TRABAJO FIN DE GRADO

El método Húngaro de Asignación: Aplicaciones

Realizado por: Danae López Reyes

Supervisado por: Eduardo Conde Sánchez

FACULTAD DE MATEMÁTICAS

DPTO DE ESTADÍSTICA E INVESTIGACIÓN OPERATIVA

Abstract

The linear sum assignment problem is one of the most interesting problems in linear programming and in combinatorial optimization. It consists of finding a minimum weight perfect matching in a weighted bipartite graph.

It is a problem that has interested many people throughout history, because it has many applications,from the allocation of resources in military operations in the Second World War, to the configuration of satellite communication and other communication technologies. Other applications that will be considered specifically in this study are the following:

- The traveling salesperson (or, salesman) problem (TSP) is a well-known and important combinatorial optimization problem. The goal is to find the shortest tour that visits each city in a given list exactly once and then returns to the starting city. As it will be shown, the solution of the assignment problem determines a lower bound of the optimal objective of the TSP and it can be used as the starting solution of a cutting algorithm for the TSP.

- Matrix visualization: Permutation of the rows and columns of matrices to analyze multivariate data with medium to low sample size. The

idea is to make a data matrix more understandable by simultaneously rearranging rows (variables) and columns (cases) + grouping. This process is related to the TSP and hence, the solution of the assignment problem could be considered as an initial rearrangement that, possibly, should be improved in order to have an optimal representation of the matrix.

The Hungarian algorithm is the most relevant procedure that have been devised in order to solve the linear assignment problem. The genesis of this method was reported by the author H. W. Kuhn. In 1953, while reading the classical graph theory book by Konig, he encountered an efficient algorithm for determining the maximum cardinality of a bipartite graph. In this essay we will describe this method, study its complexity and show the main properties that make it one of the most useful algorithm in Operations Research.

Índice general

Capítulo 1

Introducción

El problema de Asignación es uno de los problemas más importantes de la Programación Lineal. Prueba de ello es que en Google, al buscar 'Assignment Problem' aparecen alrededor de 2.390.000 resultados. Es un problema que ha interesado a diferentes colectivos por todas las aplicaciones que tiene, desde asignar los escasos recursos en operaciones militares hasta la comunicación por satélite. En su forma más general, el problema es como sigue:

Hay un número de trabajadores y tareas. Cualquier trabajador puede ser asignado para desarrollar cualquier tarea, produciéndose algún coste que puede variar dependiendo del trabajador y la tarea asignados. El problema consiste en asignar un trabajador a cada tarea de modo que la suma de los costes sea minimizada.

Se planteará el problema con una matriz de costes $C = c_{ij}$ y se buscará la asignación de cada fila (trabajadores) a cada una de las columnas (tareas).

Como se verá más adelante, el problema posee propiedades interesantes desde el punto de vista de la Programación Matemática. Es especialmente

importante la propiedad de unimodularidad que permitirá emplear toda la potencia de los resultados existentes en Programación Lineal. En el capítulo 2 indagaremos algo más en la estructura matemática del problema.

El capítulo 3 recoge algunos detalles históricos de la evolución en la metodología de resolución.

El capítulo 4 está dedicado a analizar en profundidad el método utilizado para resolver el problema de Asignación, el Método Húngaro. Además veremos que se pueden especializar los algoritmos existentes en este área con la idea de hacerlos más eficientes.

En el capítulo 5 vemos como el problema de asignación esta muy relacionado con el problema del circuito del viajante y se propone para resolverlo un algoritmo de planos de corte que parte de la solución proporcionada por el método de asignación. Además, vemos que ambos tienen gran cantidad de aplicaciones (algunas de las cuales veremos en este trabajo) como por ejemplo: La comunicación por satélite, la visualización de datos o la impresión de las placas de los circuitos.

Capítulo 2

El problema de asignación

El problema de asignación (PA) es uno de los problemas más interesantes de la Programación lineal. En su formato más básico, el problema consiste en asignar a cada trabajador una tarea, sabiendo que hay el mismo número de tareas que de trabajadores, n.

Los datos necesarios para plantear un problema de asignación serán:

Una matriz de costes de tamaño $n \times n$, $C = c_{ij}$ y se pretende asignar a cada fila una columna, de forma que dos filas diferentes no tengan asignada la misma columna (a dos trabajadores diferentes no se les puede asignar la misma tarea) y que la suma de las correspondientes entradas de la matriz se minimice. Este enfoque tiene el inconveniente de que para prohibir determinadas asignaciones es necesario considerar que los coeficientes c_{ij} correspondientes son valores suficientemente grandes como para que en la práctica se impida realizarlas.

También se puede plantear el problema usando un grafo, en este caso no será necesario usar costes grandes para prohibir determinadas asignaciones.

En este caso se usa un grafo bipartito $G = (U, V; E)$ de forma que U tiene tantos vértices como filas tiene la matriz de costes C, es decir, tantos nodos como trabajadores, V tiene tantos vértices como columnas tiene C (tantos vértices como tareas) y E es el conjunto de aristas que van de U a V con un peso c_{ij}. El problema consiste entonces en encontrar un subconjunto de aristas de forma que cada vértice pertenece exactamente a una arista y la suma de los costes o pesos de estas aristas es mínima.

En la práctica se trabajará tanto con el grafo como con la matriz de costes, pretendiendo dar una visión mas completa del problema de Asignación.

Para resolver el problema de Asignación supondremos, sin pérdida de generalidad, que los costes c_{ij} son no negativos, ya que la mayoría de algoritmos de preprocesamiento, de los cuales se hablará más adelante en profundidad, producen una matriz de costes reducidos no negativa. Si no lo fueran bastaría hacer un sencillo cambio que consiste en restar a cada coeficiente el mínimo de los coeficientes de la matriz de coste y el problema resultante sería equivalente.

2.1. Modelo matemático

Por simplicidad, en esta sección vamos a dar la formulación matemática del problema de asignación asumiendo la existencia de una matriz de costes $n \times n$. La formulación basada en un grafo bipartito es similar.

