III Congreso de Economía Regional de Castilla y León **Segovia 1992, Comunicaciones 2:** 702-715

DESARROLLO DE ALGORITMOS PARA EL TSP CON VENTANAS DE TIEMPO

Alberto ARAGON TORRE

Joaquin A. PACHECO BONROSTRO

Dpto.ECONOMIA APLICADA (MATEMATICAS EMPRESARIALES)

E.U.E.EMPRESARIALES DE BURGOS. UNIV. DE VALLADOLID.

1.- Introducción

Uno de los problemas más clásicos de la teoría de redes más presentes en la literatura de la programación matemática en los últimos 40 años es el conocido como problema del Vendedor Ambulante o T.S.P. (Traveling Salesman Problem). Su formulación es la siguiente: Un individuo tiene que visitar n-1 ciudades partiendo de una ciudad inicial (ciudad 1), y volviendo a ella después de visitar las otras n-1 ciudades, de forma que la ruta elegida sea la más corta (o menos costosa).

En este problema y, sobretodo, en el de la generalización que constituye los problemas de Rutas de Vehículos o V.R.P. (Vehicle Routing Problem) es donde más 'exito' han tenido la aplicación 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.

Los problemas de Rutas de Vehículos, dada su parecido planteamiento al del T.S.P. han adoptado las técnicas de solución de este, es más: la absoluta mayoría de las técnicas de solución de los problemas de rutas son básicamente algoritmos del T.S.P. pero con la incorporación de algún paso previo o posterior. Resulta por tanto imprescindible para quien quiera estudiar los problemas de rutas, un conocimiento profundo de los algoritmos del T.S.P.

En cuanto a estos, si nos fijamos en el planteamiento del problema, el problema tiene (n-1)! soluciones factibles, si el nº de ciudades es grande (20 o más) se requererían años emnumerando y comparando todas las soluciones. De hecho todas los algoritmos de solucion exactos de este problema requieren un tiempo de computación exponencial en el nº de ciudades n.

Por tanto, dada la dificultad computacional que requiere este problema tenemos dos tipos de algoritmos posibles:

1.- Algoritmos exactos, de tipo Branch & Bound la mayoría, que reducen mucho la emnumeración de todas las soluciones posibles, asegurándo la obtención de la solución óptima. Sin embargo, el tiempo de computación al ser función exponencial del nº de ciudades, hace 'prohibitivo' su uso para un nº grande de ciudades.

2.- Algoritmos heurísticos o de aproximación, que no aseguran la obtención de la solución óptima, sino de una solución subóptima relativamente proxima a la óptima, en un tiempo de computación menor (crecimiento polinomial en función del nº de ciudades).

En esta segunda dirección estan trabajando la mayoría de las empresas que en la actualidad desarrollan sistemas para la obtención de rutas óptimas para sus clientes. La razón es clara: los usuarios quieren programas que lean los datos y las características de los problemas e inmediatamente obtengan la solución; los usuarios no quieren que ante algún cambio imprevisto en los parámetros (cambio de costes, aparición de un nuevo punto de carga o descarga..) el programa no les haga perder tiempo en la obtención de la solución adecuada.

Otra razón es que, en general, los algoritmos heurísticos son, conceptualmente, más fáciles de asimilar y por tanto es más fácil adaptarlos ante cambios que se produzcan en la estructura del problema a tratar.

En los últimos doce años, los investigadores operativos interesados en los problemas de rutas y programación de vehículos, han hecho énfasis en el desarrollo de algoritmos para problemas de la vida real. Sin embargo, el tamaño de estos problemas ha crecido enormemente, y muchas de las restricciones que surgían en la práctica no han sido consideradas.

Muchos de los algoritmos existentes han sido diseñados para resolver problemas de Rutas *puros* en el sentido en que solo se consideraban aspectos *espaciales*. Ellos no son capaces de captar o recoger todo el conjunto de rstricciones que ocurren en la préctica. Una de estas restricciones es la llegada del vehículo, o vendedor, a la localización de cada cliente en una específica *ventana de tiempo*, es decir, intervalo de tiempo en el cual este debe ser servido. La incorporación de estos problemas da lugar a Problemas Mixtos de Programación y Rutas, y *demanda* algoritmos en los que además se tengan en cuenta aspectos temporales.

