GRADO EN MATEMÁTICAS

Departamento de Estadística e Investigación Operativa

TRABAJO DE FIN DE GRADO:

EL PROBLEMA DEL VIAJANTE (TSP)

Miguel Infantes Durán

Dirigido por:
Justo Puerto Albandoz

Dedicado a
mi familia

II

Resumen

El clásico problema del viajante de comercio se puede enunciar como: 'si un viajante parte de una ciudad y las distancias a otras ciudades son conocidas, ¿cuál es la ruta óptima que debe elegir para visitar todas las ciudades y volver a la ciudad de partida?'. El estudio de este problema de programación entera es el objeto de este trabajo.

Comenzaremos con una introducción histórica mostrando las inquietudes que hicieron nacer el problema del viajante a mediados del siglo XX. A continuación, veremos la gran dificultad del problema, hallar un circuito Hamiltoniano. En el último capítulo estudiaremos diferentes métodos de resolución basados en la ramificación y la acotación. Por último, veremos algún ejemplo a modo de aplicación del problema.

Abstract

The classic problem of the traveling salesman can be stated as: 'If a traveler starts in a city and distances to other cities are known, what is the optimal route you should choose to visit all cities and return to the city of departure?'. The study of this whole programming problem is the object of this work.

We will begin with a historical introduction showing the concerns that gave rise to the problem of the traveler in the mid-twentieth century. Next, we will see the great difficulty of the problem, to find a Hamiltonian circuit. In the last chapter we will study different resolution methods based on branch and bound. Finally, we will see some examples as an application of the problem.

Índice general

Índice de figuras

Capítulo 1

Historia

1.1. Introducción

Este trabajo trata sobre el estudio del problema del viajante de comercio, en el que un comerciante quiere visitar n ciudades empezando y terminando en la misma y debe determinar el recorrido a seguir para que la distancia total recorrida sea mínima.

El objetivo es minimizar la distancia total del recorrido. Para estudiar dicha optimización surge la programación lineal. Esta rama de las matemáticas, cuya historia empieza a mediados de siglo XX con George Dantzig, trata el problema de encontrar un mínimo de una función lineal en un poliedro descrito mediante un sistema de ecuaciones lineales e inecuaciones con variables no negativas.

Existen tres aspectos en la historia de cualquier problema matemático: cómo surgió, cómo influyó su investigación en otros avances matemáticos, y cómo fue resuelto el problema finalmente. Si, como en el TSP, el problema es desarrollar un algoritmo que satisfaga estándares de eficiencia, entonces, no ha sido resuelto todavía. De hecho, el TSP es el problema más destacado de los problemas de optimización combinatoria.

1.2. Ciclos y mensajeros

Definición 1. Un grafo es un par $G = (V, A)$ donde V es un conjunto finito de elementos llamados vértices y A es un conjunto de pares de vértices que llamaremos aristas o arcos.

Definición 2. Un ciclo es una sucesión de vértices $u_1, u_2, \ldots, u_p \in V$ tales que $u_1, u_2, \ldots, u_{p-1}$ son distintos, $u_p = u_1$ y $(u_i, u_{i+1}) \in A$. Si el ciclo contiene a todos los vértices de V, entonces se llama ciclo hamiltoniano.

El TSP es el problema de encontrar un ciclo Hamiltoniano con la menor longitud posible. Inversamente, el problema de decidir si un grafo contiene un ciclo Hamiltoniano es un caso especial del TSP.

El Reverendo T. P. Kirkman pudo ser el primero en considerar ciclos Hamiltonianos en un contexto general [39]. Enunció una condición suficiente sobre que un grafo poliédrico admite un ciclo; además mostró que un poliedro con un número impar de vértices, cada uno con un número par de aristas, no puede contener un ciclo Hamiltoniano.

La irrupción de Hamilton en este campo ocurrió prácticamente al mismo tiempo. Inventó un sistema de álgebra no conmutativa, en el cuál las acciones de los elementos de la base pueden ser interpretados en términos de caminos en el dodecaedro regular. Hamilton llamó a este álgebra el *Cálculo Icosiano* (los vértices adyacentes en el dodecaedro corresponden a caras adyacentes en el icosaedro) y utilizó la interpretación gráfica como base para un puzzle, comercializado como 'The Icosian Game'.

Otro precursor del TSP fue Menger (1930). Este problema surge como consecuencia de una nueva definición que éste propuso: la longitud de una curva se define como el límite superior mínimo del conjunto de todos los números que pueden ser obtenidos tomando cada conjunto finito de puntos de la curva y determinando la longitud más corta del grafo poligonal uniendo todos los puntos. Llamamos a ésto el *problema del mensajero*: encontrar el mínimo camino uniendo todos los puntos de un conjunto cuya distancia entre ellos es conocida. Por supuesto, este problema siempre puede ser resuelto en un número finito de intentos. No se conoce ninguna regla que reduzca el número de intentos a menos que el número de permutaciones de los puntos dados. La regla de comenzar por el origen e ir al punto más cercano, después al más cercano, y, así, sucesivamente, no determina el camino más corto, en general.

1.3. El nacimiento del vendedor ambulante

El primer uso del término 'Problema del viajante o del vendedor ambulante' en los círculos matemáticos podría datarse en 1931-32. No sabemos quien introdujo el TSP en las Matemáticas, pero no hay duda de que Merrill Flood es el máximo responsable de difundirlo dentro de la comunidad. Él lo conoció por primera vez por A.W. Tucker en 1937 cuando estaba ocupado con el problema de una ruta de autobús escolar en New Jersey. Flood escribió que Tucker recordaba haberlo escuchado de Hassler Whitney en la Universidad de Princeton, alrededor del año 1931-32. Ello haría que Whitney fuera el pionero del TSP, pero Whitney negó cualquier relación entre él y el TSP.

Una de las razones para la popularización del problema fue su gran conexión con nuevos temas en problemas de combinatoria que estaban naciendo en el apartado de la programación lineal (problema de asignación). El TSP era como esos otros problemas pero, aparentemente, más difícil de resolver. Esto lo convirtió en un reto para otra mucha gente.

1.4. El papel inicial

La aparición de "Solution of a large-scale traveling-salesman problem"[13] fue uno de los principales eventos en la historia de la optimización combinatoria. Para entender el significado necesitamos cierto material previo en programación lineal, especialmente sobre los problemas de asignación y de transporte.

El problema de asignación es elegir n elementos (uno por cada fila y columna) de una matriz n x n, $C = (c_{ij})$ tal que la suma de los elementos elegidos sea la más pequeña posible. Hay $n!$ posibilidades, así que un algoritmo debe hacer algo más que considerar todas las opciones. Un enfoque con la programación lineal es considerar el poliedro P en el espacio n^2 especificado por todas las matrices $X = (x_{ij})$ satisfaciendo las condiciones

$$x_{ij} \geq 0, \qquad \sum_j x_{ij} = 1 \quad (\forall\, i), \qquad \sum_i x_{ij} = 1 \qquad (\forall\, j)$$

y minimizar $\sum_{i,j} c_{ij} x_{ij}$.

La razón es que, según Birkhoff [5], los vértices de P son precisamente todas las matrices X en las que aparece exactamente un 1 en cada fila y

columna y 0 en el resto de entradas. Los $n!$ vértices de P corresponden a las $n!$ posibilidades y $\sum c_{ij}x_{ij}$ indica los costes de elegir una determinada asignación. Por ello, los algoritmos de programación lineal se pueden usar para resolver el problema.

El problema de transporte es más general que el de asignación. Matemáticamente, es el problema de escoger una matriz $m \times n$, $X = (x_{ij})$ tal que

$$x_{ij} \geq 0, \qquad \sum_j x_{ij} = a_i \quad (\forall\, i), \qquad \sum_i x_{ij} = b_j \qquad (\forall\, j)$$

tal que $\sum c_{ij}x_{ij}$ sea mínimo. En este caso, a_i y b_j son enteros dados no negativos con $\sum_i a_i = \sum_j b_j$. El problema de transporte (según Hitchcock [35]) modela esta situación: en cada una de las m fuentes $i\,(i = 1, \ldots, m)$ una cantidad a_i de cierta mercancía está disponible y en cada uno de los n destinos $j\,(j = 1, \ldots, n)$ la cantidad b_j de la mercancía es necesaria. Cuesta exactamente c_{ij} enviar una unidad de la mercancía desde la fuente i al destino j; ¿cómo podemos distribuir lo suficiente para satisfacer todos los requisitos y minimizar el coste total?

Allá por 1953, existían códigos efectivos que implementaban el método simplex en general y adaptado para el caso de problemas de transporte y de asignación. Dantzig, Fulkerson y Johnson dieron un método para resolver un problema de planificación de un petrolero [12]. La solución llegaría unos años más tarde tras quedar el método obsoleto, pero sobrevivió aún así porque el método puede ser usado para estudiar cuestiones básicas sobre la teoría de la combinatoria. Ford y Fulkerson escribieron su primer informe sobre los flujos en las redes [21], iniciando un tema primordial, y Selmer Johnson [36] consiguió un resultado fascinante en la programación de trabajos a través de procesadores en una producción en masa, siendo el primer teorema en combinatoria importante en el estudio de programación de máquinas. Volvamos al problema planteado anteriormente.

Si c_{ij} es la distancia desde la ciudad i hasta la ciudad j, entonces el TSP es similar al problema de asignación. Interpretaremos $x_{ij} = 1$ para referirnos a que el vendedor en su trayecto se mueve de la ciudad i a la j. Pero la solución al problema de asignación puede, bajo esta interpretación, ser un conjunto de subcircuitos. Cada ciudad es visitada exactamente una vez, pero en un conjunto de subcircuitos desconectados; ésto no resuelve el TSP. Así que debemos añadir la condición de que no se creen subcircuitos, que puede ser expresada matemáticamente como 2^{n-1} inecuaciones adicionales a las que

ya existían. Desafortunadamente, el conjunto de vértices del nuevo poliedro Q, a diferencia del conjunto de vértices del anterior poliedro P, puede contener matrices con entradas distintas a 0 o 1; así que debemos imponer la condición adicional de que cada entrada en X debe ser 0 o 1. Ello crea dos dificultades. La primera es que tenemos, en vez de $2n$ ecuaciones en variables no negativas, un gran número de inecuaciones añadidas que considerar. Una segunda dificultad más seria es que el requisito de que las variables tengan que ser 0 o 1 es una restricción de programación lineal entera.

Ahora consideremos el conjunto de todas las matrices X que satisfacen los requisitos del TSP. Supongamos conocido, además de las inecuaciones que impiden la creación de subcircuitos, todas las inecuaciones necesarias para crear un nuevo poliedro R cuyos vértices sean precisamente los circuitos: entonces podemos aplicar la programación lineal. Matemáticamente, R puede ser descrito como encontrar las facetas de la envolvente convexa de los circuitos; pero se ha observado que la descripción de las facetas es complicada (Heller [34]; Kuhn [42]). Sin embargo, tras escuchar una de las conferencias de Kuhn, Dantzig, Fulkerson y Johnson especularon que, comenzando por un circuito óptimo, podría ser posible probar la optimalidad recurriendo a unas cuantas inecuaciones más (llamadas cortes). El método parecía funcionar en problemas pequeños, así que probaron en un problema de 49 ciudades. Tenían que considerar la posibilidad de que fueran necesarios billones de cortes pero para dicho problema solo necesitaron 25 cortes.

Dantzig, Fulkerson y Johnson no solo resolvieron un TSP, también mostraron que el concepto de planos de corte era relevante para la programación entera y que la complejidad de la estructura de un problema de optimización combinatoria en la forma de programación lineal no es un obstáculo insalvable. Además usaron, por primera vez, el concepto de ramificación y acotación.

1.5. Programación entera: cortes, ramificación y acotación

Asumamos que tenemos un poliedro P y queremos minimizar, sobre los puntos enteros de P, una función objetivo lineal. Consideremos la envolvente convexa Q de los puntos enteros de P. Si la supiéramos de antemano, o pudiéramos generar todas las inecuaciones definiendo las caras de Q, entonces el método simplex aplicado a esas caras resolverían el problema de programa-

ción entera. Ahora bien, para un P arbitrario, las caras de Q no son conocidas.

Gomory inventó varios algoritmos [26, 27, 28] que, principalmente, podían encontrar todas las caras de Q, pero tenían como objetivo simplemente encontrar un óptimo para un objetivo determinado en Q. Cada uno de los algoritmos implicaba una generación sistemática de planos de corte que cortaban parte de P pero dejaban Q intacto. No es difícil hallar planos de cortes con esta propiedad, pero es un logro mayor mostrar como crear estos cortes de forma que, tras un número finito de pasos, el vértice óptimo de la intersección de P con los semiespacios generados por los cortes sería el óptimo en Q.

Los algoritmos de Gomory han continuado afectando a la teoría de la programación entera, pero, en la actualidad, no suelen usarse, más que de forma a práctica. El método computacional más popular (especialmente cuando necesitamos que solo algunas de las variables sean enteras) es el de ramificación y acotación (branch and bound). Lo que propone este método es tomar una variable x que queremos que sea entera, pero cuyo valor actual no lo sea, y ramificarla en valores enteros cercanos, considerando los casos: x es al menos el mayor valor entero más alto, o x es el es el valor entero cercano más pequeño. Ello genera la construcción de un árbol de búsqueda, con nodos correspondiendo a problemas de programación lineal con varias restricciones y que no crece en los nodos donde alguna acotación del valor de la solución indica que el árbol no necesita ser explorado bajo esos nodos. El concepto apareció por primera vez en el documento de Dantzig, Fulkerson y Johnson [13] aunque el término *branch and bound* fue usado por primera vez por Little, Murty, Sweeney y Karel [49]. Parece ser que los primeros en mostrar que la filosofía del método de ramificación y acotación era importante en programación entera fueron Land y Doig [43].

Una forma importante de encontrar cotas para el algoritmo de ramificación y acotación es conocida como relajación lagrangiana. Held y Karp [32, 33] fueron los primeros en usar una versión de este procedimiento general para el TSP y, luego, se convirtió en una herramienta para problemas de optimización combinatoria.

1.6. Problemas fáciles y difíciles

A finales de la década de 1960, se apreció que debía haber una diferencia significativa entre los problemas difíciles como el TSP, para los cuales los únicos algoritmos de optimización disponibles eran de una naturaleza enumerativa, y los problemas fáciles como el de asignación o el de transporte, para los cuales existe un algoritmo *bueno* (un término acuñado por Edmonds en 1965 para describir métodos de resolución cuyo tiempo de ejecución aumenta polinomialmente con el tamaño del problema, a lo sumo). Había evidencia de que todos los métodos enumerativos creados para el TSP y otros problemas difíciles se comportaban peor y era lógico conjeturar que los problemas como el TSP eran de tal complejidad que el esfuerzo computacional necesario para cualquier método crecía de forma no polinomial con el tamaño del problema.

De los documentos de Cook, Karp y Levin [10, 37, 46], emergió la idea de que muchos de los problemas sospechosos de ser difíciles eran todos computacionalmente equivalentes, en el sentido de que un algoritmo polinomial para uno de ellos, podría usarse para resolver todos ellos en tiempo polinomial también. La forma de establecer estos resultados es primero mostrar que todos estos problemas permiten formulaciones como problemas de programación entera y luego probando que la programación entera es un caso especial de cierto problema individual difícil (incluyendo el TSP). Estos últimos problemas se llaman *NP-difícil*. Los conceptos de la teoría de complejidad computacional generaron bastante investigación durante la década de 1970.