Sea $X = x_{ij}$ una matriz binaria definida de la forma:

$$x_{ij} = \begin{cases} 1 \text{ si la fila } i \text{ está asignada a la columna } j \\ 0 \text{ en otro caso} \end{cases}$$

Formulamos el problema de asignación como:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

sujeto a:

$$\sum_{j=1}^{n} x_{ij} = 1 \ (i = 1,2,...,n)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ (j = 1,2,...,n)$$

$$x_{ij} \in \{0, 1\} \ (i,j = 1,2,...,n).$$

La matriz de las restricciones es una matriz $(2n \times n^2)$. En la formulación sobre un grafo bipartito esta matriz coincidirá con la de incidencia vértice-arco del grafo. Describamos la matriz más exhaustivamente: tiene una fila i por cada vértice $i \in U$, una fila $n + j$ por cada vértice $j \in V$ y una columna por cada arista $[i, j]$. Las entradas de esta matriz son 1 y 0. Además, esta matriz cumple la propiedad de ser totalmente unimodular.

$$\left(\begin{array}{cccc|cccc|c|cccc}
1 & 1 & \ldots & 1 & & & & & & & & \\
 & & & & 1 & 1 & \ldots & 1 & & & & \\
 & & & & & & & & \ldots & & & \\
 & & & & & & & & & 1 & 1 & \ldots & 1 \\
\hline
1 & & & & 1 & & & & & 1 & & & \\
 & 1 & & & & 1 & & & & & 1 & & \\
 & & \ldots & & & & \ldots & & \ldots & & & \ldots & \\
 & & & 1 & & & & 1 & & & & & 1
\end{array}\right)$$

Si no fueran posibles todas las asignaciones, se eliminarían las columnas correspondientes en la matriz anterior.

A continuación daremos un resultado que será útil para comprobar que efectivamente la matriz de restricciones es totalmente unimodular.

Definición 1. *Una matriz entera se dice* ***totalmente unimodular*** *si el determinante de cada submatriz cuadrada es 0, 1 ó -1.*

Condición suficiente: [1] *Una matriz entera A con $a_{ij} = 0, \pm 1$ es totalmente unimodular si hay a lo más 2 elementos no nulos en cada columna y las filas de la matriz pueden dividirse en 2 subconjuntos I, J de forma que:*

1. *Si una columna tiene 2 elementos con el mismo signo, sus filas están en diferentes subconjuntos.*

2. *Si una columna tiene 2 elementos de diferente signo, sus filas están en el mismo subconjunto.*

De esta forma, tomando como subconjunto $I = 1, ..., n$ las primeras n filas, y $J = n + 1, ..., 2n$ vemos que nuestra matriz cumple la condición suficiente. Esta elección de las filas de la matriz es válida, tanto si son posibles todas las asignaciones, como si no.

Esta propiedad es de gran utilidad, ya que un problema de Programación Lineal Entera con matriz totalmente unimodular nos garantiza que **su solución óptima es entera**. [1]

2.2. El problema dual

Podemos escribir el problema dual de (2.1) dado que la propiedad de unimodularidad permite reemplazar la restricción de que las variables son binarias por la restricción de que sean no negativas.

$$max \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

sujeto a :

$$u_i + v_j \leq c_{ij} \ (i,j = 1,2,...,n).$$

Donde u_i y v_j son las variables duales asociadas a las restricciones del problema primal de asignación.

El **teorema de holgura complementaria** dice que un par de soluciones respectivamente factible para el primal y el dual, es óptimo sí y solo sí

$$x_{ij}(c_{ij} - u_i - v_j) = 0 \ (i,j = 1,2,...,n).$$

Llamaremos **costes reducidos** a $\overline{c}_{ij} = c_{ij} - u_i - v_j$.

El teorema de holgura complementaria nos permite encontrar la solución óptima del problema dual cuando se conoce la solución óptima del primal y viceversa. Además, el teorema de dualidad fuerte asegura que, si existe solución óptima, los valores objetivo del problema primal y dual coinciden.

Para cada solución factible primal x, se tiene:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij} - u_i - v_j)x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}x_{ij} - \sum_{i=1}^{n}u_i\sum_{j=1}^{n}x_{ij} - \sum_{j=1}^{n}v_j\sum_{i=1}^{n}x_{ij}$$

Pero atendiendo a las restricciones del problema primal

$$\sum_{i=1}^{n}x_{ij} = 1 \ (i = 1,2,...,n) \ , \ \sum_{j=1}^{n}x_{ij} = 1 \ (j = 1,2,...,n)$$

Por lo que finalmente la función objetivo transformada queda :

$$\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}x_{ij} - \sum_{i=1}^{n}u_i - \sum_{j=1}^{n}v_j$$

Esto es, la diferencia de la función objetivo del primal con la función objetivo del problema dual. Por esta razón, iremos buscando hacer ceros, cuando son evaluadas en dos soluciones óptimas en sus respectivos problemas, en la expresión $(c_{ij} - u_i - v_j)x_{ij}$, es decir, si x_{ij} es 1, entonces $c_{ij} - u_i - v_j$ debe ser 0 y viceversa, de forma que se tenga $(c_{ij} - u_i - v_j)x_{ij} = 0$ ya que de esta manera se obtiene una solución primal y una solución dual con el mismo valor objetivo, por lo que ambas son soluciones óptimas.

Capítulo 3

Historia y desarrollo

En esta sección veremos el desarrollo del método utilizado para resolver el PA [6].

A pesar de la simplicidad de la formulación del PA, en los últimos 50 años este problema atrajo a cientos de investigadores, acompañando y a veces anticipando el desarrollo de la optimización combinatoria.

Los orígenes del problema de asignación datan del siglo XVIII, cuando Monge (1781) formuló el problema de transporte de la masa continua como un enorme problema de asignación que minimiza el coste de transportar todas las moléculas. La estructura combinatoria del problema fue investigada al principio del siglo XX (Miller, Kónig, Frobenius) mientras que el primer algoritmo de enumeración implícita (en tiempo exponencial) fue propuesto en los años 40 por Easterfield (1946).

El problema fue formulado en un contexto moderno en los años 50, curiosamente no por un matemático, si no por un psicólogo. Thorndike (1950), presidente de la división de evaluación y medición de la asociación america-

na de psicología, definió el problema de asignación de la misma forma que un profesor hoy en día se lo explica a sus alumnos. "Sea un conjunto de N tareas y N trabajadores, debemos asignar a cada trabajador una tarea de forma que el beneficio sea máximo".

El nombre del problema, sin embargo, no fue inventado por Thorndike, sino por Votaw y Orden (1952) en un documento titulado: 'El problema de asignación de personal'.

