326-332.

102.- MOLE,R. and JAMESON,S.
"A Sequential Route Building Algorithm Employing a Generalized Savings Criteria".
Oper.Res.Quart., (1.976), 503-511.

103.- MONDOU,J.F. CRAINIC,T.G. and NGUYEN,S.
"Shortest Path Algorithms: A Computational Study with C Programming Language".
Computers & Oper.Res., 18, 8, (1.991), 767-786.

104.- NAG,B. GOLDEN,B.L. and ASSAD,A.A.
"Vehicle Routing with Site Dependencies".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 7-46.

105.- NORBACK,J.P. and LOVE,R.F.
"Geometric Approaches to Solving the Traveling Salesman Problem".
Management Science, vol.23, n° 11, (1.977), 1208-1223.

106.- NURMI,K.
"Traveling Salesman Problem Tools for Microcomputers".
Computers & Ops.Res. Vol. 18, n° 8, (1.991), 741-749.

107.- NYGARD,K.E., GREENBERG,P., BOLKAN,W.E. and SWENSON,E.J.
"Generalized Assignment Methods for the Deadline Vehicle Routing Problem".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 7-46.

108.- O'BRIEN,S.K. y NAMEROFF,S.
Turbo-Pascal 7: Manual de Referencia.
Osborne McGraw-Hill. Madrid, 1.993.

109.- OR,I.
"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. (1.976).

110.- ORLOFF,C.S.
"Route Constrained Fleet Scheduling".
Transportaion Sci., 10, (1.976), 149-168.

111.- PALETTA,G.
"A Multiperiod Traveleing Salesman Problem: Heuristic Algorithms".
Computers & Ops.Res. Vol. 19, nº 8, (1.992), 789-795.

112.- PAPE,U.
"Car Transportation by Truck".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 425-438.

113.- PARDO,L., FELIPE,A. y PARDO,J.A.
Programación Lineal Entera.
Díaz de Santos. Madrid, 1990.

114.- PERL,J. and DASKIN,M.S.
"A Warehouse Location-Routing Problem".
Transp.Res.B, vol.19b, nº 5, (1.985), 381-396.

115.- POTVIN,J.Y., LAPALME,G. and ROUSSEAU,J.M.
"Integration of AI and OR Techniques for Computer-Aided Algorithmic Design in the Vehicle Routing Domain".
J.Opl.Res.Soc., vol. 41, nº 6, (1.990), 517-525.

116.- POWELL,W.P.
"A Comparative Review of Alternative Algorithms for the Dynamic Vehicle Allocation Problem".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 249-292.

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

118.- PSARAFTIS,H.
"K-Interchange Procedures for local Search in a Procedence-Constrained Routing Problem".
European J.Oper.Res., 13, (1.983b), 391-402.

119.- PSARAFTIS,H.N.
"Dynamic Vehicle Routing Problems".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 223-248.

120.- PSARAFTIS,H.N., SOLOMON,M.M., MAGNANTI,T.L. and KIM,T.U.
"Routing and Scheduling on a Shoreline with Release Times".
Management Science, vol.36, n° 2, (1.990), 212-223.

121.- RAFT,O.M.
"A Modular Algorithm for an Extebded Vehicle Scheduling Problem".
European Journal of Operational Research 11, (1.982), 67-76.

122.- RHEE,W.T.
"On the fluctuations of the Stochastic Travelling Salesperson Problem".
Matehmatics of Operations Research, vol.16, n° 3, (1.991), 482-489.

123.- RIBEIRO,C., MINOUX,M. and PENNA,M.
"An Optimal Column Generation with Ranking Algorithm for very Large Scale Set Partitioning Problems in Traffic Assignment".
Eur.J.Oper.Res., 41, (1.989), 232-239.

124.- ROSENKRANTZ,D., STERNS,R. and LEWIS,P.
"Approximate Algorithms for the Traveling Salesperson Problem".
15th Annual IEEE Symposium of Switching and Automata Theory, (1.974), 33-42.

125.- ROSS,G.T. and SOLAND,R.M.
"A Branch and Bound Algorithm for the Generalized Assignment Problem".
Mathematical Programming 8 , (1.975), 91-103.

126.- ROTE,G.
"The N-Line Travelling Salesman Problem".
Networks, vol.22, n° 1, (1.992), 91-108.

127.- ROUSSEAU,J.M.
"Customization versus a General Purpose Code for Routing and Scheduling Problems".
In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., (1.988), Nort-Holland, 469-479.

128.- ROY,J. and CRAINIC,T.G.
"Improving Intercity Freight Routing with a Tactical Planning Model".
Interfaces, 22, 3, (1.992), 31-44.

129.- RUBENKING,N.J.
Turbo-Pascal 6.0: Técnicas y Utilidades.
Anaya Multimedia. Madrid, 1.993.

130.- RUSSELL,R.A.
"An Effective Heuristic for the M-Tour Traveling Salesman Problem with Some Side Conditions".
Operations Research, vol. 25, n° 3, (1.977), 517-524.

131.- SAVELSBERGH,M.W.P.
"Local Search for Routing Problems with Time Windows".
Ann.Oper.Res., 4, (1.985), 285-305.