El TSP con ventanas de tiempo se puede definir de la forma siguiente: un individuo debe visitar n localizaciones, partiendo de una localización inicial, (localización 1), y volviendo a ella después de recorrer las n-1 restantes, de forma que el tiempo del recorrido sea mínimo; (el coste total se supone linealmente proporcional al tiempo empleado). El tiempo de visita a cada localización i, i=1,..n, no debe superar un tiempo máximo l_i, y debe ser mayor que un tiempo e_i (en caso de

producirse antes se debe esperar). El tiempo de salida es el comienzo de la ventana de tiempo en la localización 1 e_1. La matriz de distancias de tiempo se define como (t_{ij}).

En la línea señalada anteriormente se inserta la comunicación aquí presentada, en la que se pretende la adaptación de algoritmos existentes para el TSP al TSP con ventanas de tiempo, TSPTW. Concretamente, se propondrán adaptaciones a algoritmos de Little, de tipo Branch & Bound, y el algoritmo de Christofides basado en Programación Dinámica.

2.- ADAPTACION DEL ALGORITMO DE LITTLE

2.1.- Descripcion del Algoritmo de Little

En el trabajo titulado *An Algorithm for the Traveling Salesman Problem* Little y otros describieron la siguiente 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 básado en un árbol de búsqueda donde en cada paso todas las posibles soluciones del problema en ese momento, son particionadas en dos o más subconjuntos, cada subconjunto va a representar un vértice en este árbol de decisión. En este caso, en cada paso la partición va a dar lugar a dos subconjuntos: uno con las soluciones conteniendo un arco específico (i,j) y otro no.

Esta *ramificación* (particionamiento o división), la elección del arco (i,j) concretamente, se realiza de acuerdo con algún determinado criterio que tienda a reducir la cantidad de tiempo de búsqueda que conduzca a la solución óptima. Después de ramificar se calculan cotas inferiores del coste total para cada uno de los subconjuntos obtenidos. El siguiente espacio de soluciones que va a ser *explorado* será el correspondiente al menor de las cotas inferiores obtenidas.

Este proceso continua hasta obtener un ciclo hamiltoniano, que me determian una solución del TSP. De esta forma, de los subconjuntos de soluciones que me quedan por explorar en las otras ramas del árbol de búsqueda, sólo necesitan ser exploradas aquellos cuya correspondiente cota inferior sea menor que el coste de la mejor solución que haya obtenido hasta ese momento. De esta forma, con este proceso de *ramificación y acotación,* se consigue ahorrar una gran cantidad de *exploraciones improductivas.*

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, se basa en un proceso de reducción de la matriz de costes (o distancias). Para analizar este proceso de reducción hay que tener en cuente el siguiente teorema:

Sea n el número de nodos a visitar (incluyendo el inicial 1), y sea $C=(c_{ij})$, donde c_{ij} es el coste de viajar entre cada par de ciudades (i j); sea $x=(x_{ij})$ cualquier solución factible al TSP, entonces sea

$$p_i = \min_{j=1,\ldots,n} \{c_{ij}\} \qquad \text{y} \qquad q_j = \min_{i=1,\ldots,n} \{c_{ij} - p_i\}$$

para i,j =1,...,n; se define

$$c'_{ij} = c_{ij} - p_i - q_j, \qquad \text{para i,j=1,..,n;}$$

se tiene que $c'_{ij} \geq 0$, y además

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij} = \sum_{i,j=1}^{n} c'_{ij}x_{ij} + \sum_{i=1}^{n} p_i + \sum_{j=1}^{n} q_j \geq \sum_{i=1}^{n} p_i + \sum_{j=1}^{n} q_j ,$$

Por tanto, para el conjunto de soluciones factibles x del TSP,

minimizar $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij}$, equivale a minimizar $\sum_{i=1}^{n} \sum_{j=1}^{n} c'_{ij}x_{ij}$;

añadiendo al valor obtenido en la segunda expresión la cantidad $\sum_{i=1}^{n} p_i + \sum_{j=1}^{n} q_j$; siendo además esta cantidad cota inferior del problema para la matriz original $C=(c_{ij})$.

En cada vértice del árbol de decisión, el proceso de reducción consiste en restar a cada fila de la matriz C su menor elemento p_i , y posteriormente, a la matriz $(c_{ij}-p_i)$ restarle a cada columna su menor elemento q_j , para obtener 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. 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 se tiene la garantía de tener al menos un 0 por fila y columna. Una vez que se elige el arco (i,j), se particiona o ramifica el conjunto de soluciones que se tienen en ese momento, las que incluyen el arco (i,j), *rama izquierda*, y las que no, *rama derecha*.