A pesar de que el carácter *NP-difícil* del TSP no implica que el tiempo de ejecución exponencial para la solución sea inevitable, sirve para reforzar la creencia de que no existe un algoritmo polinomial para el TSP.

Por supuesto, la probable dificultad del TSP general no excluye la posibilidad de que existan procesos de solución polinomial para casos especiales del problema. De hecho, uno de los aspectos más sorprendentes de la teoría de complejidad computacional es que aparece una clara frontera que separa los problemas fáciles de los difíciles. En un lado del barrera, uno encuentra casos especiales de los problemas difíciles que puede probarse que son *NP-difíciles* en sí mismos. Al otro lado, uno encuentra problemas que son similares a uno de los problemas difíciles pero que, sin embargo, resultan ser resolubles en un tiempo polinomial, como el TSP con una matriz de distancias triangular superior (i.e., $c_{ij} = 0$ si $i > j$). El primer TSP cuya estructura permitía un algoritmo polinomial fue un problema de trabajo secuencial de Gilmore y Gomory [25].

1.7. Haciendo frente al carácter NP-difícil

Una de las consecuencias más prácticas del carácter *NP*-difícil de un problema es que limita a 3 las estrategias de resolución. Para empezar, no se acepta la aparente dificultad del problema y se intenta buscar alguna estructura especial que lo haga resoluble óptimamente. No funcionaría, no hay garantía de que se pueda encontrar una solución óptima en una cantidad de tiempo razonable y podría ocurrir una de estas dos cosas: insistimos en la optimalidad de la solución (y arriesgarnos a necesitar mucho tiempo), o insistimos en un método rápido de resolución (y aceptamos la posibilidad de no encontrar la solución óptima).

En la primera opción tenemos los métodos de ramificación y acotación y los planos de corte. También otro método de optimización enumerativo, la técnica recursiva de programación dinámica, ha sido aplicada al TSP (Bellman [3]; Held y Karp [31]). Debido a los grandes requisitos de almacenamiento, la programación dinámica solo puede resolver pequeños problemas. Aún así, el método tiene características útiles para el tiempo de ejecución y es la base para un procedimiento efectivo de acotación en los algoritmos prácticos.

La segunda opción, la referente a la aproximación, es indudablemente la más usada en la práctica. El TSP ha servido de nuevo como herramienta para probar nuevas ideas sobre la aproximación de solución en problemas de optimización combinatoria. Los métodos heurísticos se pueden dividir en función de si construyen una sola solución factible, o de si tratan, sistemáticamente, de mejorar una solución inicial dada. En la primera categoría, hay TSP heurísticos como el de la regla del vecino más próximo, cuya suboptimalidad ya fue estudiada por Menger [50]. En la última categoría, podemos encontrar los procesos de intercambio de aristas de Lin [47] y de Lin y Kernighan [48], que ya fueron mencionados por Flood [20].

El diseño de algoritmos de aproximación llevó inmediatamente al análisis de su comportamiento como una cuestión bastante relevante. Dado un heurístico, ¿qué se puede decir sobre el valor que produce en comparación con el óptimo? Existen tres formas de responder a esta pregunta.

La primera es estrictamente empírica. El heurístico es aplicado a un conjunto de problemas test y los valores de la solución obtenidos son comparados

con los valores óptimos o, si no se conoce éstos últimos, a las cotas inferiores de esos valores. Las dificultades de esta aproximación son la generación de problemas test correctos que sean de alguna forma representativos y la correcta interpretación estadística de los resultados.

A finales de la década de 1960, empezó a ser popular un segundo enfoque; primeramente fue propuesto en un contexto de programación de multiprocesos (Graham [29, 30]). La idea es obtener cotas de la peor desviación posible del óptimo que el heurístico pueda producir e idear instancias del problema malo para el que el heurístico consiga esta desviación. Tal análisis del peor caso provoca una garantía de rendimiento. Para el TSP Euclidiano, la heurística que equivale a duplicar un árbol de expansión mínima y eliminar múltiples visitas a ciudades fue, por un largo periodo de tiempo, el mejor método conocido en el sentido del *peor-caso*; su máximo error era 100 %. En 1976, dos resultados importantes fueron publicados. Christofides [9] presentó un heurístico cuyo máximo error era del 50 %. Sahni y Gonzalez [59] mostraron que, para el TSP general, garantizar un error máximo ajustado es tan difícil como encontrar el óptimo.

Debemos mencionar en este momento que en 1955, un resultado de *peor-caso* de una naturaleza diferente fue obtenido por Few [19]. Para el TSP Euclidiano en el cuadrado unidad con n puntos, Few dio un algoritmo que construye un tour de longitud como mucho $\sqrt{2n} + 1{,}75$.

La dificultad obvia del enfoque *peor-caso* es que todo nuestro juicio de un heurístico está basado en su bajo rendimiento en estancias de problemas patológicos. La evidencia empírica puede sugerir que ese mal rendimiento es extremadamente raro. En tal situación, sería interesante realizar un análisis riguroso del comportamiento de la heurística en un sentido probabilístico. ¿Puede decirse algo sobre el error esperado?

Una vez más, el TSP abrió el camino hacia el área de investigación en análisis probabilístico de algoritmos combinatorios. En 1975, sobre la base de un antiguo resultado de Beardwood, Halton y Hammersley [2], Karp [38] mostró que un heurístico para el TSP Euclidiano, que divide el conjunto de ciudades en subconjuntos y concatena los recorridos óptimos de cada uno de ellos, produce un error del *peor-caso* que, como función de n, crece más lento que el valor de la solución óptima esperada. De esta forma, para un n lo suficientemente grande, el error esperado se hace tan pequeño como se quiera.

La primera dificultad que encontramos en el enfoque probabilístico es que

presupone la especificación de una distribución probabilística sobre el conjunto de todos los ejemplos de TSP. El problema de decidir que distribución es razonable es comparable al problema de encontrar un conjunto apropiado de problemas test.

La segunda dificultad es que casi todos los resultados de esta naturaleza tienden a ser asintóticos. El análisis probabilístico tiene que verse como una explicación para el comportamiento observado de algún principio de diseño que como una garantía de rendimiento de algún tipo.

Los tres enfoques discutidos anteriormente han proporcionado algoritmos de aproximación para el TSP.

1.8. En la actualidad

En la década de 1990, Applegate, Bixby, Chvátal, y Cook desarrollaron el programa Concorde, el cual es usado en muchos de los registros de soluciones recientes. Gerhard Reinelt publicó el TSPLIB en 1991, una colección de instancias de pruebas de dificultad variable, la cual es usada por muchos grupos de investigadores para comparar resultados. En 2006, Cook y otros, obtuvieron un recorrido óptimo para 85.900 ciudades dado por un problema de diseño de microchip. Actualmente TSPLIB es la instancia más larga resuelta. Para otras muchas instancias con millones de ciudades, la solución puede ser encontrada garantizando que la solución contiene un 2-3 % del recorrido óptimo.

En la figura (1.1) podemos observar la evolución de este problema desde 1954 hasta 1987 y en la figura (1.2), una gráfica sobre la evolución del TSP en la historia.

1954	G. Dantzig, R. Fulkerson, S. Johnson	49 ciudades
1971	M. Held, R.M. Karp	57 ciudades
1971	M. Held, R.M. Karp	64 ciudades
1975	P.M. Camerini, L. Fratta, F. Maffioli	67 ciudades
1975	P. Miliotis	80 ciudades
1977	M. Grötschel	120 ciudades
1980	H. Crowder and M. W. Padberg	318 ciudades
1987	M. Padberg and G. Rinaldi	532 ciudades
1987	M. Grötschel and O. Holland	666 ciudades
1987	M. Padberg and G. Rinaldi	1002 ciudades
1987	M. Padberg and G. Rinaldi	2392 ciudades

Figura 1.1: Evolución del TSP a lo largo de la historia.

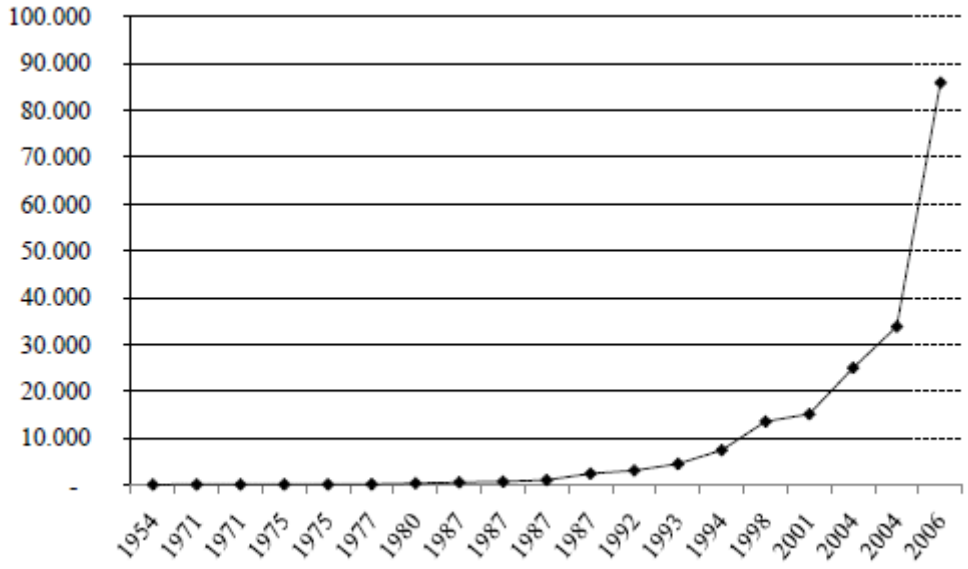

Figura 1.2: Evolución del TSP a lo largo de la historia.

Capítulo 2

El problema del circuito Hamiltoniano

2.1. Introducción

En ciertas industrias surge el siguiente problema: un número n de objetos tienen que ser manufacturados usando una sola instalación de procesamiento. La instalación, puede o no tener que ser reiniciada, después de que el objeto p_i haya sido fabricado, dependiendo de la combinación de objetos (p_i, p_j). El coste de reiniciar la instalación es constante independientemente de p_i, que acaba de ser fabricado o de p_j que es el siguiente. Supongamos que estos n objetos tienen que ser fabricados de forma continua y cíclica, de forma que una vez acabado el último, empieza de nuevo el primero.

Entonces surge el problema de si se puede encontrar una secuencia cíclica de producción que no requiera reiniciar la instalación. La respuesta a esta pregunta depende de si un grafo G cuyos vértices representan los objetos y donde la existencia de un arco (x_i, x_j) implica que p_j puede proseguir la producción del objeto p_i en la instalación sin reinicio, posee un circuito Hamiltoniano o no. (Un circuito Hamiltoniano es un circuito que pasa una y solo una vez por cada vértice del grafo.)

Si no puede hallarse una secuencia cíclica de producción que no requiera reinicio, ¿cuál es la secuencia de fabricación que incluye el menor coste adicional de reinicio? La respuesta a esta pregunta puede darse tras una aplicación iterativa de un algoritmo que determina un circuito Hamiltoniano en un grafo.

Por lo tanto, un método para saber si un grafo contiene un circuito Hamiltoniano o no, tiene aplicaciones directas en problemas de procesamiento de operaciones.

En este capítulo trataremos dos problemas:
Problema (i). Dado un grafo G, encontrar un circuito Hamiltoniano de G, (o alternativamente todos los circuitos), si existe alguno.
Problema (ii). Dado un grafo *completo* G cuyos arcos tienen costes arbitrarios $\mathbf{C} = [c_{ij}]$ asociados, hallar el circuito Hamiltoniano con el menor coste total. Si G no es completo, puede considerarse que el coste de los arcos que faltan es infinito.

Solucionar el TSP y sus variantes tiene un gran número de aplicaciones prácticas en diversos campos. Por ejemplo, consideremos el problema en el que un vehículo deja un almacén para repartir mercancías a una serie de clientes y luego vuelve al almacén. El coste del viaje es proporcional a la distancia total recorrida por el vehículo tal que, dada la matriz de las distancias entre clientes, el viaje más barato es la solución para el correspondiente problema del viajante. Problemas similares ocurren en el reparto de correo, transporte escolar según el número de paradas, etc.

Es obvio que el problema (i) anterior es un caso especial del problema (ii). En consecuencia, para cualquier grafo G dado, se le puede asignar costes aleatorios a los arcos de G para formar un problema del viajante. Si la solución a este problema -i.e. el circuito Hamiltoniano más corto- tiene un valor finito, entonces es, en efecto, un circuito Hamiltoniano de G (i.e. la respuesta al problema (i)). Si, al contrario, la solución es un valor infinito, entonces G no contiene un circuito Hamiltoniano. De todas formas, podemos considerar un grafo completo G_1 con costes generales $[c_{ij}]$ y consideremos el problema de encontrar ese circuito Hamiltoniano de G_1 cuyo arco más largo sea mínimo.

En el grafo G_1 anteriormente mostrado, podemos encontrar un circuito Hamiltoniano. Llamemos a este circuito ϕ_1 y sea $\hat{c}_1$ el coste del arco más largo en ϕ_1. Eliminemos de G_1 cualquier arco cuyo coste sea mayor o igual que $\hat{c}_1$ para obtener el grafo G_2. Hallemos el circuito Hamiltoniano ϕ_2 en G_2 y sea $\hat{c}_2$ el coste del arco más largo en ϕ_2. Una vez más, eliminemos de G_2 cualquier arco con el coste mayor o igual que $\hat{c}_2$ para formar G_3 y continuemos con el mismo proceso hasta el grafo G_{m+1} que no contenga circuitos Hamiltonianos. El circuito Hamiltoniano ϕ_m en G_m es entonces, por definición, la solución al TSP, ya que la falta de un circuito Hamiltoniano en G_{m+1} implica que no existe circuito Hamiltoniano en G_1 que no use al menos un arco con un

coste mayor o igual que $\hat{c}_m$. En consecuencia, un algoritmo que encuentre un circuito Hamiltoniano en un grafo también resuelve el TSP. Si la solución del TSP para G_1 tiene un coste finito, entonces se ha encontrado el circuito Hamiltoniano correspondiente en G; si la solución tiene un coste infinito entonces no existe ningún circuito Hamiltoniano en G.

2.2. Circuitos Hamiltonianos en un Grafo

Dado un grafo G, no existe un criterio sencillo o un método algebraico para decidir si G contiene un circuito Hamiltoniano o no. Existen criterios como los dados por Pósa [55], Nash-Williams [53] y Ore [54] que son interesantes desde el punto de vista teórico pero son demasiado débiles para ser útiles sobre grafos arbitrarios en la práctica. Los métodos algebraicos que han ido apareciendo para determinar circuitos Hamiltonianos no son capaces de trabajar con problemas de más de unas decenas de vértices ya que requieren una gran cantidad de memoria computacional y de tiempo. Más acertada es la técnica enumerativa de Roberts y Flores [58, 57] que no requiere un gran almacenamiento computacional pero cuyo tiempo de ejecución sigue creciendo de manera exponencial con el número de vértices en el grafo.

2.2.1. Un método algebraico

Este método está basado en el trabajo de Yau [65], Danielson [11] y Dhawan [15] e implica la generación de todos los caminos elementales mediante multiplicaciones de matrices sucesivamente.