132.- SAVELSBERGH,M.W.P.
"The Generalized Assignment Heuristic Revisited".
Congreso *CORS/TIMS/ORSA*, Vancouver, 1.989.

133.- SAVELSBERGH,M.W.P.
"An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems".
European Journal of Operational Research, 47, (1.990), 75-85.

134.- SCHAGE,L.
"Formulation and structure of More Complex/Realistic Routing and Scheduling Problems".
Networks, vol.11, n° 2, (1.981), 229-232.

135.- SEXTON,T.R. and BODIN,L.D.
"Optimizing Single Vehicle Many-to-Many Operations with Desired Delevery Times: I. Scheduling".
Transportation Sci., 19, (1.985a), 378-410.

136.- SEXTON,T.R. and BODIN,L.D.
"Optimizing Single Vehicle Many-to-Many Operations with Desired Delevery Times: II. Routing".
Transportation Sci., 19, (1.985b), 411-435.

137.- SKITT,R.A. and LEVARY,R.R.
"Vehicle Routing via Column Generation".
European J.Oper.Res., 21, (1.985), 65-76.

138.- SOLOMON,M.M.
"On the Worst-Case Perfomance of Some Heuristics for the Vehicle Routing and Scheduling Problem with Time Windows Constraints".
Networks, 16, (1.986), 161-174.

139.- SOLOMON,M.M.
"Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints".
Operations Research, vol.35, n° 2, (1.987), 254-265.

140.- SOLOMON,M., BAKER,E. and SCHAFFER,J.
"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., (1.988a), Nort-Holland, 85-106.

141.- SOLOMON,M.M. and DESROSIERS,J.
"Time Windows Constrained Routing and Scheduling Problems".
Transp.Scien., 22, (1.988b), 22, 1-13.

142.- SOUMIS,F., SAUVE,M. and LE BEAU,L.
"The Simultaneous Origin-Destination Assignment and Vehicle Routing Problem".
Transportation Science, vol.25, n° 3, (1.991), 188-200.

143.- STERN,H.I. and DROR,M.
"Routing Electric Meter Readers".
Compt.& Ops.Res., vol.6, (1.979), 209-223.

144.- STEWART,W.R. and GOLDEN,B.L.
"A Lagrangian relaxation Heuristic for Vehicle Routing".
Eur.Journ.Oper.Res., 15, (1.984), 84-88.

145.- SWERSEY,A.J. and BALLARD,W.
"Scheduling School Buses".
Mangement Sciences, 30, (1.984), 844-853.

146.- SYSLO,M.M., DEO,N. and KOWALIK,J.S.
Discrete Optimization Algorithms with Pascal Programs.
Prentice-Hall. New Jersey, 1.983.

147.- TALAGRAND,M.
"Complete Convergence of the Directed TSP".
Matehmatics of Operations Research, vol.16, n° 4, (1.991), 881-887.

148.- TANCHOCO,J.M. and SINRIECH,D.
"OSL: Optimal Single-Loop Guide Paths for AGVS".
Int.J.Prod.Res., vol.30,n° 3, (1.992), 665-681.

149.- TSITSIKLIS,J.N.
"Special Cases of Traveling Salesman and Repairman Problems with Time Windows".
Networks, vol.22, n° 3, (1.992), 263-282.

150.- VAN LANDEGHAM
"A Bi-criteria Heuristic for the Vehicle Routing Problem with Time Windows".
Eur.J.Oper.Res., 36, (1.988), 217-266.

151.- VAN VLIET,A., BOENDER,C.G.E. and RINNOOY KAN,A.H.G.
"Interactive Optimization of Bulk Sugar Deliveries".
Interfaces, 22, 3, (1.992), 4-14.

152.- VOLGENANT,T. and JONKER,R.
"A Branch and Bound Algorithm for the Symmetric Traveling Salesman Problem based on the 1-tree Relaxation".
Eur.J.Ops.Res. 9, (1.982), 83-89.

153.- WATERS,C.D.J.
"A Solution Procedure for the Vehicle-Scheduling Problem Based on Iterative Route Improvement".
J.Opl.Res.Soc., vol.38, n° 9, (1.987), 833-839.

154.- WATERS,C.D.J.
"Expanding the Scope of Linear Programming Solutions for Vehicle Scheduling Problems".
Omega, vol.16, n° 6, (1.988), 577-583.

155.- WATERS,C.D.J.
"Vehicle-Scheduling Problems with Uncertainty and Omitted Customers".
J.Opl.Res.Soc., vol. 40, n° 12, (1.989), 1.099-1.108.

156.- WEBB,M.H.J.
"Some Methods of Producing Aproximate Solutions to Traveling Salesman Problem with Hundreds or Thousands of Cities".
Ops.Res. Q., 22, (1.971), 49-66.

157.- WILIAMS,J.
"Algorithm 232: Heapsort".
Comm. ACN 7, (1.964), 347-348.

158.- WIORKOWSKI,J. and McELVAIN.
"A Rapid Heuristic Algorithm for the Approximate Solution of the Traveling Salesman Problem".
Trans. Res. 9, (1.975), 181-185.

159.- WUNDERLICH,J., COLLETTE,M., LEVI,L. and BODIN,L.
"Scheduling Meter Readers for Southern California Gas Company".
Interfaces, 22, 3, (1.992), 22-30.