En la rama izquierda, se deben tener en cuenta los siguientes puntos:

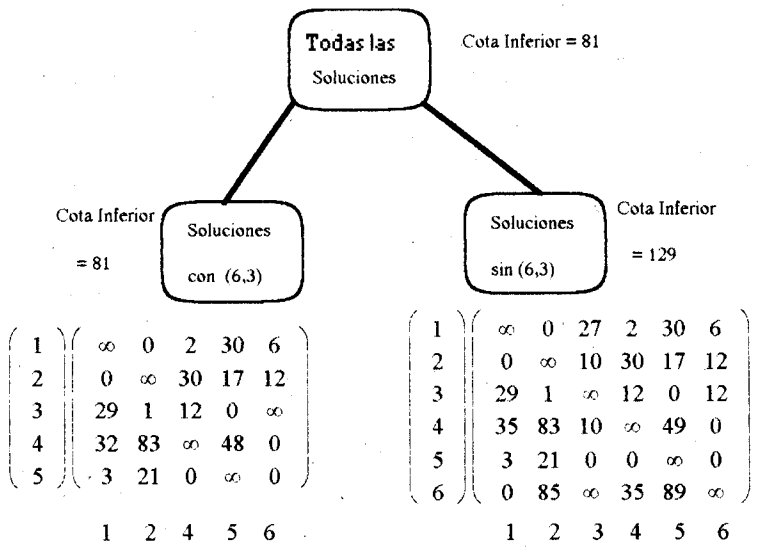
- Si se ha elegido el arco (i,j) para ese conjunto de soluciones, se ha de asegurar que no se incluya en posteriores pasos arcos que *salgan* de i y *lleguen* a j; por tanto se elimina en la matriz C' la fila i y la columna j.
- Al elegirse el arco (i,j), se ha de asegurar que en pasos posteriores no se eliga el arco (j,i); para ello se redefine $c'_{ij} = \infty$.
- Se ha de prevenir, una vez elegido el arco (i,j), la elección de arcos en pasos posteriores que puedan formar subciclos o subrutas con los arcos ya seleccionados en esa dirección del árbol de búsqueda. Para ello si i es el nodo términal de un *camino* de la forma $(u_1, u_2, \ldots, i)$, formado con los arcos seleccionados, y j es el nodo inicial de otro *camino* $(j, v_2, \ldots, v_m)$, con esos mismos arcos, se ha de *impedir* la entrada del arco (v_m, u_1), haciendo ∞ el correspondiente elemento de C'. (Ver figuras).

1	∞	3	93	13	33	9
2	4	∞	77	42	21	16
3	45	17	∞	36	16	28
4	39	90	80	∞	16	28
5	39	90	80	∞	56	7
6	3	88	18	46	92	∞