Thorndike explicó que presentó el problema a un matemático, quien observó que hay un número finito de permutaciones en el problema de asignar trabajadores a vacantes, por lo que desde el punto de vista matemático no había problema. Solo había que probarlos todos y elegir el mejor. El matemático concluyó ahí su estudio del problema, pero Thorndike observó:

'Cuando consideremos solo 10 vacantes y 10 hombres, hay alrededor de 3,5 millones de permutaciones. Probar todas las posibles permutaciones y elegir la mejor puede ser una solución matemática pero no es práctica'.

Es divertido que un psicólogo de los años 50 comprendiera mejor la complejidad de un problema que un matemático (los matemáticos entenderían esto 15 años después con Edmonds (1965)).

El primer algoritmo moderno en tiempo polinomial para el problema de asignación, inventado por Harold W. Kuhn hace 50 años, fue bautizado como 'El método Húngaro' para destacar que deriva de resultados anteriores de Kónig y Egerváry obtenidos en los primeros años del siglo XX.

Analizaremos por tanto los resultados de estos matemáticos húngaros para entender el origen del método Húngaro.

3.1. Teorema de Kónig

A continuación se dan algunos conceptos con el fin de situarnos antes de abordar el teorema expuesto por Kónig.

Definición 2. *Sea $G = (U, V; E)$ un grafo bipartito. Un* ***emparejamiento*** *es un subconjunto M de E de forma que cada vértice de G incide en a lo sumo una arista de M. Si $|U| = |V| = n$, un emparejamiento de cardinalidad n es llamado* ***perfecto****.*

Definición 3. *Un* ***cubrimiento*** *de un grafo por vértices es un subconjunto C de $U \cup V$ de forma que cada arista de G incide en al menos un vértice de C.*

Teorema 1. ***(Kónig 1916)****: En un grafo bipartito la máxima cardinalidad de un emparejamiento es igual a la mínima cardinalidad de un cubrimiento por vértices.*

El problema de encontrar un emparejamiento de máxima cardinalidad puede ser modelado como un problema lineal cuya matriz de restricciones es la matriz de adyacencia nodo-arco del grafo bipartito.

El lema probado por Farkas (1902), otro matemático húngaro, fue un primer paso en la dirección de encontrar una solución al problema. El teorema de Egerváry extenderá el resultado de Kónig al caso en que se consideran pesos diferentes para cada asignación, y por tanto desarrolla plenamente el aspecto dual del problema. El aspecto más notable de la contribución de Kónig es la técnica que usa para la demostración.

Para demostrar que hay un emparejamiento M cuya cardinalidad es igual al mínimo cubrimiento por vértices, Kónig dió una prueba constructiva que puede ser resumida en :

1. Sea cualquier emparejamiento que no sea el máximo (puede ser el vacío), hay un algoritmo para producir un nuevo emparejamiento que aumenta en 1 la cardinalidad.

2. Si el algoritmo falla entonces existe un cubrimiento por vértices que tiene la misma cardinalidad que el emparejamiento utilizado.

A pesar de que fue descrito ligeramente por otro camino, el algoritmo del punto 1 es precisamente el algoritmo de caminos alternantes, el cual explicaremos detalladamente en el próximo capítulo. Por otro lado, el punto 2 constituye el resultado de dualidad fuerte que se verifica en programación lineal.

3.2. Teorema de Egerváry

Nos remontamos a los años 50, cuando Kuhn leyó el libro de Kónig. En éste encontró el problema de emparejamiento, un caso especial del problema de asignación que surge cuando la matriz de costes es binaria, y una anotación a pie de página que hacía referencia al documento de Egerváry en húngaro. De acuerdo con Kónig, este documento extiende su resultado al caso con pesos.

Entonces, Kuhn usó un diccionario de Húngaro y escribió en inglés la traducción del documento. El principal resultado probado por Egerváry es

un teorema en el que demuestra que en el problema de asignación el problema dual y el problema primal tienen el mismo valor objetivo. Egerváry no proporcionó un algoritmo explícito para probar esto. Sin embargo describió un método iterativo, que más adelante, Kuhn bautizó como **el Método Húngaro** en honor a Kónig y Egerváry.

Estos dos grandes matemáticos húngaros también compartieron un trágico destino. En 1944, en el periodo de las atrocidades antisemitas, que tuvieron lugar después de la ocupación nazi, Kónig se suicidó. En 1958, en el periodo de la represión comunista que siguió a la revolución del 1956, asustado por varias acusaciones, Egerváry se suicidó.

Se mencionará brevemente un reciente descubrimiento histórico que conecta el algoritmo húngaro con un documento póstumo de Jacobi (1890) escrito en latín.

Jacobi dió la siguiente definición, en la cual el problema de asignación es fácilmente reconocible: 'Sean n^2 números en una tabla cuadrada de forma que hay n series horizontales y n series verticales, cada una teniendo n números. Con estos números queremos seleccionar n transversales, es decir, todos en diferentes series verticales y horizontales, de forma que determinen la máxima suma de los n números seleccionados'.

Jacobi no solo definió el PA, también dió un algoritmo para resolverlo en tiempo polinomial. Ollivier y Sadik (2007) han observado que es esencialmente idéntico al Método Húngaro, anticipándose en varias décadas a los resultados que hemos visto en las secciones anteriores.

3.3. La comunicación por satélite y el problema de asignación

El problema de asignación sigue muy presente en nuestros días en el contexto de las tecnologías de la información y las comunicaciones.

En sistemas de telecomunicaciones basados en satélites, un satélite es usado para transmitir información entre n estaciones terrestres diferentes. El **problema de asignación de intervalos de tiempo** consiste principalmente en transmitir la información deseada en el mínimo tiempo total posible. Veamos cómo se puede plantear este problema como un problema de asignación.

A bordo de cada satélite hay n transpondedores, y las interconexiones entre las estaciones de envío y recepción se obtienen a través de un interruptor $n \times n$. Un conjunto específico de n interconexiones es llamado **permutado**, y puede ser representado por una matriz binaria de permutaciones $P = (p_{ij})$, es decir, una matriz $n \times n$ con exactamente una entrada en cada fila y columna, que contendrá un 1 en la posición (i, j) si se envía información desde la estación i a la estación j y un 0 en otro caso. Su labor es expresar en que estaciones hay flujo de información.

Se define también la matriz de tráfico, una matriz $n \times n$ no negativa T, que especifica para cada (i, j) la cantidad total de tiempo t_{ij} necesario para transmitir la información requerida de la estación i a la estación j.

El problema consiste entonces en encontrar una secuencia de permutaciones y los correspondientes tiempos de transmisión de forma que toda la información sea transmitida en el mínimo tiempo total posible.