El "producto de vértices internos"de un camino $x_1, x_2, ..., x_{k-1}, x_k$ está definido como la secuencia de vértices $x_2, x_3, ..., x_{k-1}$ excluyendo los vértices x_1 y x_k. La "matriz de adyacencia con las variables modificadas"$\mathbf{B} = [\beta(i,j)]$ es una matriz n x n donde $\beta(i,j) = x_j$ si hay un arco desde x_i hasta x_j y cero en caso contrario. Ahora, supongamos que tenemos una matriz $\mathbf{P}_l = [p_l(i,j)]$, donde $p_l(i,j)$ es la suma de los productos de vértices internos de todos los caminos elementales con cardinal l entre los vértices x_i y x_j, para $x_i \neq x_j$. Asumimos $p_l(i,i) = 0$ para todo i. El producto de matrices algebraicas $\mathbf{B} \cdot \mathbf{P}_l \equiv \mathbf{P}'_{l+1} = [p'_{l+1}(s,t)]$ viene ahora dado por:

$$p'_{l+1}(s,t) = \sum_k \beta(s,k) \cdot p_l(k,t) \tag{2.1}$$

i.e. $p'_{l+1}(s,t)$ es la suma de los productos escalares de todos los caminos desde x_s hasta x_t con cardinal $l+1$. Como todos los caminos de x_k a x_t representados por los productos escalares de $p_l(k,t)$ son elementales, los únicos caminos no elementales que pueden resultar de la expresión [2.1] son aquellos cuyo producto escalar en $p_l(k,t)$ contiene el vértice x_s. De esta forma, si todos los términos que contienen a x_s son eliminados de p'_{l+1}, saldrá como resultado $p_{l+1}(s,t)$. La matriz $\mathbf{P}_{l+1} = [p_{l+1}(s,t)]$ con todos los términos diagonales hechos 0, es entonces la matriz de todos los caminos elementales de cardinal $l+1$.

Continuando de esta forma, $\mathbf{B} \cdot \mathbf{P}_{l+1}$ producirá $\mathbf{P}_{l+2}$, etcétera hasta que la matriz de camino $\mathbf{P}_{n-1}$ sea generada dando todos los caminos Hamiltonianos (de cardinal $n-1$), entre todos los pares de vértices. Los circuitos Hamiltonianos son, entonces, dados por los caminos de $\mathbf{P}_{n-1}$ y esos arcos de G que unen el último con el primer vértice de cada camino. El valor inicial de la matriz $\mathbf{P}$ (i.e. $\mathbf{P}_1$), puede ser tomado como la matriz de adyacencia $\mathbf{A}$ del grafo con todos los elementos diagonales siendo cero.

Las desventajas de este método son obvias. Al tiempo que el proceso de multiplicación de matrices continúa (i.e. al tiempo que l aumenta), cada elemento de la matriz $\mathbf{P}_l$ está compuesto de más y más términos hasta cierto valor crítico de l a partir del cual el número de términos empieza a decrecer otra vez. Esto se debe a que para pequeños valores de l y para grafos en la práctica, el número de caminos de cardinal $l+1$ suele ser mayor que el número de caminos de cardinal l, mientras que para grandes valores de l suele pasar lo contrario. Más aún, como la longitud de cada término producto escalar de un vértice consigo mismo es aumentado en uno cada vez que l crece en uno, los requisitos de almacenamiento de la matriz $\mathbf{P}_l$ crecen muy rápidamente hasta un máximo valor crítico de l donde los requisitos empiezan a disminuir otra vez.

Una leve modificación en el método anterior reduce tanto los requisitos de almacenamiento como de tiempo. Como solo estamos interesados en circuitos Hamiltonianos y, como mencionamos anteriormente, esto puede ser obtenido por los términos del producto escalar en las celdas diagonales de la matriz $\mathbf{B} \cdot \mathbf{P}_{n-1}$, entonces solo la celda $p_{n-1}(1,1)$ es necesaria. Ésta puede ser obtenida calculando y manteniendo en cada estado no toda la matriz $\mathbf{P}_l$ sino solo la primera columna de $\mathbf{P}_l$. La primera columna de $\mathbf{P}_{l+1}$ viene dada multiplicando $\mathbf{B}$ con la primera columna de $\mathbf{P}_l$. Esta modificación reduce tanto los requisitos de tiempo como de almacenamiento del método en un factor n.

2.2.2. El método de enumeración de Roberts y Flores

Contrariamente a los métodos algebraicos que intentan hallar todos los circuitos Hamiltonianos a la vez y, por tanto, almacenar todos los caminos que podrían ser parte de esos circuitos, el método de enumeración considera un camino en un tiempo que va extendiéndose hasta que un circuito Hamiltoniano es encontrado o se observa que el camino no va a llevar a ningún circuito. El camino es entonces modificado para asegurarse de que la búsqueda continua. De esta forma, se requiere poco almacenamiento para la búsqueda y los circuitos Hamiltonianos son encontrados en menos tiempo.

El siguiente esquema enumerativo que usa la técnica de búsqueda hacia atrás fue sugerida por primera vez por Roberts and Flores [58, 57]. El método comienza formando una matriz $k \times n$, $\mathbf{M} = [m_{ij}]$ donde el elemento m_{ij} es el i-ésimo vértice (digamos x_q) tal que existe un arco (x_j, x_q) en el grafo $G = (X, \Gamma)$. Los vértices x_q en el conjunto $\Gamma(x_j)$ se pueden organizar arbitrariamente para formar las entradas de la j-ésima columna de la matriz $\mathbf{M}$. El número de filas k de la matriz $\mathbf{M}$ es el máximo grado de salida de un vértice.

El método procede de la siguiente forma. Un vértice inicial (digamos x_1) es elegido y forma la primera entrada del conjunto S que almacenará el camino buscado en cualquier momento. El primer vértice (digamos a) en la columna x_1 es añadido a S. Después el primer vértice factible (digamos b) en la columna a es añadido a S, después el primer vértice factible (digamos c) en la columna b es añadido a S, etcétera. Por factible entendemos un vértice que no esté ya en S. Dos posibilidades evitarán que entre un vértice en $S = x_1, a, b, c, ..., x_{r-1}, x_r$ en algún estado r:
(1) No hay ningún vértice factible en x_r
(2) El camino representado por la secuencia de vértices en S tiene cardinal n - 1, i.e. forma un camino Hamiltoniano.
En el caso (2):
(i) arco (x_r, x_1) existe en G y ya se ha encontrado un circuito ahí.
(ii) arco (x_r, x_1) no existe y no se puede obtener un circuito Hamiltoniano.

En los casos (1) y (2.(ii)) debemos usar el método "vuelta atrás", mientras que en el caso (2.(i)) la búsqueda puede ser interrumpida mostrando el resultado (si solo se pide un circuito Hamiltoniano), o (si todos los circuitos son requeridos), los elementos de salida tienen que ser seguidos del método de vuelta atrás.

El método de vuelta atrás implica eliminar el último vértice introducido x_r de S para crear el conjunto $S = x_1, a, b, c, ..., x_{r-1}$ y añadir a S el primer vértice factible siguiente a x_r en la columna x_{r-1} de la matriz **M**. Si no existe un vértice factible se da un paso más en el método de vuelta atrás y así sucesivamente.

El fin de la búsqueda ocurre cuando el conjunto S solo contiene el vértice x_1 y no hay ningún vértice factible que añadir a S. Los circuitos Hamiltonianos encontrados son todos los que existen en el grafo.

2.2.2.1. Mejoras respecto al método básico

En cierto momento durante la búsqueda, el camino viene dado por el conjunto $S = \{x_1, x_2, ..., x_r\}$, y el siguiente vértice en considerarse para incluir en la lista es $x* \notin S$. Consideremos las siguientes situaciones en las que un vértice está aislado en el subgrafo que queda tras eliminar de $G = (X, \Gamma)$ todos los vértices que forman el camino.

(a) Si existe un vértice $x \in X - S$ tal que $x \in \Gamma(x_r)$ y $\Gamma^{-1}(x) \in S$, entonces si cualquier otro vértice x* distinto de x es añadido a S, dicho vértice x no podrá ser rechazado posteriormente por ningún vértice final futuro del camino formado. Dicho camino no puede desembocar en un circuito Hamiltoniano. De esta forma, bajo estas condiciones, x es el único vértice que puede ser añadido a S para extender el camino.
(b) Si existe un vértice $x \in X - S$ tal que $x \notin \Gamma^{-1}(x_1)$ y $\Gamma(x) \subset S \cup x*$ para algún otro vértice x*, entonces x* no puede añadirse a S ya que no existe un camino entre x y x_1 en el subgrafo que queda. El camino compuesto de $S \cup x*$ no puede desembocar en un circuito Hamiltoniano y otro vértice, distinto de x*, debe considerarse para añadirlo al conjunto S.

Las pruebas para las condiciones (a) y (b) ralentizarán el proceso iterativo y para grafos pequeños, no hay ninguna mejora sobre el método original. Para grafos con más vértices, las pruebas provocan una mejoría en los tiempos de computación.

2.3. El método de los múltiples caminos

Examinando el algoritmo enumerativo de Roberts y Flores, observamos que, incluso con las mejoras introducidas anteriormente, hay que tener en cuenta las consecuencias de ampliar el camino que se está formando sobre el resto del grafo. En general, la formación de un camino S_0 mediante la búsqueda (S_0 se considera como una secuencia ordenada de vértices y como un conjunto ordinario) implica otros caminos $S_1, S_2, \cdots$ en otras partes del grafo. Estos caminos ayudan a completar un circuito Hamiltoniano más rápidamente o a concluir que no existe un circuito Hamiltoniano que contenga el camino S_0.

El método descrito en esta sección fue desarrollado originalmente por Selby [60] para grafos no dirigidos. Aquí se mostrará una versión modificada para grafos dirigidos. El método procede de la siguiente forma.

Supongamos que en cierto momento de la búsqueda se ha formado un camino S_0 con sus consecuentes $S_1, S_2, \cdots$. Consideremos cualquier vértice intermedio en cualquiera de estos caminos (por vértice intermedio entendemos cualquier vértice que no sea el primero o el último), es obvio que como dicho vértice ya está unido en un camino por dos arcos, todo otro arco que vaya desde o hacia dicho vértice puede ser eliminado y para el vértice final de cualquiera de los caminos, también se puede eliminar todos los otros arcos que acaben en él. Además, excepto para el caso en el que solo existe un camino (digamos S_0) y que pasa por todos los vértices de G (i.e. cuando S_0 es un camino Hamiltoniano), cualquier arco que empiece en el final de cualquier camino hasta el vértice inicial del mismo camino puede ser eliminado ya que dichos arcos cerrarán ciclos no Hamiltonianos.

La eliminación de todos estos arcos dejará el grafo G con muchos vértices (todos los vértices intermedios de los caminos) teniendo solo un arco acabando en ellos y un arco que emane de ellos también. Todos estos vértices intermedios y sus arcos incidentes son eliminados de G y, en su lugar, introducimos un solo arco por cada camino desde su vértice inicial hasta su último, el resultado es llamado grafo reducido $G_k = (X_k, \Gamma_k)$; donde k es un índice que indica el estado en el que se encuentra la búsqueda.

Consideremos ahora la extensión del camino S_0 formado por la búsqueda, por otro vértice x_j que es un vértice factible de añadir a S_0 en el sentido mencionado en el algoritmo de Roberts y Flores, i.e. existe un arco en G_k desde el vértice final de S_0 (llamado $e(S_0)$) hasta el vértice x_j. Las consecuencias

de añadir x_j a S_0 son las siguientes.

(1) Primero eliminar todos los arcos que sean innecesarios de G_k, i.e.

   (I) Todos los arcos que acaben en x_j o que emanen de $e(S_0)$, excepto el arco $(e(S_0), x_j)$.

   (II) Cualquier arco desde x_j al vértice inicial de S_0.

   (III) Si x_j fuera el vértice inicial de otro camino S_j, eliminar también cualquier arco desde el vértice final de S_j hasta el inicial de S_0.

(2) Llamemos al grafo que existe tras la eliminación de arcos $G_k^{'} = (X_k, \Gamma_{k'})$.

   - Si existe un vértice x de $G_k^{'}$ que no sea el final de ninguno de los caminos $S_0, S_1, \cdots$ etc. y que, como resultado de la eliminación de arcos, ahora tenga un grado de entrada unidad, i.e. $\mid \Gamma_{k'}^{-1}(x) \mid = 1$, entonces borrar todos los arcos que emanen del vértice $v = \Gamma_{k'}^{-1}(x)$ excepto del arco (v, x).
   - Si existe un vértice x de $G_k^{'}$ que no sea el inicial de cualquier camino y que, como resultado de la eliminación de arcos, ahora tiene un grado de salida unidad, i.e. $\mid \Gamma_{k'}(x) \mid = 1$, entonces borrar todos los arcos que emanen de x excepto el arco $(x, \Gamma_{k'}(x))$.
   - Actualizar todos los caminos y eliminar cualquier arco que vayan desde los vértices iniciales hasta los finales.

   Repetir el paso (2) hasta que no puedan eliminarse más arcos.

(3) Eliminar del grafo final $G_k^{'}$ cualquier vértice que tenga grado de entrada y salida unidad, i.e. vértices que se han convertido ahora en vértices intermedios de caminos. Esta eliminación es llevada a cabo de la forma mencionada anteriormente y el resultado es un nuevo grafo reducido G_{k+1} que reemplaza el grafo anterior G_k.

Si ahora haber añadido el vértice x_j al camino S_0 provoca que el grado de entrada o de salida (o ambos) de algún vértice x al final del paso (2) se haga cero, entonces no existe un circuito Hamiltoniano. El vértice x_j es rechazado y se elige otro vértice x_j del conjunto de vértices $\Gamma_{k'}[e(S_0)]$ como posible extensión del camino S_0, hasta que dicho conjunto se agote y entonces se produce el proceso vuelta atrás (i.e. $e(S_0)$ se elimina de S_0 y reemplazado por otro vértice, etc.). Nótese que el proceso de vuelta atrás implica que hay que almacenar la suficiente información sobre los arcos eliminados en los pasos (1) y (2) en cada estado k, para luego ser capaces de reconstruir G_k desde

G_{k+1} para cualquier k, cuando el proceso de vuelta atrás ocurra.

Si (en cierto momento) al final del paso (2), se encuentra que solo queda un camino que pase por todos los vértices, entonces la existencia de un circuito Hamiltoniano puede comprobarse inmediatamente. Si no se encuentra un circuito (o si se encuentra uno, pero queremos todos los posibles), puede ocurrir el proceso de vuelta atrás.

Una forma computacional mejorada consiste en comprobar en cada iteración del paso (2) (en lugar de al final) que los grados de entrada y salida de todos los vértices en $G_k^{'}$ son distintos de cero. Entonces, en el momento en el que uno de ellos sea cero se puede producir el proceso de vuelta atrás y cuando encontremos un camino Hamiltoniano, también encontramos un circuito Hamiltoniano sin necesidad de comprobar el arco de vuelta.

Si ninguno de los casos anteriores ocurre, i.e. si (en el momento k), al final del paso (2), hay más de un camino restante y todos los grados de entrada y salida son distintos de cero, no se puede sacar ninguna conclusión. x_j es añadido a S_0 y se elige otro vértice para extender el nuevo camino S_0 aún más. Los pasos (1), (2) y (3) son repetidos empezando en el nuevo grafo reducido.