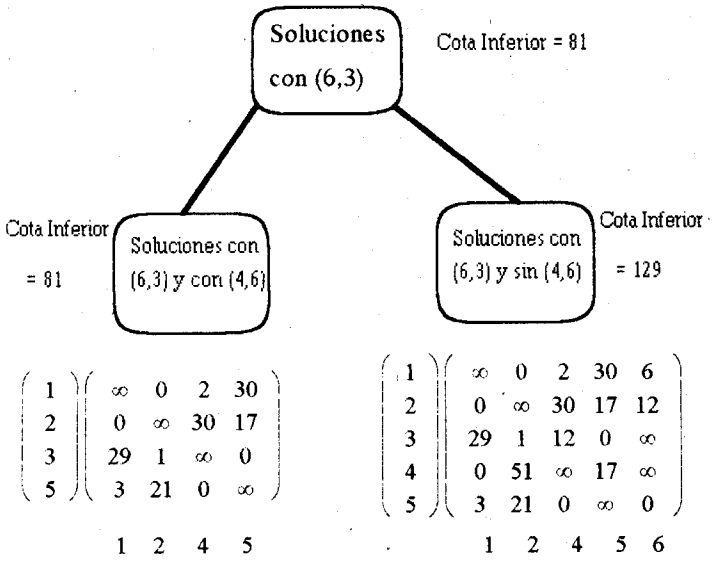
→

1	∞	0	75	2	3	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

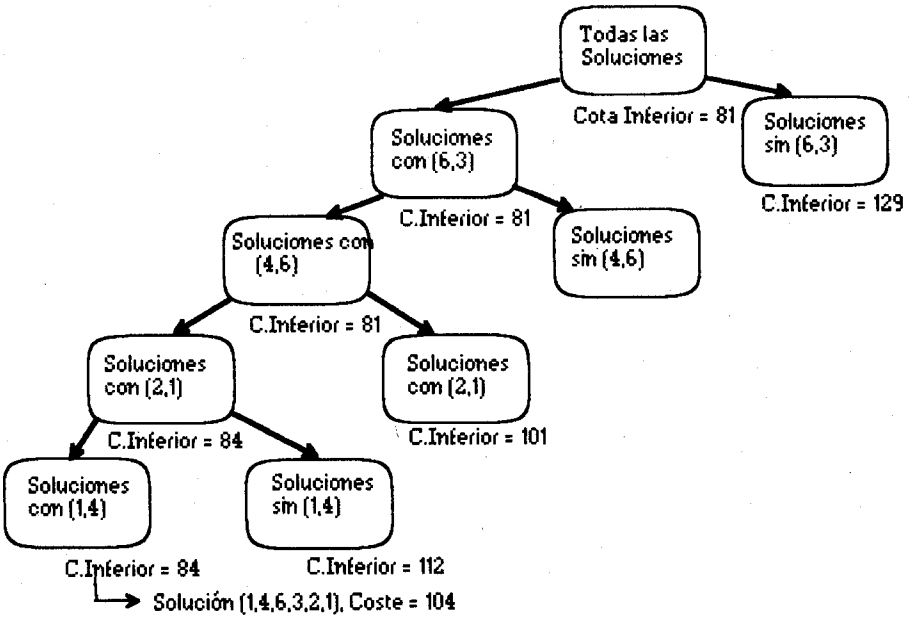
Proceso de Reducción de la Matriz. Cota Inferior =81



Ramificación de la soluciones correspondientes a la selección del arco (6,3)



Continua el proceso de Reducción, Acotamiento y Ramificación



Esquema del Arból de Busqueda Branch & Bound

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.

La cota inferior del vértice asignado a la rama izquierda va a coincidir con el coste acumulado hasta ese momento, es decir, coste acumulado en pasos anteriores más la cantidad $\Sigma_{i=1}^{n} p_i + \Sigma_{j=1}^{n} q_j$, del proceso de reducción. Para la cota inferior del vértice de la rama derecha Syslo, Deo y Kowalik [] proponen elegir el arco correspondiente al 0 de la matriz **C'** que más *penalize* esta rama, es decir, que asegure una mayor cota superior a esta rama. Para ello, para cada (i,j) con $c'_{ij}=0$, se determina la suma del menor elemento de la fila i y de la columna j, exceptuando a c'_{ij} en ambos casos, y se elige el arco (i,j) a ramificar correspondiente a la mayor de estas cantidades, ya

que en la rama derecha al tomar c'_{ij} el valor de ∞, se asegura que en el proceso de reducción, el coste aumente al menos en esa cantidad. Por tanto la cota inferior que pueda asegurar a la rama derecha es la cota inferior de la izquierda más esa cantidad.

Inicialmente se analizará o *explorará* la rama izquierda. El cada paso de este tipo se reducirá en uno la dimensión de la matriz C; se repite el proceso hasta que obtenga una solución. A partir de ese momento se analizar las ramas que no hayan sido analizadas en pasos anteriores siempre que su cota inferior no sea mayor que el coste de la mejor solución obtenida hasta ese momento. (Ver gráficos).

2.2.- Introducción de Adaptaciones a las Ventanas de Tiempo

Los pasos que se introducen al algoritmo descrito anteriormente son los siguientes:

1.- Redefinir la matriz de distancias original de la forma siguiente: Para cada par de puntos de visita (i,j) se chequea su factibilidad respecto las ventanas de tiempo; es decir, se examina si:

$$e_i + t_{ij} \leq l_j;$$

en caso de no ser así en la matriz de distancias de tiempo original se redifine $t_{ij} = \infty$, ya que (i,j) sería un arco infactible.