Capítulo 4

El método Húngaro

4.1. Algoritmos del método Húngaro

El algoritmo que usaremos para resolver el Problema de Asignación, el método Húngaro, es un algoritmo primal-dual. Comienza con una solución dual factible y una solución primal infactible (parcial), ambas soluciones verificando las condiciones de holguras complementarias, donde menos de n filas son asignadas. Para obtener dichas soluciones, se comenzará describiendo el **algoritmo de preprocesamiento**, una fase inicial en la que se obtiene la matriz de costes reducidos y se determinará una asignación parcial.

4.1.1. Algoritmo de preprocesamiento

Para este algoritmo denotaremos:

$$row(j) = \begin{cases} i \text{ si la columna } j \text{ está asignada a la fila } i \\ 0 \text{ si la columna } j \text{ no está asignada} \end{cases}$$

Del mismo modo,

$$\varphi(i) = \begin{cases} j \text{ si la fila } i \text{ está asignada a la columna } j \\ 0 \text{ si la fila } i \text{ no está asignada} \end{cases}$$

ALGORITMO 1 : Fase de preprocesamiento

Solución dual factible y solucion primal parcial.

for i:=1 to n **for** j:=1 to n do $x_{ij} := 0$;

comentario: reducción de fila y columna;

for i:=1 to n do $u_i := \min\{c_{ij}\} : j = 1, 2, ..., n$;

for j:=1 to n do $v_j := \min\{c_{ij} - u_i\} : i = 1, 2, ..., n$;

comentario: encontrar una solución factible parcial;

for $j := 1$ to n do row(j) := 0;

for $i := 1$ to n do

for $j := 1$ to n do

if (row(j) = 0 and $c_{ij} - u_i - v_j = 0$) then

$x_{ij} := 1$, row(j) := i;

break

endif

endfor

endfor

En el algoritmo anterior, u_i y v_j satisfacen las restricciones duales. Además, los valores x_{ij} obtenidos satisfacen las condiciones de holgura complemen-

taria.

Se denotará G^0 al grafo bipartito que solo contiene los vértices y aristas con coste reducido 0.

Con este algoritmo obtenemos la matriz de costes reducidos y una asignación parcial en el grafo G^0. En todos los algoritmos trabajaremos con este grafo.

Desde el punto de vista matricial, en la matriz de costes reducidos llamaremos **ceros encuadrados**, a los ceros de la matriz que corresponden a las aristas que son asignadas.

La asignación parcial se lleva a cabo de la siguiente forma: Se recorre la matriz de costes reducidos de izquierda a derecha y de arriba a abajo. Si encuentra un cero no tachado, lo encuadra. Una vez encuadra un cero, tacha todos los ceros que estén en la fila y columna de ese cero.
También se podría dar un algoritmo alternativo, que hiciera primero la reducción de las columnas, obteniendo en general una matriz de costes reducidos diferente.

Ilustraremos la explicación del algoritmo con un ejemplo. A medida que avanzamos en el texto seguiremos con este mismo ejemplo para facilitar la comprensión de los algoritmos.

Ejemplo Sea $C = c_{ij}$ la matriz siguiente

$$\begin{pmatrix} 7 & 9 & 8 & 9 \\ 2 & 8 & 5 & 7 \\ 1 & 6 & 6 & 9 \\ 3 & 6 & 2 & 2 \end{pmatrix}$$

Le aplicamos el **algoritmo de preprocesamiento**. Buscamos el mínimo en cada fila y obtenemos:

$u_i = \begin{pmatrix} 7 & 2 & 1 & 2 \end{pmatrix}$

Por lo que obtenemos la matriz:

$$\begin{pmatrix} 0 & 2 & 1 & 2 \\ 0 & 6 & 3 & 5 \\ 0 & 5 & 5 & 8 \\ 1 & 4 & 0 & 0 \end{pmatrix}$$

A continuación denotamos los valores $v_j = \begin{pmatrix} 0 & 2 & 0 & 0 \end{pmatrix}$ quedando finalmente la matriz $\overline{C}$ de costes reducidos:

$$\begin{pmatrix} 0 & 0 & 1 & 2 \\ 0 & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & 0 & 0 \end{pmatrix}$$

Siguiendo el algoritmo, debemos obtener una asignación parcial que es la siguiente:

$$\begin{pmatrix} \boxed{0} & 0 & 1 & 2 \\ 0 & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & \boxed{0} & 0 \end{pmatrix}$$

Con lo que se obtiene: $row = \begin{pmatrix} 1 & 0 & 4 & 0 \end{pmatrix}$ y $\varphi = \begin{pmatrix} 1 & 0 & 0 & 3 \end{pmatrix}$

Vemos que han sido asignadas menos de n (en nuestro ejemplo $n = 4$) filas.

4.1.2. Algoritmo de caminos alternantes

Una vez obtenida la solución parcial dada por el algoritmo de preprocesamiento, el algoritmo Húngaro resuelve en cada iteración un problema primal restringido, operando en el grafo parcial G^0. El objetivo es conseguir aumentar la cardinalidad de la asignación actual.

El algoritmo húngaro incluye un algoritmo auxiliar llamado **Algoritmo de caminos_alternantes** que explicaremos detalladamente.

El objetivo de éste es, una vez se tiene la asignación parcial, dar la mejor manera de asignar aristas o encuadrar ceros en la matriz de costes reducidos para que haya el máximo número de ceros posible.

De esta forma, y con el fin de exponer el algoritmo de Caminos_Alternantes, daremos las siguientes definiciones:

Definición 4. *Se dirá que un eje o arista de i a j está **asignado** si pertenece al grafo G^0 y hemos encuadrado en la matriz de costes reducidos el elemento c_{ij}.*

Definición 5. *Llamaremos **camino alternante** a un camino no orientado en el que la secuencia de aristas que lo forman están asignadas y no asignadas de forma alterna.*

El algoritmo de **Caminos_Alternantes** busca una posible **trayectoria de aumento** a partir de un vértice k no asignado. Encuentra un camino

alternante que empieza en k y con un número impar de aristas, de forma que todas las aristas en posición impar del camino no están asignadas y las que se encuentran en posición par sí lo están. Intercambiando las aristas asignadas y las no asignadas se consigue aumentar el número de asignaciones en 1 manteniendo la condición de holgura complementaria.

En cualquier iteración un vértice es **etiquetado** si pertenece a un posible camino desde k. Se dirá que un vértice es **escaneado** si es utilizado para el camino alternante construido.