2.4. Comparación de los métodos

La versión original del algoritmo de Roberts y Flores, su versión mejorada y el método de los múltiples caminos son comparados en esta sección. Los tres métodos se comparan en base al tiempo de computación requerido para encontrar un circuito Hamiltoniano si solo existe uno o indicar que tal circuito no existe. Las pruebas se llevaron a cabo en grafos no dirigidos aleatorios con grados de vértices en un rango predefinido. Un total de 200 grafos fueron probados y los resultados dados son medias. Todos los grafos contenían circuitos Hamiltonianos [8].

La figura (2.1) muestra los tiempos de computación requeridos en el algoritmo original de Roberts y Flores con respecto al número de vértices en el grafo y cuando el rango de grados de los vértices es 3-5. Debido a las grandes variaciones de tiempo que ocurren para grafos del mismo tamaño, se muestran tres curvas en la figura dando la media, el máximo y el mínimo tiempo de computación en diferentes grafos con el mismo número de vértices. Di-

cha figura está hecha en una representación semilogarítmica que indica que los requisitos de tiempo de computación crecen de forma exponencial con el número de vértices. Una fórmula que muestre el tiempo aproximado de computación T como función del número de vértices n en el grafo con grados de vértices en el rango 3-5, viene dada por:

$$T = 0{,}85 \times 10^{-4} \times 10^{0{,}155n}(seconds - -CDC6600)$$

La versión mejorada del algoritmo de Roberts y Flores no es mucho mejor que la original y el algoritmo sigue requiriendo un tiempo de computación creciente más o menos exponencial con respecto a n. La razón de los tiempos de computación de las dos versiones para grafos no dirigidos con grados de vértices en el rango 3-5 se muestra en la figura (2.2). En esta figura se observa que la versión 'mejorada' es incluso peor, mientras que para grafos con más de 20 vértices, hay más de un 50 % de mejora en tiempo de computación usando el método mejorado.

Con el mismo conjunto de grafos mencionado anteriormente, el algoritmo de múltiples caminos demuestra ser muy efectivo. Esto se observa en la figura (2.3) mostrando los tiempos de computación requeridos en este algoritmo. Se observa en esta figura (representada en una escala lineal), que el tiempo crece lentamente con el número de vértices en los grafos (lo cual no quiere decir que a partir de cierto número de vértices crezca de forma no lineal).

Una ventaja más de este método es que existe una variación pequeña en los tiempos obtenidos para resolver diferentes grafos del mismo tamaño, siendo de esta forma sencillo estimar con una confianza razonable el tiempo de computación para diversos problemas. Es más, los experimentos realizados con los grafos cuyos vértices tienen grado diferentes en el rango 3-5, han mostrado que este método queda invariante según el grado de sus vértices.

Los resultados mostrados en las tres figuras se refieren siempre a encontrar un circuito Hamiltoniano en un grafo. Sería interesante mencionar algunas experiencias computacionales con los tres tipos de algoritmos cuando se quiere encontrar todos los circuitos Hamiltonianos. De esta forma, para un grafo no dirigido de 20 vértices con grados de vértices en el rango 3-5, el algoritmo original de Roberts y Flores toma 2 segundos en encontrar todos los circuitos Hamiltonianos (18 en total); la versión mejorada de dicho algoritmo tardó 1,2 segundos y el algoritmo de múltiples caminos tardó 0,07 segundos, todos los tiempos según un ordenador CDC 6600 [8].

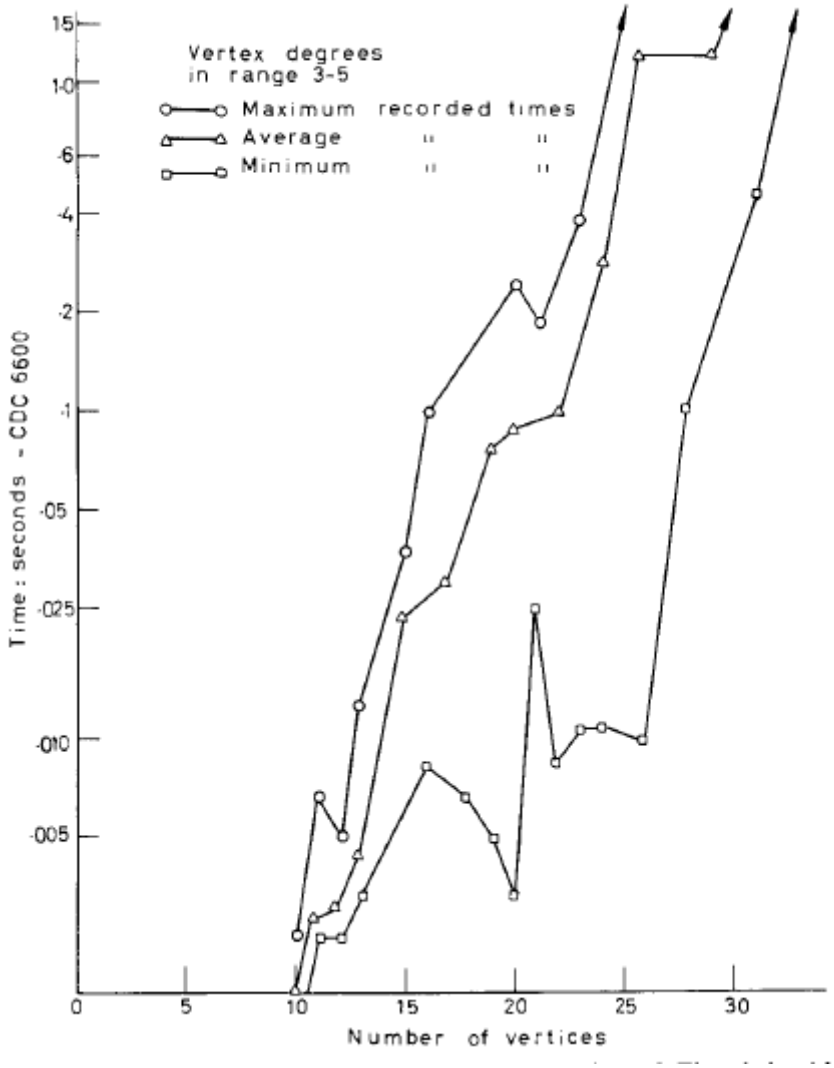

Figura 2.1: Comportamiento computacional del algoritmo de Roberts y Flores

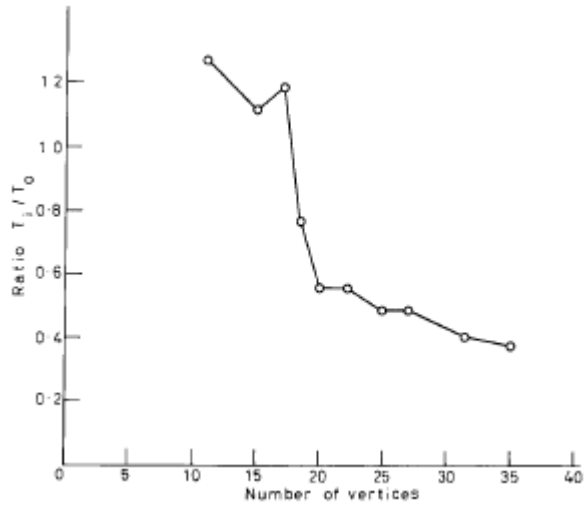

Figura 2.2: Comportamiento computacional del algoritmo de Roberts y Flores mejorado

T_0: tiempo de computación del método original
T_1: tiempo de computación del método mejorado

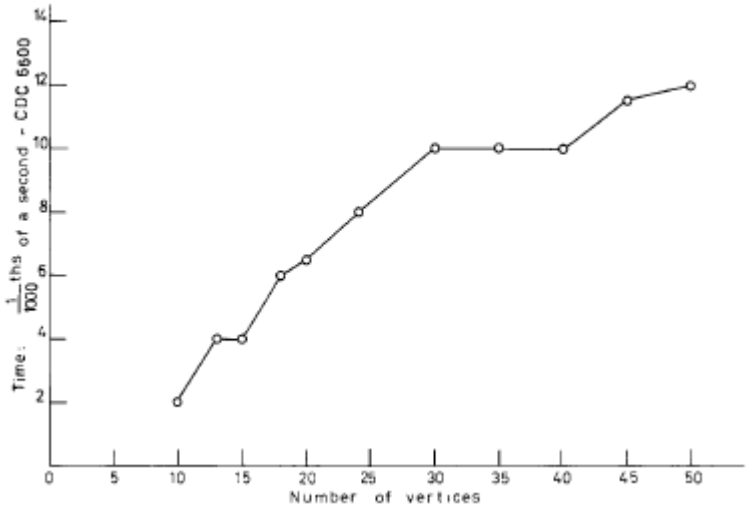

Figura 2.3: Comportamiento computacional del algoritmo de múltiples caminos

2.5. Un Problema Simple de Programación

En la introducción describimos el caso de una empresa que produce unidades $p_1, p_2, \ldots, p_n$ usando una sola instalación basada en una rotación cíclica y preguntamos sobre la secuencia de producción que requiere un menor número de restablecimientos de dicha instalación. Se construyó un grafo $G = (X, A)$ con vértices x_i que representan las unidades p_i $(i = 1, 2, \ldots, n)$ y arcos (x_i, x_j) que indican que la unidad p_j puede seguir a la unidad p_i en la instalación sin necesidad de restablecimiento.

Ahora bien, si el grafo G contiene un circuito Hamiltoniano (digamos $x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_n}$) entonces la correspondiente secuencia de artículos $p_{i_1}, p_{i_2}, p_{i_3}, \ldots, p_{i_n}$ puede ser fabricada por la instalación sin necesidad de restablecimiento, ya que, por definición de circuito Hamiltoniano:

$(x_{i_k}, x_{i_{k+1}}) \in A$ para todo $k = 1, 2, \ldots, (x_{i_{n+1}} \equiv x_{i_1})$ y A ha sido definido anteriormente como el conjunto $\{(x_i, x_j) \mid c_{ij} = 0\}$.

Si G no contiene circuito Hamiltoniano, podemos construir el grafo $G_1 = (X_1, A_1)$ donde:

$$X_1 = X \cup \{y_1\} \tag{2.2}$$

$$A_1 = A \cup \{(x, y_1) \mid x \in X\} \cup \{(y_1, x) \mid x \in X\} \tag{2.3}$$

i.e. se introduce un vértice ficticio y_1 en G junto a los arcos que van desde, y hacia, él a los vértices reales de G.

Si el grafo G_1 contiene un circuito Hamiltoniano, tendrá la forma: $x_{i_1}, x_{i_2}, \ldots, x_{i_r}$, $\boxed{y_1}$, $x_{i_{r+1}}, \ldots, x_{i_n}$, que puede ser interpretado como que los artículos se fabrican en la secuencia :

$$p_{i_{r+1}}, p_{i_{r+2}}, \ldots, p_{i_n}, p_{i_1}, p_{i_2}, \ldots, p_{i_r} \tag{2.4}$$

con una operación de restablecimiento entre el final del último artículo y el principio del primero en la secuencia. De esta forma, el vértice ficticio tiene como propósito marcar la posición en la secuencia donde es necesario un restablecimiento de la instalación. En términos del grafo, el vértice ficticio y_1 y sus arcos asociados producen un camino entre dos vértices reales. Así, si existe una secuencia de unidades con un restablecimiento, el hecho de añadir y_1 a G siempre causará la existencia de un circuito Hamiltoniano en G_1 ya

que el arco (x_i, x_j) necesario, puede ser reemplazado por dos arcos ficticios (x_i, y_1) y (y_1, x_j).

Si el grafo G_1 no contiene un circuito Hamiltoniano, construimos el grafo $G_2 = (X_2, A_2)$ desde el grafo G_1, donde:

$$X_2 = X_1 \cup \{y_2\} \tag{2.5}$$

$$A_2 = A_1 \cup \{(x, y_2) \mid x \in X\} \cup \{(y_2, x) \mid x \in X\} \tag{2.6}$$

y continuar de la misma forma.

Teorema 1. Si el grafo $G_m = (X_m, A_m)$ dado por:

$$X_m = X \cup \left[\bigcup_{j=1}^{m} \{y_j\}\right] \tag{2.7}$$

$$A_m = A \cup \left[\bigcup_{j=1}^{m} \{(x, y_j) \mid x \in X\}\right] \cup \left[\bigcup_{j=1}^{m} \{(y_j, x) \mid x \in X\}\right] \tag{2.8}$$

contiene un circuito Hamiltoniano, pero el grafo G_{m-1}, definido de forma similar, no lo contiene, entonces m es el número mínimo de restablecimientos de la instalación que son necesarios y si el circuito Hamiltoniano de G_m es:

$$x_{i_1}, \ldots x_{i_\alpha}, y_1, x_{i_{\alpha+1}}, \ldots, x_{i_\beta}, y_2, x_{i_{\beta+1}}, \ldots, x_{i_\gamma}, y_3, x_{i_{\gamma+1}}, \ldots, etc., \ldots, x_{i_\delta}, y_m, x_{i_{\delta+1}}, \ldots, x_{i_n} \tag{2.9}$$

entonces los artículos deben ser producidos en m secuencias dadas por:

$$p_{i_{\alpha+1}}, \ldots, p_{i_\beta} \text{ seguido por } p_{i_{\beta+1}}, \ldots, p_{i_\gamma}, \ldots, etc., \ldots \text{ seguido por } p_{i_{\delta+1}}, \ldots, p_{i_n}, p_{i_1}, \ldots, p_{i_\alpha} \tag{2.10}$$

La demostración de este teorema se obtiene inmediatamente por inducción sobre el argumento que precede a dicho teorema.

2.5.1. Aspectos Computacionales

En el razonamiento de la sección anterior, el grafo G se incrementó en un solo vértice ficticio en cada momento específico. Si la solución óptima del problema implica m restablecimientos de la instalación, entonces necesitamos $m + 1$ intentos para buscar circuitos Hamiltonianos en los grafos

$G, G_1, \ldots, G_m$, siendo solo el último de estos intentos exitoso y el que conduce a la solución del problema. Obviamente m está acotado inferiormente por n y, en general, para problemas prácticos m solo será una fracción muy pequeña de n. Sin embargo, es necesario desde el punto de vista computacional un proceso diferente para generar y probar los grafos de circuitos Hamiltonianos, ya que resulta [7] que es más rápido encontrar un circuito Hamiltoniano en un grafo que contiene solo uno que probar que no existe circuito Hamiltoniano en un grafo que no lo tenga. Este hecho sugiere que un algoritmo mejor sería uno que empezase por una cota superior B para el número (mínimo) óptimo de restablecimientos de la instalación m y, de forma secuencial, formar y probar los grafos G_B, G_{B-1}, etc. hasta que se encuentre un grafo G_{m-1} que no contenga un circuito Hamiltoniano. Demos una pequeña descripción de dicho algoritmo:

(1) Encontrar una cota superior B sobre el número (mínimo) óptimo de restablecimientos de la instalación que puedan ser necesarios. (Ver Apéndice A).

(2) Formar el grafo G_B de acuerdo a las ecuaciones (2.7) y (2.8)

(3) ¿Tiene G_B un circuito Hamiltoniano? Si la respuesta es afirmativa, almacenar el circuito en el vector H sobrescribiendo cualquier circuito anterior e ir al paso (4); si la respuesta es no, ir al paso (5).

(4) $B \leftarrow B - 1$, ir al paso (2).