2.- El proceso de obtención ramificación se realiza igual que en el algoritmo original. Una vez obtenida cada ruta, se comprueba la factibilidad de cada ruta y se halla el tiempo total de su recorrido considerando las ventanas de tiempo. Para ello vale calcular los tiempos de espera que se producen, y sumarlos al tiempo total obtenido inicialmente. Es decir sea la ruta $1 = i_1, i_2,..., i_n, i_{n+1} = 1$, se considera el siguiente proceso recursivo

Paso 1.: Tiempo Total = e_1, k = 1;
Paso 2.: Tiempo Total = Tiempo Total + $t_{i_k i_{k+1}}$,
Paso 3.: si Tiempo Total < e_{k+1} entonces Tiempo Total = e_{k+1},

si Tiempo Total > l_{k+1} entonces Tiempo Total = ∞;

Paso 4.: si k < n hacer k = k+1 e ir al paso 2;

en caso contrario ir al paso 5.

Paso 5.: Fin.

3.- MODIFICACION DEL ALGORITMO DE CHRISTOFIDES

3.1.- Descripcion del Proceso Recursivo de Programación Dinámica de Christofides

Sea X = {1,...,n} y X' = X-{1}, el TSPTW puede ser considerado como el problema de encontrar el camino más corto desde el origen 1, visitando todos los vértices del conjunto X', hasta regresar al origen 1 y respetando las ventanas de tiempo en cada vértice. Christofides, Mingozzi & Toth, [], proponen el siguiente algoritmo de Programación Dinámica. Los estados son de la forma (S,j), con $S \subset X$, y $j \in S$, y d(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. El valor de la solución óptima d(X, 1) viene determinada por la siguientes ecuaciónes de recurrencia:

$d(\{i\},i) = e_i + t_{1i}$, como condiciones iniciales;

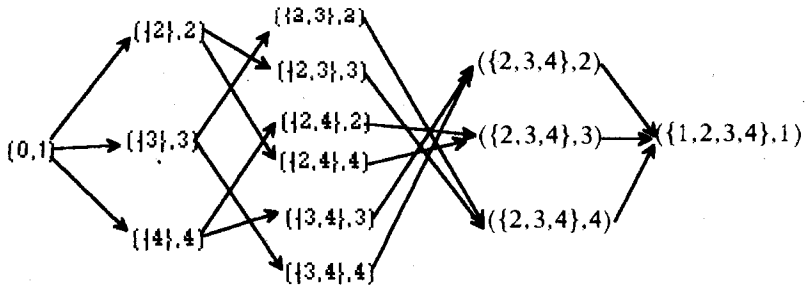
$d(S,j) = \min_{i \in S-\{j\}} \{d(S-\{j\},i)+t_{ij}\}$, para $S \subset X$ y $j \in S$;

se redefine $d(S,j) = e_j$ si $d(S,j) < e_j$, y $d(S,j) = \infty$ si $d(S,j) > l_j$.

3.2.- Mejoras en la Aplicación del Proceso Recursivo

La anterior formulación del problema por programación dinámica es conceptualmente muy sencilla y la programación de este algoritmo recursivo con un compilador de un lenguaje de alto nivel (sobre todo si usa memoria dinámica como TURBO PASCAL o C) no resulta en absoluto dificultosa. Sin embargo el tiempo de cálculo empleado es muy alto, ya que el número de veces que se utiliza dicha fórmula recursiva es de n!, lo que hace que sea poco efectivo excepto para problemas con muy pocas localizaciones.

En estos casos, la mayoría de los autores aconsejan replantear el problema anterior como el de hallar el camino mínimo en la red definida por el conjunto de estados de la forma (S,j), con $S \subset X$ y $j \in S$, desde el origen $(\varnothing, 1)$ hasta el final $(X, 1)$, considerando como arcos y distancias de estos arcos en la red las definidas por la recursión anterior. (Ver dibujo para n = 4).



Para un número n de localizaciones iniciales, el número de vértices de está *red de estados* viene dado por $\sum_{k=1}^{n-1} k \cdot \binom{n-1}{k} + 2 = (n-1) \cdot 2^{n-2} + 2$.

Por otra parte algoritmos para el problema del camino mínimo en una red, como el de Floyd y el de Djikstra utilizan un tiempo de cálculo polinomial en el número de elementos de la red. Concretamente a continuación, se describirá una adaptación del algoritmo de Djikstra o del *etiquetado* para esta red, en el que el número de cálculos requeridos es de $\theta(n^2 \cdot 2^n)$.