Llamaremos LV a los vértices etiquetados de V, SU a los vértices de U escaneados y SV a los vértices pertenecientes a V escaneados.

ALGORITMO 2 : Caminos_Alternantes(k)

Encontrar un camino alternante desde un vértice no asignado $k \in U$

SU:= LV:= SV:= $\emptyset$;

fail:=`false`, *sink*:=0, $i := k$;

while (*fail* = `false` **and** *sink* = 0) **do**

SU := SU $\cup$ $\{i\}$;

for each j $\in$ V $\setminus$ LV : $c_{ij} - u_i - v_j = 0$ **do** $pred_j$:= i, LV := LV $\cup$ {j} ;

if LV $\setminus$ SV = $\emptyset$ **then** *fail* := `true`

else

let j be any vertex in LV $\setminus$ SV ;

SV := SV $\cup$ $\{j\}$;

if row(j) = 0 **then** *sink* := j **else** i:= row(j)

endif

endwhile

return *sink*

Cada iteración consiste en dos fases:
En el grafo G^0, se parte de un vértice no asignado $k \in$ U, lo etiquetamos y lo escaneamos. A continuación se etiquetan todos los vértices j $\in$ V no escaneados (no hemos pasado aún por ellos).

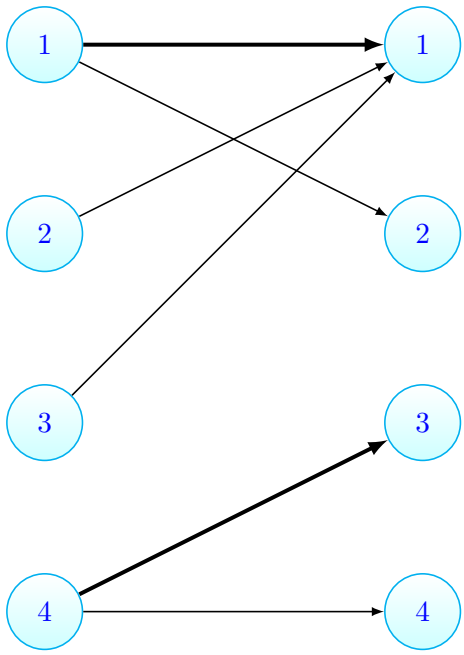
En la segunda fase, un vértice no escaneado de V es seleccionado para continuar el camino, por tanto es escaneado. El vértice $row(j) \in$ U entonces se convierte en el candidato para la siguiente iteración.

En cada iteración se añade un vértice $j \in LV \backslash SV$ al conjunto de vértices SV, y consideramos un vértice diferente, $i = row(j)$. El proceso puede terminar en esta fase de dos maneras diferentes:

1. Si el vértice j $\in$ V que seleccionamos no está asignado, se habrá obtenido una trayectoria de aumento de k a j

2. Si V no contiene vértices etiquetados y no escaneados, es decir, si todos los vértices posibles para aumentar el camino ya han sido utilizados en el recorrido, entonces no podremos seguir aumentando el camino.

Ejemplo : Continuamos con el ejemplo anterior. Con los ceros de la matriz de costes reducidos $\overline{C}$ se determinan las aristas en el grafo bipartito G^0. Veamos como se encuentra el grafo con la asignación parcial dada por el algoritmo de preprocesamiento:

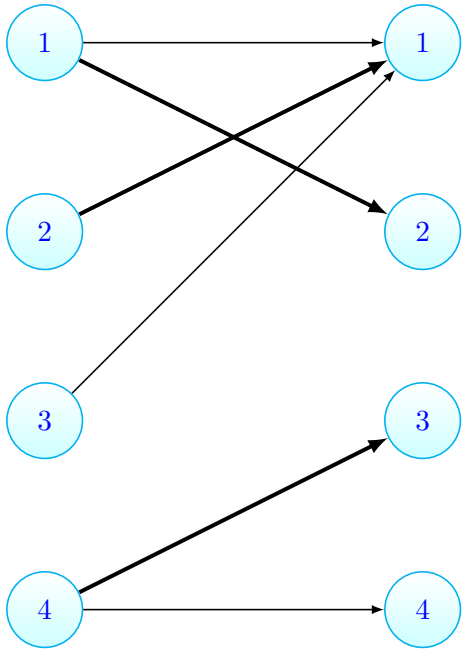


donde las aristas que se han resaltado con un trazo más grueso muestran la asignación parcial actual.

Llamamos a **Caminos_Alternantes(2)** ya que 2 es un vértice no asignado:

- $SU = LV = SV = \varnothing, fail =$ `false`$, sink = 0$;
- $i = 2 : SU = \{2\}, pred_1 = 2, LV = \{1\}, row(1) = 1; j = 1 : SV = \{1\}$
- $i = 1 : SU = \{2, 1\}, pred_2 = 1, LV = \{1, 2\}; j = 2 : SV = \{1, 2\}, sink = 2$.

Hemos acabado y se ha encontrado una trayectoria de aumento que asigna el vértice 2. El grafo resultante después de llamar a Caminos_Alternantes(2) es el siguiente:



Y en la matriz de costes reducidos se obtienen los siguientes ceros encuadrados:

$$\begin{pmatrix} 0 & \boxed{0} & 1 & 2 \\ \boxed{0} & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & \boxed{0} & 0 \end{pmatrix}$$

También debemos llamar a **Caminos_Alternantes(3)**, pero en este caso no conseguirá una trayectoria de aumento ya que no hay ninguna arista $a \in SV \setminus LV$. Por tanto, el proceso termina sin asignar 3.

4.1.3. Algoritmo Húngaro

Una vez descritos el algoritmo inicial y el auxiliar, se procede a detallar el algoritmo principal del método Húngaro.

En el **algoritmo Húngaro** la asignación en cada etapa se almacena en las matrices row y φ. Las matrices u, v, row y φ se inicializan mediante el algoritmo de preprocesamiento que explicamos anteriormente.
Denotaremos por $U^0 \subseteq U$ el conjunto de vértices de U que están asignados.