(5) Stop. $m = B + 1$ es el número mínimo de restablecimientos de la instalación necesarios y la última secuencia en H es la secuencia de fabricación de artículos requerida.

El paso (1) del algoritmo anterior requiere alguna explicación adicional. Obviamente, cuanto más ajustada sea la cota superior inicial B, menor será el número de iteraciones del algoritmo principal. Un procedimiento para calcular una buena cota superior inicial viene descrito en el Apéndice A, mientras en el paso (3) anterior se puede comprobar la existencia de un circuito Hamiltoniano en el grafo G_B usando el algoritmo de los múltiples caminos visto en una sección anterior.

Capítulo 3

Métodos de Ramificación y Acotación

3.1. Introducción

El origen de la idea del método de ramificación y acotación comienza con el trabajo de Dantzig, Fulkerson y Johnson [13, 14] sobre el TSP. En algún sentido el TSP ha servido como un campo de prueba para el desarrollo de métodos resolutivos para la optimización discreta. El término ramificación y acotación fue acuñado por Little, Murty, Sweeny y Karel (1963) [49] en conjunción con su algoritmo del TSP.

Los métodos enumerativos (ramificación y acotación) resuelven un problema de optimización discreta rompiendo su conjunto factible en pequeños subconjuntos, calculando cotas del valor de la función objetiva en cada subconjunto que se crea, para usarlos en descartar ciertos subconjuntos en futuras consideraciones. Las cotas son obtenidas reemplazando el problema por un subconjunto dado con un problema (relajado) más sencillo. El proceso termina cuando cada subconjunto ha producido una solución factible o no se ha encontrado una mejor que la que ya se tenía. La mejor solución encontrada durante el proceso es un óptimo global.

Para cualquier problema P, denotamos por $v(P)$ el valor de una solución óptima del propio P. Los ingredientes principales de cualquier procedimiento de ramificación y acotación para un problema de optimización discreta P de la forma $min\{f(x) \mid x \in S\}$ son:

(i) una relajación de P, i.e. un problema R de la forma $min\{g(x) \mid x \in T\}$,

tal que S $\subseteq$ T y para todo $x, y \in S, f(x) < f(y)$ implica $g(x) < g(y)$;

(ii) una regla de separación o ramificación, i.e. una regla para romper el conjunto factible S_i del correspondiente subproblema P_i en subconjuntos $S_{i1}, ..., S_{iq}$, tal que $\bigcup_{j=1}^{q} S_{ij} = S_i$;

(iii) un procedimiento para la cota inferior, i.e. uno para encontrar (o aproximar inferiormente) $v(R_i)$ para el problema relajado R_i de cada subproblema P_i;

(iv) una regla de selección de subproblema, i.e. una regla para elegir el siguiente problema en proceder.

Otros ingredientes que no siempre están presentes pero son útiles serían:

(v) un proceso para la cota superior, i.e. uno para encontrar soluciones factibles de P;

(vi) un proceso de pruebas, i.e. uno para usar las implicaciones lógicas de las restricciones y las cotas para arreglar los valores de algunas variables o para descartar el problema completo.

Como el apartado más importante es (i), clasificaremos los métodos de ramificación y acotación para el TSP de acuerdo a la relajación que utilizan.

La formulación de programación entera del TSP a la que nos referiremos al discutir los diversos métodos de resolución está definido en un grafo dirigido completo $G = (V, A)$ en n vértices, con conjunto de vértices $V = \{1, ..., n\}$, conjunto de aristas $A = \{(i, j) \mid i, j = 1, ..., n\}$ y costes no negativos c_{ij} asociados. El hecho de que G sea completo no implica restricción, ya que a las aristas que queramos ignorar podemos asignarles el coste $c_{ij} = \infty$. El TSP puede ser formulado, según Dantzig, Fulkerson y Johnson [13], como el problema de minimizar

$$\sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{3.1}$$

sujeto a:

$$\sum_{j \in V} x_{ij} = 1, i \in V, \tag{3.2}$$

$$\sum_{i \in V} x_{ij} = 1, j \in V, \tag{3.3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subset V, S \neq \emptyset, \tag{3.4}$$

$$x_{ij} = 0, 1; \qquad i, j \in V, \tag{3.5}$$

donde $x_{ij} = 1$ si la arista (i,j) está en la solución, $x_{ij} = 0$ en caso contrario.

La inecuación para la eliminación de subciclos (3.4) puede ser escrita también como

$$\sum_{i \in S} \sum_{j \in V - S} x_{ij} \geq 1, \qquad \forall S \subset V, S \neq \emptyset \tag{3.6}$$

Un caso importante es el TSP simétrico, en el que $c_{ij} = c_{ji}$, para todo i, j. El TSP simétrico puede definirse en un grafo completo no dirigido G=(V,E) en n vértices, con conjunto de vértices V, conjunto de aristas E y costes arbitrarios c_{ij}. Se puede enunciar como el problema de minimizar

$$\sum_{i \in V} \sum_{j > i} c_{ij} x_{ij} \tag{3.7}$$

sujeto a:

$$\sum_{j < i} x_{ji} + \sum_{j > i} x_{ij} = 2, \qquad i \in V, \tag{3.8}$$

$$\sum_{i \in S} \sum_{j \in S, j > i} x_{ij} \leq |S| - 1, \qquad \forall S \subset V, S \neq \emptyset, \tag{3.9}$$

$$x_{ij} = 0, 1; \qquad i, j \in V, j > i, \tag{3.10}$$

donde la inecuación para eliminar subciclos (3.9) puede ser también escrito como

$$\sum_{i \in S} \sum_{j \in V - S, j > i} x_{ij} + \sum_{i \in V - S} \sum_{j \in S, j > i} x_{ij} \geq 2, \qquad \forall S \subset V, S \neq \emptyset \tag{3.11}$$

A continuación mostramos el esquema de dos versiones de un procedimiento de ramificación y acotación para el TSP. Antes de usar alguna de estas versiones, hay que elegir una relajación R del TSP. Ambas versiones llevan una serie de subproblemas. Se diferencian en que la versión 1 resuelve un subproblema (relajado) R_k solo cuando el nodo k es seleccionado y eliminado de la lista, mientras que la versión 2 resuelve cada subproblema (relajado) en cuanto se crea, i.e. antes de añadirlo a la lista. A pesar de que todos los procedimientos de ramificación y acotación usados en la práctica difieren entre sí en muchos detalles, todos ellos pueden verse como variantes de una de estas dos versiones.

Método de ramificación y acotación para el TSP

Versión 1

1. (Inicialización) Poner el TSP en la lista (de subproblemas activos). Inicializar la cota superior en $U = \infty$

2. (Selección de subproblema) Si la lista es vacía, stop: el ciclo asociado a U es óptimo (o, si $U = \infty$, TSP no tiene solución). En caso contrario, elige un subproblema TSP_i, de acuerdo a la regla de selección de subproblema y elimina TSP_i de la lista.

3. (Cota inferior) Resolver la relajación R_i del TSP_i o acotar por abajo $v(R_i)$ siendo L_i el valor obtenido.
   Si $L_i \geq U$, volver al paso (2).
   Si $L_i < U$ y la solución define un ciclo para el TSP, almacenarlo en el lugar del mejor ciclo anterior, hacer $U \leftarrow L_i$, e ir al paso (5).
   (Ahora, $L_i < U$ y la solución no define un ciclo).

4. (Cota superior: opcional) Usar un heurístico para encontrar un ciclo para TSP. Si se encuentra uno mejor que el actual, reemplazarlo por el último y actualizar U.

5. (Reducción: opcional) Eliminar del grafo del TSP_i todos los arcos cuya inclusión en un ciclo aumentaría su valor sobre U.

6. (Ramificación) Aplicar la regla de ramificación a TSP_i, i.e. generar nuevos subproblemas $TSP_{i1}, ..., TSP_{iq}$, colocándolos en la lista, e ir al paso (2).

Versión 2.

1. (Inicialización) Como en la versión 1, pero resolviendo R antes de colocar TSP en la lista.

2. (Selección de subproblema) Igual que en la versión 1.

3. (Cota superior: opcional) Igual que el paso (4) en la versión 1.

4. (Reducción: opcional) Igual que el paso (5) en la versión 1.

5. (Ramificación) Usar la regla de ramificación para definir el conjunto de subproblemas $TSP_{i1}, ..., TSP_{iq}$ que se generarán por el subproblema actual TSP_i.

6. (Cota inferior) Si todos los subproblemas que se generarán por TSP_i de acuerdo a la regla de ramificación ya han sido generados, ve al paso (2). En caso contrario, genera el siguiente subproblema TSP_{ij} definido por la regla de ramificación, resuelve la relajación R_{ij} del TSP_{ij} o acota por abajo $v(R_{ij})$ siendo L_{ij} el valor obtenido.
   Si $L_{ij} \geq U$, repite el paso (6).
   Si $L_{ij} < U$ y la solución define un ciclo para el TSP, se almacena en lugar del mejor ciclo anterior, hacer $U \leftarrow L_{ij}$, y repetir el paso (6).
   Si $L_{ij} < U$ y la solución no define un ciclo, coloca TSP_{ij} en la lista y repite el paso (6).

En ambas versiones, el procedimiento puede ser representado por un árbol enraizado cuyos nodos corresponden a los subproblemas generados, con el nodo raíz correspondiendo al problema original y los nudos sucesivos de un nodo i dado asociado con TSP_i corresponden a los subproblemas $TSP_{i1}, ..., TSP_{iq}$ definidos por la regla de ramificación.

Es fácil ver que bajo hipótesis leves de la regla de ramificación y la relajación usada, ambas versiones del proceso anterior son finitas.

Ahora discutiremos varias especializaciones del proceso mostrado anteriormente, clasificadas según la relajación usada. Cuando evaluamos y comparamos varias relajaciones, uno debe tener en cuenta que una buena relajación es una que: (i) de una cota inferior fuerte, i.e. produce una pequeña diferencia $v(TSP) - v(R)$; y (ii) sea fácil de resolver. Naturalmente, son objetivos difíciles generalmente.

3.2. El problema de asignación con la función coste del TSP

La relajación más sencilla del TSP, históricamente la primera en usarse, es el problema obtenido de la formulación de programación entera (3.1), (3.2), (3.3), (3.4), (3.5) eliminando las restricciones (3.4), i.e. el problema de asignación (PA) con la misma función coste que el TSP. Fue usada, entre otros, por Eastman [17], Little, Murty, Sweeney y Karel [49] o Shapiro [61].

Una asignación (i.e., una solución para el PA) es la unión de ciclos dirigidos, por lo tanto un tour o conjunto de subtours. Hay $n!$ asignaciones distintas, de las cuales $(n-1)!$ son tours. Por lo tanto, de media, una de cada n asignaciones es un tour. Además, en el contexto actual, solo son interesantes las asignaciones que no tienen elementos diagonales (i.e., satisfacen $x_{ii} = 0$, $i = 1, \ldots, n$) y el número de ello es $n!/e$ redondeado al entero más próximo, i.e. $\lfloor n!/e + 1/2 \rfloor$. Por lo tanto en media, una en cada n/e asignaciones 'sin diagonal' es un tour. Esta frecuencia relativamente alta de tours en las asignaciones sugiere que $v(PA)$ es probable que sea una gran cota en $v(TSP)$, y la experiencia computacional apoya esa teoría. Para comprobar cómo de buena es una cota para problemas generados aleatoriamente, se realizó el siguiente experimento [45]. Se generaron 400 problemas con $50 \leq n \leq 250$, con costes generados aleatoriamente de una distribución uniforme de los enteros en los intervalos [1,100] y [1,1000], resolviendo tanto el PA como el TSP. Se encontró que de media $v(PA)$ fue el 99,2 % de $v(TSP)$. Más allá, se descubrió que la cota mejora con el tamaño del problema, para problemas con $50 \leq n \leq 150$ y $150 \leq n \leq 250$ los resultados fueron 98.8 % y 99.6 % respectivamente.

El PA puede resolverse con el método Húngaro en como mucho $O(n^3)$ pasos (Kuhn [41], Christofides [8], Lawler [44]). Los problemas de asignación PA_i que se resuelven en cada nodo del árbol de búsqueda difieren del problema de asignación inicial PA en esos arcos que se han excluido, mientras que otros arcos se han incluido en la solución. Estas modificaciones no presentan dificultad. Debido a que el método Húngaro produce a cada iteración una cota inferior de $v(PA)$, el proceso de resolución de un subproblema puede pararse en cuanto la cota inferior coincida con la superior U. En el caso típico, el problema de asignación PA_j resuelto en el nodo j del árbol de búsqueda difiere del problema PA_i resuelto en el nodo i solo en el arco que pertenece a la solución óptima de PA_i y ha sido excluido de la solución de PA_j y posiblemente otros arcos son necesarios para mantener la misma posición con

respecto a la solución de PA_j que tiene con respecto a esa de PA_i. Cuando ocurra este caso, el problema PA_j puede resolverse con el método Húngaro empezando por la solución óptima del problema en el nodo en a lo sumo $O(n^2)$ pasos.

Esta cota inferior $v(PA)$ puede ser mejorada añadiendo una penalización. Puede ser calculada como el mínimo crecimiento en la función objetivo causado por el primer pívot simplex que elimina algún arco de la solución o por la primera iteración del método Húngaro. Es más, el arco que debe incluirse en la solución por el pívot puede ser restringido por algún subtour de la solución del PA. La experiencia computacional indica, de todas formas, que el impacto de esa penalización tiende a decrecer con el tamaño del problema y es despreciable siempre excepto en pequeños problemas. En el experimento computacional con los 400 problemas generados aleatoriamente, añadir una penalización a $v(PA)$ aumenta el valor de la cota inferior un 0.03 % de media, de 99.2 % a 99.23 % del $v(TSP)$.

3.2.1. Reglas de ramificación

Evaluando las ventajas y las desventajas de las diferentes reglas de ramificación deberíamos tener en cuenta que el objetivo último es resolver el TSP con el menor número de subproblemas posible. Por tanto, una buena regla de ramificación es una que (a) genera pocos sucesores de un nodo del árbol de búsqueda y (b) genera subproblemas fuertemente restrictivos, i.e. excluye muchas soluciones de cada subproblema. Estos criterios normalmente se enfrentan y los méritos de las reglas dependen de la compensación.

Discutiremos las diversas reglas de ramificación en términos de conjuntos de arcos E_k e I_k excluidos e incluidos, respectivamente, en la solución del subproblema k. El símbolo k es en general una notación que indica la ascendencia de cada subproblema, es decir, si $k = (pqr)$, entonces el subproblema k es el sucesor número r de (pq), que sucesivamente es el sucesor número q de p. En términos de las variables x_{ij}, la interpretación de los conjuntos E_k y I_k es que el subproblema k está definido por las condiciones

$$x_{ij} = \begin{cases} 0, & (i,j) \in E_k \\ 1, & (i,j) \in I_k \end{cases} \tag{3.12}$$

además de las restricciones del problema principal. Por tanto la relajación del subproblema k viene dada por (3.12) junto a (3.1), (3.2) y (3.4).