Considérese los vértices de la forma (S,j), $\forall S \subset X', j \in S$, además de los vértices $(\varnothing, 1)$ inicial y $(X, 1)$ final. Se define u(S,j) como la distancia mínimma del vértice inicial al (S,j) en cada momento, y $\theta(S,j)$ como el vértice inmediatamente anterior en ese camino. Sea además, T el número de elementos que quedan por seleccionar, el algoritmo para hallar la mínima distancia entre el vértice inicial y final consta de los siguientes pasos:

Paso 1.:

Definir $u(\{j\},j) = e_j$, si $t_{1j} < e_j$

∞, si $t_{1j} > l_j$;
t_{1j}, en caso contrario;
$u(\varnothing,1) = 0$; y
$u(S,j) = \infty$; para el resto de los vértices.

Definir inicialmente T como el conjunto de todos los vértices menos el inicial.
y $\theta(S,j) = (\varnothing, 1)$

Paso 2.:

Buscar $(S,k) \in T/\ u(S,k) = \min_{(S',j)} u(S',j)$;

Poner T = T - {(S,k)};

Si (S,k) = (X,1) entonces parar; si no ir al paso 3.

Paso 3.:

Hacer $u(S \cup \{j\},j) = \min\{u(S \cup \{j\},j),\ u(S,k)+t_{kj}\}$;
y redefinir $u(S \cup \{j\},j) = e_j$ si $d(S \cup \{j\},j) < e_j$, y $d(S \cup \{j\},j) = \infty$ si $d(S,j) > l_j$.
$\forall j \in X' - S$ si X'-S $\neq \varnothing$, y para j =1 si X' = S;

Si $u(S \cup \{j\},j)$ dismunuye entonces hacer $\theta(S \cup \{j\},j) = (S,k)$.

El tiempo del camino mínimo viene dado por u(X,1). Para hallar el camino mínimo en esta red, que en este caso me da el recorrido óptimo para el TSPTW, se utiliza el vector **θ**, ya que $\theta(S,j)$ indica que vértice es el predecesor de (S,j) en este camino mínimo. De esta forma se obtiene la sucesión de vértices

$$(X,1) \to (X',r) \to (X'-\{r\},s) \to \ldots \to (S,k) \to \ldots \to (\{j\},j) \to (\varnothing,1).$$

La sucesión de las segundas componentes de estos vértices, de forma inversa, me define la solución del TSPTW, y el valor de u(S,k) me indica el tiempo en el que cada una de estas localizaciones k son visitadas.

Referencias y Bibliografía

ARAGON,A. y PACHECO,J.
"Mejora de Algoritmos Heurísticos para el TSP".
VI Congreso de ASEPELT-ESPAÑA. GRANADA 1.992

ASSAD,A.A. and GOLDEN,B.L. (eds.)
"Vehicle Routing Problem. Methods & Studies".
Nort-Holland.

CHRISTOFIDES,N., MINGOZZI,A., TOTH,P. and SANDI,C.
"State Space Relaxations Procedures for the Computation of Bounds to Routind Problems".
Networks 11, 145-164.

CHRISTOFIDES,N., MINGOZZI,A., TOTH,P. and SANDI,C. (eds.) (1.979)
"Combinatorial Optimization".
Wiley.

INFANTE,R. (1.977)
"Métodos de Programación Matemática".
UNED.

LAWLER,E.L., LENSTRA,J.K., RINNOY KAN,A.H.G.and SHMOYS,D.B. (Editores) (1.985)
"The Traveling Salesman Problem".
Wiley.

LITTLE,J.D.C., MURTY,K.G., SWEENEY,D.W. and KAREL,C. (1.963)
"An Algorithm for Traveling Salesman Problem".
Oper. Res. 11, 972-989.

MILLER,C.E., TUCKER,A.W. and ZEMLIN,R.A.(1.960)

"Integer Programming Formulation of Travelling Salesman Problems".
JACM 7, 326-329.

PARDO,L., FELIPE,A. Y PARDO,J.A. (1990)
"Programación Lineal Entera".
Díaz de Santos.

SANCHIS,F.J. y MORALES,A. (1.984)
"Programación en lenguaje PASCAL".
Paraninfo.

SYSLO,M.M., DEO,N. and KOWALIK,J.S. (1.983)
"Discrete Optimization Algorithms with Pascal Programs".
Prentice Hall.