ALGORITMO 3 : Húngaro

Inicializar u, v, row, φ y U^0 ;

while $|U^0| < n$ **do**

Sea k cualquier vértice en $U \setminus U^0$;

while $k \notin U^0$ **do**

$sink :=$ Caminos alternantes (k);

if $sink > 0$ **then** [**comentario**: aumentar la solución primal]

$U^0 := U^0 \cup \{k\}$, $:= sink$;

repeat

$i := pred_j$, $row(j) := i$, $h := \varphi(i)$, $\varphi(i) := j$, $j := h$

until $i = k$

else[**comentario:** actualizar la solución dual]

$\delta :=\min\{c_{ij} - u_i - v_j\} : i \in SU$, $j \in V \setminus LV$;

for each $i \in SU$ **do** $u_i := u_i + \delta$

for each $j \in LV$ **do** $v_j := v_j + \delta$

endif

endwhile

endwhile

Analicemos en profundidad el algoritmo:

Comienza seleccionando un vértice k de U que no está asignado. Llama a **Caminos_Alternantes(k)** para intentar obtener una trayectoria de aumento. Se hace esto con todos los vértices no asignados. Si consigue llegar a una asignación de todos los vértices, habremos acabado, puesto que tendremos una asignación óptima (por la descripción del algoritmo de Caminos_Alternantes, que busca la manera óptima de encuadrar ceros en la matriz de costes reducidos, es decir, la manera óptima de asignar vértices de U con vértices de V). Si no es éste el caso, después de llamar a **Caminos_Alternantes** de todos los vértices no asignados, describimos los conjuntos SU, LV y SV de dichos vértices. Una vez hecho esto, determinamos en la matriz de costes reducidos las submatrices LV, que serán las columnas de la matriz que se correspondan con los vértices $v \in LV$, y $\overline{SU}$ que es el conjunto de filas de la matriz que se corresponden con los vértices $u \notin SU$.

Veamos una proposición que nos demuestra que, si consideramos la matriz de costes reducidos, las filas que no pertenecen a SU o las columnas contenidas en LV contienen a todos los ceros de la matriz.

Proposición 1. *Al aplicar el método Húngaro, si se procede a adaptar la solución dual, todos los costes reducidos nulos* $\overline{c_{ij}} = c_{ij} - u_i - v_j = 0$ *verifican* $i \notin SU$ *o* $j \in LV$

Demostración : Por reducción al absurdo, supongamos que $\overline{c_{ij}} = 0$ y $j \notin LV$.

Debemos demostrar que $i \notin SU$:

Si $i \in SU$ pueden ocurrir dos casos:

1. O bien $i = k$ siendo $k \in U \backslash U^0$ con el cual se llama al procedimiento **Caminos_Alternantes(k)**

2. O bien $i = row(l) | l \in SV$

(1) Si $i = k$ como $\overline{c_{ij}} = 0 \Rightarrow j \in LV$ $(\rightarrow\leftarrow)$

(2)Si $i = row(l) \mid l \in SV \Rightarrow \overline{c_{il}} = 0$ y $\varphi(i) = l$, como $\overline{c_{ij}} = 0$, j se añadiría a LV en la siguiente iteración del procedimiento Caminos_Alternantes(k).$(\rightarrow\leftarrow$)

Luego queda demostrado que en el conjunto de filas y columnas $\overline{SU} \cup LV$ se encuentran todos los ceros de la matriz de costes reducidos, tanto los encuadrados como los que no lo están. Por tanto, en la submatriz complementaria, estos es, $SU \cap \overline{LV}$, todos los términos son estrictamente positivos. Esto garantiza que el método Húngaro cambia de solución dual al llegar al paso de actualización, lo cual es muy importante pues de lo contrario el algoritmo quedaría estancado en las soluciones actuales.
El **Algoritmo Húngaro** modifica la solución dual realizando la siguiente transformación:

1. En $LV \cap SU$ sumamos δ por pertenecer a LV y restamos δ por pertenecer a SU, por lo que no se produce cambio: $\overline{c_{ij}} = \overline{c_{ij}} - \delta + \delta$.

2. En $LV \cap \overline{SU}$ sumamos a los nuevos costes reducidos, δ por pertenecer a LV, pero no restamos porque no pertenecen a SU: $\overline{c_{ij}} = \overline{c_{ij}} + \delta$.

3. En $\overline{LV} \cap SU$ restamos δ por pertenecer a SU, y al ser el mínimo todos los valores resultantes son ≥ 0. Además, aseguramos que se modifica la solución dual, ya que al menos uno de los nuevos costes reducidos $\overline{c_{ij}}$ con $i \in SU \wedge j \in \overline{LV}$ será cero: $\overline{c_{ij}} = \overline{c_{ij}} - \delta$.

4. En $\overline{SU} \cap \overline{LV}$, de nuevo, no se producen cambios, ya que ni sumamos ni restamos δ al no encontrarnos en ninguno de los conjuntos que producen la variación: $\overline{c_{ij}} = \overline{c_{ij}}$

Ejemplo: Continuamos con el ejemplo, partiendo del grafo obtenido después de aplicar **Caminos_alternantes(k)** para todos los k no asignados.

Como el vértice 3 es el único que no está asignado, obtenemos los conjuntos: $SU = \{2, 3\}, LV = \{1\}$ y las matrices: $row = \begin{pmatrix} 2 & 1 & 4 & 0 \end{pmatrix}$, $\varphi = \begin{pmatrix} 2 & 1 & 0 & 3 \end{pmatrix}$

A continuación determinamos en la matriz $\overline{C}$ los conjuntos hallados anteriormente:

$$\begin{array}{c} \\ \overline{SU} \\ \text{SU} \\ \text{SU} \\ \overline{SU} \end{array} \begin{array}{c} \begin{array}{cccc} \text{LV} & \overline{LV} & \overline{LV} & \overline{LV} \end{array} \\ \left(\begin{array}{c|ccc} 0 & 0 & 1 & 2 \\ \hline 0 & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ \hline 1 & 2 & 0 & 0 \end{array}\right) \end{array}$$

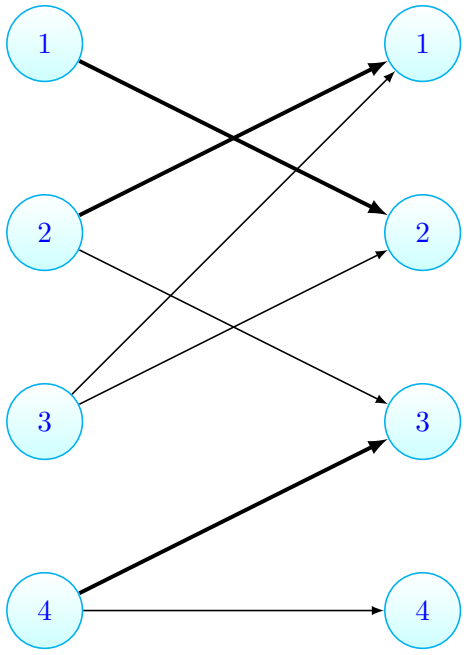
A los elementos pertenecientes a $SU \cap \overline{LV}$ les restamos δ, y a los elementos pertenecientes a $LV \cap \overline{SU}$ les sumamos δ, con lo que nos queda la matriz de nuevos costes reducidos:

$$\begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 5 \\ 4 & 2 & 0 & 0 \end{pmatrix}$$