Regla de ramificación 1. (Little, Murty, Sweeney y Karel, 1963 [49]) Dado el correspondiente subproblema relajado PA_k y sus costes reducidos $\hat{c} = c_{ij} - u_i - v_j$, donde u_i y v_j son variables óptimas duales, para cada arco (i,j) tal que $\hat{c} = 0$ definir la penalización

$$p_{ij} = min\{\hat{c}_{ih} : h \in V - \{j\}\} + min\{\hat{c} : h \in V - \{i\}\} \tag{3.13}$$

y elegir $(r, s) \in A - E_k \cup I_k$ tal que

$$p_{rs} = max\{p_{ij} : \hat{c}_{ij} = 0\} \tag{3.14}$$

Entonces generar dos sucesores del nodo k, nodos $k1$ y $k2$, definiendo

$$E_{k1} = E_k \cup \{(r, s)\}, \quad I_{k1} = I_k \tag{3.15}$$

y

$$E_{k2} = E_k, \quad I_{k2} = I_k \cup \{(r, s)\} \tag{3.16}$$

Esta regla no usa la estructura especial del TSP y tiene la desventaja de que deja la solución óptima para PA_k factible para PA_{k2}.

Las reglas siguientes están basadas en disyunciones derivadas de las inecuaciones de eliminación de subtours del problema original.

Regla de ramificación 2. (Eastman, 1958 [17]; Shapiro, 1966 [61]) Sea x^k la solución óptima del subproblema relajado actual PA_k y sea $A_S = \{(i_1, i_2), \ldots, (i_t, i_1)\}$ el conjunto de arcos del subtour de x^k con cardinal mínimo, implicando el conjunto de vértices $S = \{i_i, \ldots, i_t\}$. La restricción (3.4) para S implica la inecuación

$$\sum_{(i,j)\in A_S} x_{ij} \leq |S| - 1 \tag{3.17}$$

Sin pérdida de generalidad, asumimos $A_S - I_k = \{(i_1, i_2), \ldots, (i_s, i_{s+1})\}$, con $s \leq t$. Entonces (3.17) implica

$$x_{i_i i_2} = 0 \vee \cdots \vee x_{i_s i_{s+1}} = 0 \tag{3.18}$$

(donde $s + 1$ se toma en módulo t).

Generar s sucesores del nodo k, definidos por

$$\left.\begin{array}{c} E_{kr} = E_k \cup \{(i_r, i_{r+1})\} \\ I_{kr} = I_k \end{array}\right\} r = 1, \ldots, s \tag{3.19}$$

Ahora x^k es claramente no factible para todo PA_{kr}, $r = 1, \ldots, s$, y la elección de un subtour más corto para la ramificación mantiene el número de nodos sucesores pequeño. Aún así, la disyunción (3.18) no define una partición del conjunto factible de PA_k y, en consecuencia, los diferentes sucesores de PA_k podrían tener soluciones en común. Este defecto se remedia con la siguiente regla, que difiere de la regla número 2 solo en que fortalece la disyunción (3.18) a una que defina una partición.

Regla de ramificación 3. (Murty, 1968 [52]; Bellmore y Malone, 1971 [4]; Smith, Srinivasan y Thompson, 1977 [62]) La disyunción (3.18) puede ser fortalecida a

$$(x_{i_1 i_2} = 0) \vee (x_{i_1 i_2} = 1, x_{i_2 i_3} = 0) \vee \cdots \vee (x_{i_1 i_2} = \cdots = x_{i_{s-1} i_s} = 1, x_{i_s i_{s+1}} = 0) \tag{3.20}$$

y de la misma forma (3.19) puede reemplazarse por

$$\left.\begin{array}{c} E_{kr} = E_k \cup \{(i_r, i_{r+1})\} \\ I_{kr} = I_k \cup \{(i_1, i_2), \ldots, (i_{r-1}, i_r)\} \end{array}\right\} r = 1, \ldots, s \tag{3.21}$$

Una versión un poco diferente de la regla 3 es reemplazar el conjunto de arcos A_S de un subtour de cardinal mínimo con la de un número mínimo de arcos libres.

Regla de ramificación 4. (Bellmore y Malone, 1971 [4]) Sea x^k y S definidas como antes. Las restricciones 3.6 implican la disyunción

$$(x_{i_1 j} = 0, j \in S) \vee (x_{i_2 j} = 0, j \in S) \vee \cdots \vee (x_{i_s j} = 0, j \in S) \tag{3.22}$$

Generar t sucesores del nodo k, definidos por

$$\left.\begin{array}{c} E_{kr} = E_k \cup \{(i_r, j) : j \in S\} \\ I_{kr} = I_k \end{array}\right\} r = 1, \ldots, t \tag{3.23}$$

Al igual que en el caso de la 2, la regla 4 hace x^k no factible para todos los problemas sucesores de PA_k, pero otra vez (3.22) no parte el conjunto

factible de PA_k. Ello se remedia con la siguiente regla, que difiere de la 4 solo en definir una partición.

Regla de ramificación 5. (Garfinkel, 1973 [23]) La disyunción (3.22) puede ser fortalecida a

$$\begin{array}{c}(x_{i_1 j} = 0, j \in S) \vee (x_{i_1 j} = 0, j \in V - S; x_{i_2 j} = 0, j \in S) \vee \ldots \\ \vee (x_{i_r j} = 0, j \in V - S, r = 1, \ldots, t-1; x_{i_t j} = 0, j \in S)\end{array} \tag{3.24}$$

y de la misma forma (3.23) se reemplaza por

$$\begin{array}{c}E_{kr} = E_k \cup \{(i_r, j) : j \in S\} \cup \{(i_q, j) : q = 1, \ldots, r-1, j \in V - S\} \\ I_{kr} = I_k\end{array} \tag{3.25}$$

Las reglas de ramificación 2 y 4 (o sus variantes fortalecidas, reglas 3 y 5), basadas en las restricciones de eliminación de subtours (3.17) y (3.6), respectivamente, generan el mismo número de sucesores de un nodo k dado. Sin embargo, la regla basada en la inecuación (3.6) genera subproblemas más estrictamente restringidos, i.e., excluye un mayor número de asignaciones del conjunto factible de cada problema sucesor que la regla basada en la inecuación (3.17). En efecto, con $|S| = k$, obtenemos el siguiente resultado.

Teorema 1. (Bellmore y Malone, 1971 [4]) Cada inecuación (3.17) elimina $\lfloor (n-k)!/e + 1/2 \rfloor$ asignaciones 'sin diagonal', mientras que cada inecuación (3.6) elimina $\lfloor (n-k)!/e + 1/2 \rfloor \cdot \lfloor k!/e + 1/2 \rfloor$ asignaciones 'sin diagonal'.

Demostración. Cada inecuación (3.17) elimina aquellas asignaciones 'sin diagonal' que contengan el subtour con conjunto de arcos A_S. Hay tantas asignaciones de esa forma como asignaciones 'sin diagonal' hay en el grafo completo definido en el conjunto de vértices $V - S$. Dicha cantidad es $\lfloor (n-k)!/e + 1/2 \rfloor$.

Por otra parte, cada inecuación (3.6) elimina aquellas asignaciones 'sin diagonal' que consisten en la unión de dos de esas asignaciones, una en el grafo completo definido en S, la otra en el grafo completo definido en $V - S$. Como el valor del último es $\lfloor (n-k)!/e + 1/2 \rfloor$ y el del primero es $\lfloor k!/e + 1/2 \rfloor$, el número de asignaciones 'sin diagonal' eliminadas por cada inecuación (3.6) es la enunciada en el teorema, q.e.d.

3.3. El problema del 1-árbol con función objetiva lagrangiana

Esta relajación fue usada por primera vez para el TSP por Held y Karp [32, 33] y Christofides [6].

Consideremos el TSP simétrico y el grafo completo no dirigido $G = (V, E)$ asociado. El problema de encontrar un subgrafo H de G con n aristas que minimiza la función coste (3.7) es una relajación del TSP simétrico. Dicho subgrafo H consiste en un árbol de expansión de G, más una arista extra. Podemos restringir H a la clase $\mathscr{T}$ de subgrafos del tipo anterior en donde algún vértice arbitrario de G, digamos vértice 1, tiene grado 2 y está contenido en el único ciclo de H. Por falta de un término mejor, los subgrafos de esta clase $\mathscr{T}$ se llaman 1-árboles. Para ver que encontrar un 1-árbol que minimice (3.7) es una relajación del TSP, es suficiente con darse cuenta que el conjunto de restricciones que define la familia $\mathscr{T}$ es (3.10) y

$$\sum_{i\in S}\sum_{j\in V'-S, j>i} x_{ij} + \sum_{i\in V'-S}\sum_{j\in S, j>i} x_{ij} \geq 1, \tag{3.26}$$

para todo $S \subset V' = V - 1, S \neq \emptyset$,

$$\sum_{i\in V}\sum_{j>i} x_{ij} = n, \tag{3.27}$$

$$\sum_{j\in V} x_{1j} = 2 \tag{3.28}$$

Aquí (3.26) es una debilitación de (3.11); (3.27) es la suma de todas las ecuaciones (3.8) divididas entre 2; y (3.28) es la primera ecuación (3.8).

El problema del 1-árbol de coste mínimo puede descomponerse en 2 problemas independientes:

(α) Encontrar un árbol de expansión de coste mínimo en $G - \{1\}$;

(β) Encontrar las dos aristas de coste mínimo entre aquellos incidentes en G con el vértice 1.

Las $n - 2$ aristas del árbol de expansión encontrados bajo (α), junto a las dos aristas encontradas bajo (β), forman un 1-árbol de coste mínimo en G.

Para resolver el problema (β) se requieren $O(n)$ comparaciones, mientras que el problema (α) puede resolverse eficientemente por los algoritmos de Dijkstra [16] y Prim [56] de complejidad $O(n^2)$, o por el algoritmo de Kruskal [40] de complejidad $O(|E|log|E|)$.

El número de 1-árboles en el grafo completo no dirigido G de n vértices puede calcularse de la siguiente forma: la cantidad de árboles de expansión distintos en $G - \{1\}$ es $(n-1)^{n-3}$ (fórmula de Cayley) y por cada árbol de expansión uno puede tomar $\binom{n-1}{2}$ 1-árboles distintos insertando dos aristas que unen el vértice 1 con el árbol. De esta forma el número de 1-árboles en G es $\frac{1}{2}(n-2)(n-1)^{n-2}$, que es bastante mayor que el número de soluciones del PA. Como G tiene $(n-1)!$ tours, el número medio de tours en los 1-árboles de un grafo completo no dirigido es uno en cada $\frac{1}{2}(n-2)(n-1)^{n-3}/(n-2)!$, por lo tanto, el problema 1-árbol de mínimo coste con la misma función objetivo del TSP es una relajación débil del TSP. En el experimento computacional anteriormente mencionado sobre 140 problemas simétricos generados aleatoriamente se resolvieron los correspondientes problemas 1-árbol y encontramos el valor de un 1-árbol óptimo que corresponda solo al 63 % de $v(TSP)$. De todas formas, esta relajación puede abordarse tomando las ecuaciones (3.8) sobre la función objetivo en una forma Lagrangiana y maximizando el Lagrangiano como una función de los multiplicadores.

El problema

$$\begin{aligned} L(\lambda) &= \min_{x \in X(\mathscr{T})} \{\sum_{i \in V} \sum_{j>i} c_{ij} x_{ij} + \sum_{i \in V} \lambda_i (\sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} - 2)\} \\ &= \min_{x \in X(\mathscr{T})} \{\sum_{i \in V} \sum_{j>i} (c_{ij} + \lambda_i + \lambda_j) x_{ij}\} - 2 \sum_{i \in V} \lambda_i \end{aligned} \tag{3.29}$$

donde λ es un n-vector y $X(\mathscr{T})$ es el conjunto de vectores de incidencia de los 1-árboles en G, i.e., el conjunto definido por (3.10), (3.26), (3.27) y (3.28) es una *relajación Lagrangiana* del TSP. De la última expresión en (3.29) y el hecho de que $X(\mathscr{T})$ contiene todos los tours, es sencillo ver que para cualquier λ, $L(\lambda) \leq v(TSP)$. La relajación Lagrangiana más potente viene dada obviamente por $\lambda = \hat{\lambda}$ tal que

$$L(\hat{\lambda}) = \max_{\lambda} L(\lambda) \tag{3.30}$$

El problema (3.30) es conocido como el dual Lagrangiano del TSP.

(3.30) es una relajación más fuerte que el problema 1-árbol con la función coste del TSP. En efecto, la experiencia computacional con problemas generados aleatoriamente ha producido valores de $L(\hat{\lambda})$ del 99 % del $v(TSP)$ de media, según Christofides.

De todas formas, resolver (3.30) es mucho más complicado que resolver un problema 1-árbol. La función objetivo de (3.30), i.e. la función $L(\lambda)$ de (3.29), es lineal a trozos y cóncava en λ. Held y Karp [33], quiénes usaron por primera vez (3.30) como una relajación del TSP, probaron métodos diferentes y descubrieron que un proceso iterativo similar al método de relajación de Agmon [1] y Motzkin y Schoenberg [51] era la mejor aproximación a este tipo de problema. El método se convirtió en una referencia estándar bajo el nombre de *optimización subgradiente*, como resultado de su exitoso uso en conjunción con el TSP.

El método de optimización subgradiente para resolver (3.30) comienza con un $\lambda = \lambda^0$ arbitrario (digamos el vector nulo) y en cada iteración k actualizamos λ^k como sigue. Encontrar $L(\lambda^k)$, i.e. resolver el problema (3.29) para $\lambda = \lambda^k$. Sea $H(\lambda^k)$ el 1-árbol óptimo encontrado. Si $H(\lambda^k)$ es un tour o si $v(H(\lambda^k)) \geq U$, paramos. En caso contrario, para $i \in V$, sea d_i el grado del vértice i en $H(\lambda^k)$. Entonces el n-vector con componentes $d_i^k - 2$, $i \in V$, es un subgradiente de $L(\lambda)$ en λ^k. Sea

$$\lambda_i^{k+1} = \lambda_i^k + t^k(d_i^k - 2), \qquad i \in V, \tag{3.31}$$

donde t^k es la "longitud de paso"definido por

$$t^k = \alpha(U - L(\lambda^k))/\sum_{i \in V}(d_i^k - 2)^2 \tag{3.32}$$

con $0 < \alpha \leq 2$. Entonces hacer $k \leftarrow k + 1$ y repetir el procedimiento.

Se observa que el método converge si $\sum_{k=1}^{\infty} t^k = \infty$ y $\lim_{k\to\infty} t^k = 0$. Se satisfacen estas condiciones si se empieza con $\alpha = 2$ y se reduce periódicamente α en un factor.