Determinamos una nueva asignación parcial, ya que el algoritmo Húngaro nos asegura que hay al menos un nuevo 0:

$$\begin{pmatrix} 3 & \boxed{0} & 1 & 2 \\ \boxed{0} & 1 & 0 & 2 \\ 0 & 0 & 2 & 5 \\ 4 & 2 & \boxed{0} & 0 \end{pmatrix}$$

Con esta asignación, el grafo que se obtiene es el siguiente:

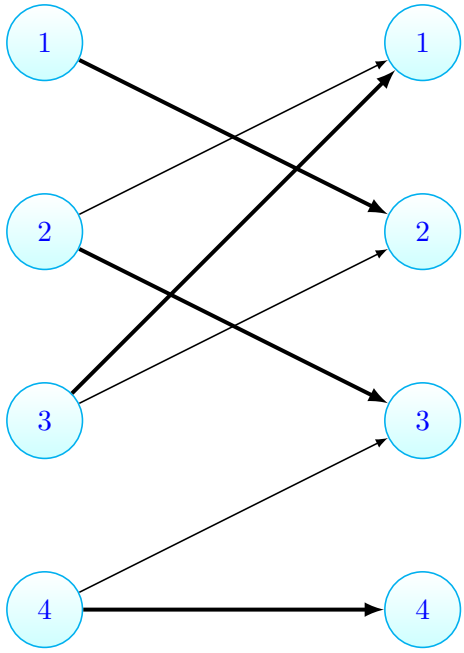


Como el vértice 3 no está asignado, llamamos a **Caminos_Alternantes(3)**:

- $SU = \{3\}$, $LV = \{1, 2\}$, $SV = \{1\}$, $row(1) = 2$
- $SU = \{2, 3\}$, $LV = \{1, 2, 3\}$, $SV = \{1\}$
- $SU = \{2, 3\}$, $LV = \{1, 2, 3\}$, $SV = \{1, 3\}$, $row(3) = 4$
- $SU = \{2, 3, 4\}$, $LV = \{1, 2, 3, 4\}$, $SV = \{1, 3\}$
- $SU = \{2, 3, 4\}$, $LV = \{1, 2, 3, 4\}$, $SV = \{1, 3, 4\}$.

Hemos descrito el proceso del algoritmo que encuentra una trayectoria de aumento, pero realmente **Caminos_Alternantes** busca todos los posibles caminos. Faltaría ver a donde llegamos si en la primera iteración escaneamos el vértice 2 en vez de 1. Pero es fácil ver que por este camino no encontramos una trayectoria de aumento, ya que nos quedamos sin vértices $v \in LV \backslash SV$.

Se ha encontrado una trayectoria de aumento, por lo que nuestro grafo queda:



Y la nueva matriz de costes reducidos queda de la siguiente forma:

$$\begin{pmatrix} 3 & \boxed{0} & 1 & 2 \\ 0 & 1 & \boxed{0} & 2 \\ \boxed{0} & 0 & 2 & 5 \\ 4 & 2 & 0 & \boxed{0} \end{pmatrix}$$

Se ha encontrado una manera de encuadrar n ceros que es la óptima, por la construcción de los algoritmos. Se ha resuelto el problema de asignación para este ejemplo.

4.1.4. Método húngaro con marcas

En el caso de problemas muy pequeños, el método Húngaro con marcas es un recurso didáctico que nos muestra cómo resolver el problema sin ayuda

de un ordenador, 'a mano'.

Dada la matriz de costes reducidos, se parte de una asignación parcial del problema, en la que se han **encuadrado** los ceros que corresponden a las asignaciones y se han **tachado** los ceros que están en filas o columnas correspondientes a un cero encuadrado.

El procedimiento a seguir es el siguiente:

1. Marcar las filas sin cero encuadrado.
2. Marcar las columnas con cero tachado en fila marcada.
3. Marcar las filas con cero encuadrado en columna marcada y proceder con los pasos 2 y 3 hasta que no se produzcan nuevas marcas.
4. Tomar filas no marcadas y columnas marcadas.
5. Tomar el mínimo de los elementos que no están en el conjunto seleccionado en el paso anterior, sea δ.
6. Restar δ a los elementos que no están en las líneas seleccionadas en 4, y sumar δ a los que están en el cruce de filas y columnas seleccionadas en 4.

Con este procedimiento de marcas, los índices de filas marcadas corresponden con el conjunto SU descrito en el algoritmo Húngaro. Las columnas marcadas coinciden con el conjunto de índices LV.

Ejemplo:

Continuamos con el mismo ejemplo, pero en lugar de detallar los conjuntos SU y LV mediante el algoritmo Húngaro, lo haremos con el sistema de

marcas. Partimos de la asignación parcial, después de aplicar el algoritmo de caminos alternantes:

$$\begin{pmatrix} 0 & \boxed{0} & 1 & 2 \\ \boxed{0} & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & \boxed{0} & 0 \end{pmatrix}$$

Aplicando los pasos del procedimiento de marcas obtenemos:

$$\begin{array}{cc} & (2) \\ & \left(\begin{array}{c|ccc} 0 & 0 & 1 & 2 \\ \hline 0 & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ \hline 1 & 2 & 0 & 0 \end{array}\right) \\ (3) & \\ (1) & \end{array}$$

Vemos que el resultado es el mismo, obtenemos la misma matriz que aplicando el algoritmo Húngaro.

Por último, en la siguiente sección se analizará la complejidad del método Húngaro.

4.2. Complejidad del método Húngaro

El algoritmo de preprocesamiento que utilizamos para hallar los costes reducidos y una asignación parcial requiere $O(n^2)$. Cada ejecución de Caminos_Alternantes(k) encuentra una trayectoria de aumento en un tiempo $O(n^2)$. Además, en el algoritmo Húngaro, el bucle exterior es ejecutado

en un tiempo $O(n)$ y el bucle interior, donde se encuentran la actualización de la solución dual y Caminos_Alternantes(k), también tiene complejidad $O(n)$. El valor de δ es también computado en tiempo $O(n^2)$.