Ejemplo. Consideremos el TSP simétrico de ocho ciudades cuyo grafo se muestra en la figura (3.1). Inicialmente $U = 25$, $\alpha = 2$, $\lambda_i^0 = 0$ para $i = 1, \ldots, 8$. El 1-árbol óptimo, mostrado con líneas sombreadas en la figura (3.2), tiene $L(\lambda^0 = 21)$. En la iteración 0 tenemos:

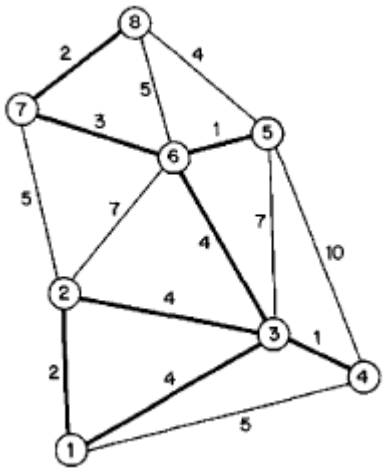

Figura 3.1: Grafo inicial $G = (V, E)$

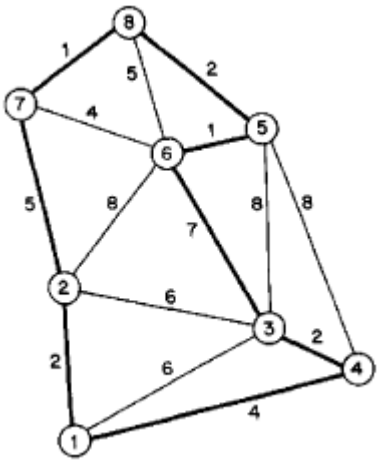

Figura 3.2: Grafo actualizado $G = (V, E)$

$$d_i^0 = (2, 2, 4, 1, 1, 3, 2, 1);$$
$$t^0 = 2(25 - 21)/8 = 1;$$
$$\lambda_i^1 = (0, 0, 2, -1, -1, 1, 0, -1)$$

Los costes de los arcos ya actualizados $(c_{ij} + \lambda_i^1 + \lambda_j^1)$ y el correspondiente 1-árbol óptimo, con $L(\lambda^1 = 24)$, se muestran en la figura (3.2).

Tenemos $d_i^1 = 2$ para $i = 1, \ldots, 8$; por lo que se ha encontrado un tour y el procedimiento acaba.

Held y Karp [33] descubrieron que si λ^0 se toma de forma que, en vez de 0, quede definido por

$$\lambda_i^0 = -(u_i + v_i)/2, \qquad i \in V, \tag{3.33}$$

donde (u, v) es una solución óptima del dual del problema de asignación con costes $c_{ij} = c_{ji}$ para todo i, j, entonces siempre tenemos $v(H(\lambda^0)) \geq v(PA)$. En efecto, para dicha elección de λ^0 tenemos,

$$\begin{aligned} L(\lambda)^0 &= \min_{x \in X(\mathscr{T})} \{\sum_{i \in V} \sum_{j > i} (c_{ij} + \lambda_i^0 + \lambda_j^0) x_{ij}\} - 2 \sum_{i \in V} \lambda_i^0 \\ &= \min_{x \in X(\mathscr{T})} \{\sum_{i \in V} \sum_{j > i} \frac{1}{2} [(c_{ij} - u_i - v_j) x_{ij} + (c_{ji} - u_j - v_i)]\} + \sum_{i \in V} (u_i + v_i) \\ &\geq v(PA), \end{aligned}$$

ya que $v(PA) = \sum_{i \in V} (u_i + v_i)$ y $c_{ij} - u_i - v_j \geq 0$, para todo i, j. Por supuesto, es necesario resolver el PA previamente.

Volgenant y Jonker [64] usaron una fórmula actualizada para λ^k y una expresión para t^k distintas de (3.31) y (3.32), respectivamente. Tomaron t^k como un escalar positivo de acuerdo a:

$$t^{k+1} - 2t^k + t^{k-1} = constante,$$

y definieron λ^{k+1} haciendo, para $i \in V$,

$$\lambda^{k+1} = \begin{cases} \lambda_i^k & si \;\; d_i^k = 2, \\ \lambda_i^k + 0{,}6t^k(d_i^k - 2) + 0{,}4t^k(d_i^{k-1} - 2) & en \;\; caso \;\; contrario. \end{cases}$$

Ninguna de las versiones de estos métodos de optimización subgradiente garantizan resolver (3.30) en tiempo polinomial con cierto grado de exactitud previsto. Aún así, las reglas de parada son tales que tras un cierto número de iteraciones el proceso termina con una aproximación λ, que proporciona una (buena, en general) cota inferior de $L(\hat{\lambda})$.

3.3.1. Reglas de ramificación

Regla de ramificación 6. (Held y Karp, 1971 [33]) En el nodo k, sean las aristas libres en el correspondiente 1-árbol (i.e. aquéllos en $E - E_k \bigcup I_k$) ordenadas de acuerdo a penalizaciones no crecientes y sean los primeros q elementos de este conjunto ordenado $J = \{\{i_1, j_1\}, \ldots, \{i_q, j_q\}$, donde q se especificará a continuación. Definimos q nuevos nodos, $k1, \ldots, kq$, como

$$\begin{aligned} I_{kr} &= I_k \cup \{\{i_h, j_h\} : h = 1, \ldots, r-1\}, \quad r = 1, \ldots, q, \\ E_{kr} &= E_k \cup \{\{i_r, j_r\}\}, \qquad\qquad\qquad\quad r = 1, \ldots, q-1, \\ E_{kq} &= E_k \cup \{\{i, j\} \notin I_{kq} : i = p \ o \ j = p\}. \end{aligned}$$

Aquí $p \in V$ es tal que I_k contiene al menos una arista incidente con p, mientras que I_{kq} contiene dos de esas aristas; q es el menor índice de una arista en J para el cuál existe un vértice con las propiedades de p.

Esta regla divide el conjunto factible y hace infactible al correspondiente 1-árbol para cada uno de los nuevos subproblemas generados, aunque el número q de los nuevos subproblemas suele ser mayor que el necesario.

Regla de ramificación 7. (Smith y Thompson, 1977 [62]) Elegir un vértice i de G cuyo grado en el 1-árbol correspondiente no sea 2 y una arista $\{i, j\}$ de coste máximo en el 1-árbol. Entonces generamos nuevos subproblemas definidos por

$$\begin{aligned} E_{k1} &= E_k \cup \{\{i, j\}\}, \qquad & I_{k1} &= I_k, \\ E_{k2} &= E_k, & I_{k2} &= I_k \cup \{\{i, j\}\}. \end{aligned} \tag{3.34}$$

Esta regla genera solo dos sucesores en cada nodo k del árbol de búsqueda, pero el mínimo 1-árbol en el subproblema k permanece factible para el subproblema $k2$.

Regla de ramificación 8. (Volgenant y Jonker, 1982 [64]) Elegir un vértice i cuyo grado en el 1-árbol correspondiente sea mayor que 2 y 2 aristas libres $\{i, j_1\}$, $\{i, j_2\}$ en el 1-árbol. Generar tres nuevos subproblemas:

$$\begin{aligned} E_{k1} &= E_k \cup \{\{i,j\} : j \notin \{j_1, j_2\}\}, & I_{k1} &= I_k \cup \{\{i, j_1\}, \{i, j_2\}\}, \\ E_{k2} &= E_k \cup \{\{i, j_2\}\}, & I_{k2} &= I_k \cup \{\{i, j_1\}\}, \\ E_{k3} &= E_k \cup \{\{i, j_1\}\}, & I_{k3} &= I_k. \end{aligned}$$

Si i es incidente con una arista en I_k, entonces no se genera el nodo $k1$.

Esta regla divide el conjunto factible y convierte al 1-árbol en el nodo k en infactible para cada uno de los nodos sucesores, mientras que el número de sucesores en cada nodo es como mucho tres.

Regla de ramificación 9. (Gavish y Srikanth, 1983 [24]) Elegir una arista $\{i, j\}$ del correspondiente 1-árbol tal que la penalización de hacer $x_{ij} = 1$ sea máxima (con reglas auxiliares para romper posibles lazos creados) y generar dos nuevos subproblemas definidos por (3.34).

3.3.2. Extensión al TSP asimétrico

Las ideas básicas de la relajación 1-árbol del TSP simétrico funcionan en el caso asimétrico [32]; en él, el 1-árbol en un grafo no dirigido puede reemplazarse por una 1-arborescencia en el grafo dirigido $G = (V, A)$, definido como una arborescencia (árbol dirigido) con raíz en el vértice 1, más un arco $(i, 1)$ uniendo un vértice $i \in V - \{1\}$ al vértice 1. Las restricciones que definen una 1-arborescencia, esto es (3.5) y

$$\begin{aligned} \sum_{i \in S} \sum_{j \in V - S} x_{ij} \geq 1, \qquad & \forall S \subset V : 1 \in S, \\ \sum_{i \in V} \sum_{j \in V} x_{ij} = n, & \\ \sum_{i \in V} x_{i1} = 1, & \end{aligned}$$

son una relajación del conjunto de restricciones (3.4), (3.5) y (3.6) del TSP.

El problema de encontrar una 1-arborescencia de coste mínimo puede descomponerse en dos problemas independientes, que son (α) encontrar una arborescencia de coste mínimo en G con raíz en el vértice 1, y (β) encontrar un arco $(i, 1)$ de coste mínimo en G. El problema (α) puede resolverse con los algoritmos de tiempo polinomial de Edmonds [18] o Fulkerson [22], o por el algoritmo de tiempo $O(n^2)$ de Tarjan [63].

Para obtener la versión Lagrangiana de la relajación 1-arborescencia, formamos la función

$$\begin{aligned} L(\lambda) &= \min_{x \in X(\mathscr{A})} \{\sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} + \sum_{i \in V} \lambda (\sum_{j \in V} x_{ij} - 1)\} \\ &= \min_{x \in X(\mathscr{A})} \{\sum_{i \in V} \sum_{j \in V} (c_{ij} + \lambda_i) x_{ij}\} - \sum_{i \in V} \lambda_i, \end{aligned}$$

donde $X(\mathscr{A})$ es el conjunto de vectores incidentes de $\mathscr{A}$, la familia de 1-arborescencias en G. Una vez más, la cota inferior más fuerte de $v(TSP)$ viene dada por $\lambda = \hat{\lambda}$ tal que

$$L(\hat{\lambda}) = \max_{\lambda} L(\lambda), \tag{3.35}$$

y puede usarse optimización subgradiente para resolver el problema (3.35). De todas formas, la experiencia computacional con esta relajación muestra que es inferior (para problemas asimétricos) a la de la relajación PA, incluso cuando ésta última usa la función objetivo original del TSP [45].

3.4. Ejemplo de TSP con método de ramificación.

Sea una instancia del TSP formada por 7 ciudades, cuyas distancias entre ellas x_{ij} vienen determinadas en la tabla (3.1)

Resolvemos el problema de asignación correspondiente utilizando el método Húngaro. Lo realizamos computacionalmente y obtenemos la solución:

$$x_{12} = x_{21}, x_{35} = x_{54} = x_{46} = x_{67} = x_{73} = 1, \;\; Z = 103.$$

Observamos que este recorrido contiene dos subcircuitos representados en la figura (3.3)

Desde/Hasta	C. 1	C. 2	C. 3	C. 4	C. 5	C. 6	C. 7
Ciudad 1		3	93	13	33	9	57
Ciudad 2	4		77	42	21	16	34
Ciudad 3	45	17		36	16	28	25
Ciudad 4	39	90	80		56	7	91
Ciudad 5	28	46	88	33		25	57
Ciudad 6	3	88	18	46	92		7
Ciudad 7	44	26	33	27	84	39	

Cuadro 3.1: Subproblema 1.

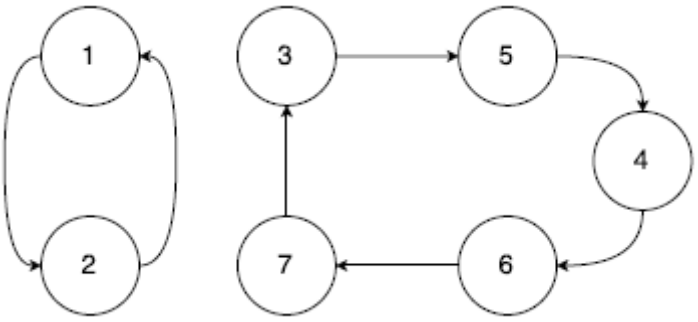

Figura 3.3: Subcircuitos del primer subproblema.

$$\{1-2-1\}\ y\ \{3-5-4-6-7-3\}.$$

Por simplicidad, decidimos romper el subcircuito $\{1-2-1\}$ en dos subproblemas:

•Subproblema 2: restricciones del subproblema 1 junto con $x_{21}=0$.

•Subproblema 3: restricciones del subproblema 1 junto con $x_{12}=0$.

Resolvemos primero el subproblema 2. Para ello, vamos a resolver el mismo problema de asignación, pero pondremos que la distancia del recorrido de la Ciudad 2 a la Ciudad 1 es ∞ (o un número muy grande M) para que el método de asignación nunca escoja x_{21}.

El subproblema 2 quedará entonces como en la figura (3.2):

Desde/Hasta	C. 1	C. 2	C. 3	C. 4	C. 5	C. 6	C. 7
Ciudad 1		3	93	13	33	9	57
Ciudad 2	M		77	42	21	16	34
Ciudad 3	45	17		36	16	28	25
Ciudad 4	39	90	80		56	7	91
Ciudad 5	28	46	88	33		25	57
Ciudad 6	3	88	18	46	92		7
Ciudad 7	44	26	33	27	84	39	

Cuadro 3.2: Subproblema 2.

Al resolverlo, obtenemos la siguiente solución:

$$x_{12} = x_{25} = x_{54} = x_{46} = x_{61} = 1, x_{37} = x_{73} = 1, \;\; Z = 125.$$

De la misma manera resolvemos el subproblema 3. Continuaremos ramificando (si es necesario) por el subproblema cuyo coste sea menor. Realizando de igual forma el cambio en la matriz de costes que en el paso anterior, obtenemos:

$$x_{14} = x_{46} = x_{67} = x_{73} = x_{35} = x_{52} = x_{21} = 1, \;\; Z = 126.$$

Hemos obtenido una solución factible, pues no contiene subcircuitos, pero puede no ser la óptima, ya que el subproblema 2, tenía un coste inferior a $Z = 126$, luego tenemos que seguir ramificando este subproblema, o bien hasta llegar a una solución factible de coste menor o igual que 126, o bien hasta que obtengamos ramas con coste mayor que este valor.

Seguimos por lo tanto ramificando el subproblema 2 cuya solución tiene dos subcircuitos:

$$\{1-2-5-4-6-1\} \; y \; \{3-7-3\}.$$

De nuevo decidimos romper el subcircuito corto, $\{3-7-3\}$, y nos vuelven a quedar dos subproblemas:

•Subproblema 4: restricciones del subproblema 2 junto con $x_{37} = 0$.

•Subproblema 5: restricciones del subproblema 2 junto con $x_{73} = 0$.

Resolvemos el subproblema 4, poniendo en la matriz de costos del subproblema 2, la entrada $c_{37} = M$, con M un número muy grande. La solución de este nuevo problema de asignación es:

$$x_{14} = x_{46} = x_{67} = x_{73} = x_{32} = x_{25} = x_{51} = 1, \;\; Z = 126.$$

Esta solución es factible al no tener subcircuitos y, además, tiene el mismo valor objetivo que la solución factible que habíamos encontrado anteriormente. Pero todavía no podemos afirmar que $Z = 126$ sea el valor óptimo pues aún nos queda abierta la rama del subproblema 5.

En el subproblema 5, obtenemos la solución:

$$x_{12} = x_{25} = x_{51} = 1, x_{37} = x_{74} = x_{46} = x_{63}, \;\; Z = 129.$$

Como el coste total de este caso es mayor que 126, ninguna solución que venga de un subproblema de éste, podrá ser óptima.

Como podemos observar, si ramificamos más cualquiera de las hojas existentes, el coste total será mayor que $Z = 126$, coste de una solución factible. Concluimos por tanto, que ambos recorridos obtenidos en los subproblemas 5 y 3, son soluciones óptimas del TSP propuesto.

En este ejemplo, hemos usado un algoritmo concreto para el problema del viajante basado en en la ramificación y acotación. Sin embargo, no tiene por qué ser el más eficiente.

3.5. Ejemplo de TSP con las provincias de Andalucía.

Consideramos el grafo (3.4) con las provincias de Andalucía, conectadas entre sí con sus correspondientes distancias. Aunque en la realidad las carreteras no unan las ciudades en líneas rectas, hemos simplificado el problema en dicho grafo.

Las distancias tomadas (en kilómetros) son reales. Vamos a resumirlas en la tabla (3.3)

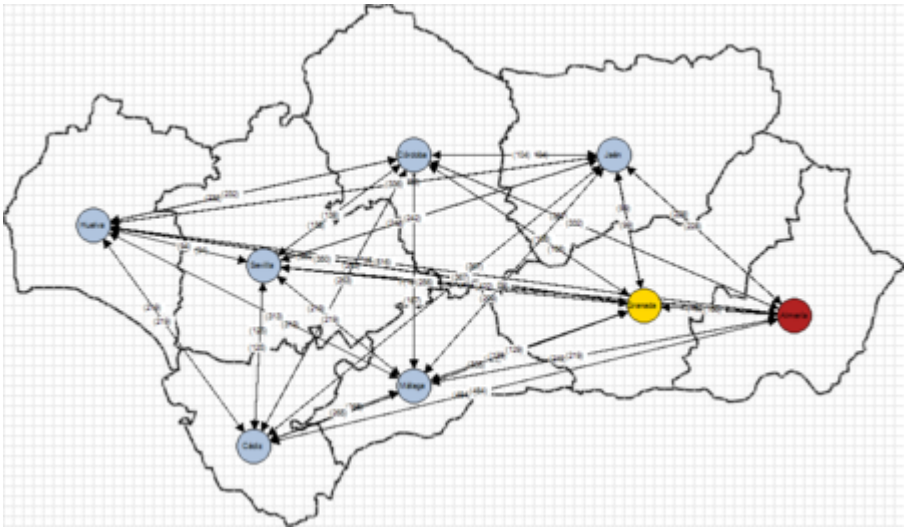

Figura 3.4: Grafo de las provincias de Andalucía.

	Almería	Cádiz	Córdoba	Granada	Huelva	Jaén	Málaga	Sevilla
Almería		484	332	160	516	228	219	422
Cádiz	484		263	335	219	367	265	125
Córdoba	332	263		166	232	104	187	138
Granada	160	335	166		350	99	129	256
Huelva	516	219	232	350		336	313	94
Jaén	228	367	104	99	336		209	242
Málaga	219	265	187	129	313	209		219
Sevilla	422	125	138	256	94	242	219	

Cuadro 3.3: Distancias entre provincias de Andalucía.

Supongamos que nos interesa recorrer un camino que una todas las capitales de provincia de Andalucía, comenzando y acabando en la misma ciudad, de forma que minimicemos los kilómetros recorridos. Introducimos los datos en el programa realizado por Xpress y obtenemos el siguiente resultado:

•La longitud total del recorrido es 1298 kilómetros.

•La ruta a seguir sería: ['*Granada*', *'Almería'*, *'Málaga'*, *'Cádiz'*, *'Sevilla'*, *'Huelva'*, *'Córdoba'*, *'Jaén'*, *'Granada'*].

Apéndice A

Algoritmo para la sección 2.5.1.

Mostremos un algoritmo para calcular una cota superior B del número mínimo de restablecimientos de la instalación requerido en el problema de la sección (2.5.1.), usado para iniciar el algoritmo de dicha sección.

(1) índice $\leftarrow 0$, $B \leftarrow 0$.

(2) Establecer las etiquetas $l(x_i) = 0 \ \forall \ x_i \in X$. Establecer $p = 0$.

(3) $G = (X, A)$, elegir cualquier $x_0 \in X$, establecer $S = \{x_0\}$

(4) Si índice $= 0$, formar $\hat{S} = S \cup \Gamma(S)$; si no, formar $\hat{S} = S \cup \Gamma^{-1}(S)$

donde: $\Gamma(x_i) = \{x_j \mid (x_i, x_j) \in A\}$ y $\Gamma(S) = \bigcup_{x_i \in S}(\Gamma(x_i))$

y también: $\Gamma^{-1}(x_i) = \{x_j \mid (x_j, x_i) \in A\}$ y $\Gamma^{-1}(S) = \bigcup_{x_i \in S}(\Gamma^{-1}(x_i))$

(5) Si $\hat{S} = S$ ir al paso (6), si no $p \leftarrow p + 1$, $l(x_i) \leftarrow p \ \forall \ x_i \in \hat{S} - S$, $\hat{S} \leftarrow S$ y volver al paso (4).

(6) Hallar $x \in \{x_i \mid l(x_i) = p\}$.

(7) Si índice $= 0$ hallar un $x' \in \{x_i \mid l(x_i) = p - 1 \ \textit{and} \ (x', x) \in A\}$.
en caso contrario: hallar un $x' \in \{x_i \mid l(x_i) = p - 1 \ \textit{and} \ (x, x') \in A\}$

(8) $X \leftarrow X - \{x\}$, $A \leftarrow A - \{(x, x_i) \mid x_i \in X\} - \{(x_i, x) \mid x_i \in X\}$.
Si $x = \{x_0\}, B \leftarrow B + 1$ e ir al paso (13); si no, ir al paso (9).

(9) $x \leftarrow x', p \leftarrow p - 1$. Si $x' = x_0$ ir al paso (10); si no, ir al paso (6).

(10) Si índice = 0, índice $\leftarrow 1$ e ir al paso (2); si no, ir al paso (11).

(11) Si $X = \{x_0\}$, $B \leftarrow B + 1$ e ir al paso (13); en caso contrario, ir al (12).

(12) índice $\leftarrow 0$, $B \leftarrow B + 1$.
$X \leftarrow X - \{x_0\}$, $A \leftarrow A - \{(x_0, x_i) \mid x_i \in X\} - \{(x_i, x_0) \mid x_i \in X\}$.

(13) Stop. B es la cota superior buscada.

El algoritmo requiere alguna explicación. Cuando índice = 0 los caminos *hacia adelante* se recorren a través de los vértices G empezando por el x_0. Estos caminos se recorren etiquetando con p los vértices de G que requieran p arcos para ser alcanzado desde x_0 (Pasos 2, 3, 4 y 5). Cuando no se puede ampliar ninguno de estos caminos, el algoritmo ejecuta los pasos 6, 7 y 8 que recorren el camino más largo hacia atrás hasta el vértice x_0 borrando del grafo los vértices (y sus arcos asociados) que se encuentran en el camino más largo. El paso (9) termina el proceso de eliminación. El paso (10) hace volver al algoritmo al principio con índice = 1 para comenzar a formar caminos *hacia atrás*, i.e caminos que acaban en x_0. Una vez más, el camino más largo que se encuentra, se elimina.

Los caminos hacia adelante y hacia atrás más largos (desde o hacia x_0), pueden considerarse conjuntamente como un único camino que contiene a x_0. El número B (de secuencia de caminos necesarios), aumenta una unidad en el paso (12) y el proceso sigue eligiendo otro vértice x_0 del grafo restante para formar nuevas secuencias hacia adelante y hacia atrás, etc. hasta que el grafo se agota.

El valor final de B, que es el número de la secuencia de caminos usado para borrar el grafo completamente (i.e. cubrir todos los vértices) es entonces, obviamente, una cota superior del número mínimo de secuencias necesarias (i.e. el número mínimo de restablecimientos de la instalación requeridos).

Bibliografía

[1] Agmon, S. *The relaxation method for linear equalities.* Canad. J. Math. 6, 1954.

[2] Beardwood, J., Halton, J., and Hammersley, J. *The shortest path through many points.* Proc. Cambridge Philos. Soc. 55, 1959.

[3] Bellman, R. *Dynamic programming treatment of the travelling salesman problem.* J. Assoc. Comput. Mach. 9, 1962.

[4] Bellmore, M., and Malone, J. *Pathology of traveling-salesman subtour-elimination algorithms.* Oper. Res. 19, 1971.

[5] Birkhoff, G. *Tres observaciones sobre el álgebra lineal.* Rev. Univ. Nac. Tucuman Ser., 1946.

[6] Christofides, N. *The shortest Hamiltonian chain of a graph.* SIAM J. Appl. Math. 19, 1970.

[7] Christofides, N. *Large scheduling problems with bivalent costs.* The Computer Jl., 1973.

[8] Christofides, N. *Graph Theory. An Algorithmic Approach.* Academic Press, London, 1975.

[9] Christofides, N. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem.* Graduate School of Industrial Administration., 1976.

[10] Cook, S. *The complexity of theorem-proving procedures.* Proc. 3rd Annual ACM Symp. Theory of Computing, 1971.

[11] Danielson, G. *On finding the simple paths and circuits in a graph.* IEEE Trans., 1968.

[12] Dantzig, G., and Fulkerson, D. *Minimizing the number of tankers to meet a fixed schedule.* Naval Res. Logist. Quart. I, 1954.

[13] Dantzig, G., Fulkerson, D., and Johnson, S. *Solution of a large-scale traveling-salesman problem.* Oper. Res. 2, 1954.

[14] Dantzig, G., Fulkerson, D., and Johnson, S. *On a linear-programming combinatorial approach to the traveling-salesman problem.* Oper. Res. 7, 1959.

[15] Dhawan, V. *Hamiltonian circuits and related problems in graph theory.* M. Soc. Report, Imperial College, London, 1969.

[16] Dijkstra, E. *A note on two problems in conexxion with graphs.* Numer. Math. I, 1959.

[17] Eastman, W. *Linear Programming with Pattern Constraints.* PhD thesis Harvard University, 1958.

[18] Edmonds, J. *Optimum branchings.* J. Res. Nat. Bur. Standards, 1967.

[19] Few, L. *The shortest path and the shortest road through n points.* Mathematika 2, 1955.

[20] Flood, M. M. *The traveling-salesman problem.* Oper. Res. 4, 1956.

[21] Ford, L., and Fulkerson, D. *Maximal flow through a network.* Canad. J. Math., 1956.

[22] Fulkerson, D. *Packing rooted directed cuts in a weighted directed graph.* Math. Programming 6, 1974.

[23] Garfinkel, R. *On partitioning the feasible set in a branch-and-bound algorithm for the asymmetric traveling salesman problem.* Oper. Res. 21, 1973.

[24] Gavish, B., and Srikanth, K. *Efficient Branch and Bound Code for Solving Large Scale Traveling Salesman Problems to Optimality.* University of Rochester, 1983.

[25] Gilmore, P., and Gomory, R. *Sequencing a one state-variable machine: a solvable case of the traveling salesman problem.* Oper. Res. 12, 1964.

[26] Gomory, R. *Outline of an algorithm for integers solutions to linear programs.* Bull Amer. Math. Soc. 64, 1958.

[27] Gomory, R. *Solving linear programming problems in integers.* Proc. Sympos. Appl. Math. 10, 1960.

[28] Gomory, R. *An algorithm for integer solutions to linear programs.* Recent advances in Mathematical Programming, 1963.

[29] Graham, R. *Bounds for certain multiprocessing anomalies.* Bell System Tech., 1966.

[30] Graham, R. *Bounds on multiprocessing timing anomalies.* SIAM J. Appl. Math. 17, 1969.

[31] Held, M., and Karp, R. *A dinamic programming approach to sequencing problems.* SIAM J. Appl. Math. 10, 1962.

[32] Held, M., and Karp, R. *The traveling salesman problem and minimum spanning trees.* Oper. Res. 18, 1970.

[33] Held, M., and Karp, R. *The traveling salesman problem and minimum spanning trees: part II.* Math. Programming I, 1971.

[34] Heller, I. *On the traveling salesman's problem.* Proc. Second Symp. Linear Programming, 19545.

[35] Hitchcock, F. *The distribution of a product from several sources to numerous localities.* J. Math. and Phys., 1941.

[36] Johnson, S. *Optimal two -and three- stage production schedules with setup times included.* Naval Res. Log. Quart. I, 1954.

[37] Karp, R. *Reducibility among combinatorial problems.* Complexity of Computer Computations, 1972.

[38] Karp, R. *Probabilistic analysis of partioning algorithms for the traveling-salesman problem in the plane.* Math. Oper. Res. 2, 1977.

[39] Kirkman, T. *On the representation of polyhedra.*

[40] Kruskal, J. *On the shortest spanning subtree of the graph and the traveling salesman problem.* Proc. Amer. Math. Soc. 7, 1956.

[41] Kuhn, H. *The Hungarian method for the assignament problem.* Naval Res. Logist. Quart. 2, 1955.

[42] Kuhn, H. *On certain convex polyhedra (abstract).* Bull Amer. Math. Soc., 1955.

[43] Land, A., and Doig, A. *An automatic method of solving discrete programming problems.* Econometrica 28, 1960.

[44] Lawler, E. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[45] Lawler, E., Lenstra, J., Rinooy, A., and Shmoys, D. *Traveling Salesman Problem: Guided Tour of Combinatorial Optimization.* Wiley, 1985.

[46] Levin, L. *Universal sequential search problems.* Problemy Peridachi Informatsii, 1973.

[47] Lin, S. *Computer Solutions of the traveling salesman problem.* Bell System Tech., 1965.

[48] Lin, S., and Kernighan, B. *An effective heuristic algorithm for the traveling salesman problem.* Oper. Res. 21, 1973.

[49] Little, J., Murty, K., Sweeney, D., and Karel, C. *An algorithm for the traveling salesman problem.* Oper. Res. 11, 1963.

[50] Menger, K. *Das Botenproblem.* Kolloquium 12, 1930.

[51] Motzkin, T., and Schoenberg, I. *The relaxation method for linear inequalities.* Canad. J. Math. 6, 1954.

[52] Murty, K. *An algorithm for ranking all the assignments in order of increasing cost.* Oper. Res. 16, 1968.

[53] Nash-Williams, C. *On Hamiltonian circuits in finite graphs.* American Mathematical Soc., 1966.

[54] Ore, O. *Theory of Graphs.* American Mathematical Society, 1962.

[55] Posa, L. *A theorem concerning Hamiltonian lines.* 1962.

[56] Prim, R. *Shortest connection networks and some generalizations.* Bell System Tech J. 36, 1957.

[57] Roberts, S., and Flores, B. *Systematic generation of Hamiltonian circuits.* Comm. of ACM, 1966.

[58] Roberts, S., and Flores, B. *An engineering approach to the travelling salesman problem.* Man. Sci., 1967.

[59] Sahni, S., and Gonzalez, T. *P-complete approximation problems.* J. Assoc. Comput. Mach. 23, 1976.

[60] Selby, G. The use of topological methods in computer-aided circuit layout.

[61] Shapiro, D. *Algorithms for the Solution of the Optimal Cost and Bottleneck Traveling Salesman Problems.* Sc. D. thesis Washington University, 1966.

[62] Smith, T., Srinivasan, V., and Thompson, G. *Computational performance of three subtour elimination algorithms for solving asymmetric traveling salesman problems.* Ann. Discrete Math. 1, 1977.

[63] Tarjan, R. *Finding optimum branchings.* Networks 7, 1977.

[64] Volgenant, T., and Jonker, R. *A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation.* European J. Oper. Res. 9, 1982.

[65] Yau, S. *Generation of all Hamiltonian circuits, paths and centres of a graph and related problems.* IEEE Trans., 1967.