En total, el tiempo de complejidad del algoritmo húngaro es sobre $O(n^4)$. Hay una variante del algoritmo húngaro que consigue rebajar el tiempo de complejidad del algoritmo a $O(n^3)$ [3].

Capítulo 5

Aplicación al problema del circuito del viajante

El **Problema del circuito del viajante** (TSP por sus siglas en inglés), responde a la siguiente pregunta: Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y regresa a la ciudad origen?

Se abordará el problema como un grafo con tantos vértices como ciudades, y aristas que conectan los vértices, siendo estas los caminos de una ciudad a otra. Además, las aristas tendrán un peso o coste c_{ij} que corresponde con el coste de viajar de la ciudad i a la ciudad j.

En esta sección se formulará el TSP como un problema de Progamación Lineal Entera y se estudiará la relación que existe con el PA. Para ello supondremos en un primer caso que de cada ciudad hay caminos posibles a todas las demás ciudades. Si no fuera así, bastaría con prohibir en la matriz

de costes ese camino, como vimos en el caso del problema de asignación.

5.1. Modelo matemático

Se definen las variables :

$$x_{ij} = \begin{cases} 1 \text{ si en el circuito se va directamente desde la ciudad } i \text{ a la ciudad } j \\ 0 \text{ en otro caso} \end{cases}$$

Sean u_i variables artificiales con $i = 1, \ldots, n$, y sea c_{ij} el coste de ir de la ciudad i a la ciudad j.

Supondremos que la ciudad 1 es desde donde se comienza el circuito y donde se finaliza.

Se formula el TSP como [7]:

$$min \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} c_{ij} x_{ij}$$

sujeto a:

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1, j = 1, \ldots, n$$

$$\sum_{j=1, i \neq j}^{n} x_{ij} = 1, i = 1, \ldots, n$$

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

$$1 \leq u_i \leq n - 1, \text{ con } i = 2, \ldots, n$$

$$u_i - u_j + (n-1)x_{ij} \leq n-2, \text{ con } 2 \leq i \neq j \leq n$$

El primer conjunto de igualdades asegura que de cada ciudad $i \in \{1, ..., n\}$ se va a exactamente una ciudad, y el segundo conjunto de igualdades asegura que a cada ciudad se llega una sola vez. La última restricción obliga a que un solo circuito cubra todas las ciudades y no dos o más circuitos disjuntos cubran conjuntamente todas las ciudades.

Proposición 2. *La formulación anterior modela el problema del circuito del viajante.*

Demostración: Por doble inclusión:

$\Rightarrow$ Vamos a coger una solución factible (que cumple la formulación) y demostraremos que es solución del TSP. Para ello basta ver que si tuviese un subcircuito, tendría que pasar por el nodo 1.

Por reducción al absurdo, supongamos que existe un subcircuito $i_1, i_2, ..., i_r$, con $1 \notin \{i_1, i_2, ..., i_r\}$ Escribamos la última restricción para cada arista de ese subcircuito:

$$x_{i_1 i_2} = 1 \rightarrow u_{i_1} - u_{i_2} + (n-1)x_{i_1 i_2} \leq n-2$$

$$x_{i_2 i_3} = 1 \rightarrow u_{i_2} - u_{i_3} + (n-1)x_{i_2 i_3} \leq n-2$$

$$\vdots$$

$$x_{i_{r-1} i_r} = 1 \rightarrow u_{i_{r-1}} - u_{i_r} + (n-1)x_{i_{r-1} i_r} \leq n-2$$

$$x_{i_r i_1} = 1 \rightarrow u_{i_r} - u_{i_1} + (n-1)x_{i_r i_1} \leq n-2$$

Sumando todas las restricciones, vemos que los términos en u_i se cancelan y todos los $x_{i(i+1)}$ son 1, quedando:

$$r(n-1) \leq r(n-2) \qquad (\rightarrow\leftarrow)$$

Por tanto, hemos demostrado que $1 \in \{i_1, i_2, ..., i_r\}$. Además se tiene, por la primera restricción del problema, que solo hay un circuito que pasa por el nodo 1, luego queda demostrado que no hay subcircuitos. Si es una solución factible, es solución del TSP.

$\Leftarrow$

Veamos la otra implicación. Suponemos que existe un circuito , veamos que cumple nuestra formulación. En particular, debemos ver que cumple la última restricción de desigualdad. Tenemos dos casos posibles:

- Si la variable $x_{ij} = 0$: $u_i \leq n-1$ siempre se tiene. Además, cada $u_j \geq 1$, o, equivalentemente $-u_j \leq -1$. Por tanto, $u_i - u_j \leq n-1-1$, es decir, $u_i - u_j \leq n-2$. Luego se cumple la restricción.

- Si la variable $x_{ij} = 1$ Se tiene que nos movemos de i a j,tomemos u_i igual a la posición que ocupa la ciudad i dentro del orden en el que se visitan las ciudades en el circuito, $u_i = t \Rightarrow u_j = t+1$ para cualquier $t \in 1, ..., n-2$. Por tanto $u_i - u_j = -1$ y en consecuencia se tiene: $u_i - u_j + (n-1) \leq n-2$.

Luego hemos probado que, dado un circuito, cumple nuestra formulación.

5.2. Resolución del TSP

A continuación se describe un algoritmo para, a partir del problema de asignación expuesto en capítulos anteriores, resolver el TSP.

- Se formula el problema del circuito del viajante como un problema de asignación y se intenta resolver con el método Húngaro.

    1. Si da una solución sin subcircuitos, hemos acabado. Se ha resuelto el TSP como un problema de Asignación.
    2. En otro caso, se obtienen al menos dos subcircuitos. Se pretende romper uno de ellos, y éste será el que no contiene al vértice 1. Se añaden cortes que prohiben ese subcircuito, y se resuelve el problema de Programación Lineal Entera resultante.

- Este proceso termina, ya que en cada iteración se tiene una región factible más pequeña, siendo el número de soluciones factibles finito.

Observación: El problema que se resuelve después de cada iteración es un problema de programación entera.

Observación: Analicemos dos posibilidades de hacer los cortes que prohiben los subcircuitos:

1. Introducir una restricción del tipo $u_i - u_j + (n-1)x_{ij} \leq n-2$ por cada arco del subcircuito que se quiere romper, y las variables $1 \leq u_i \leq n-1$ correspondientes.

2. Dado el subcircuito $i_1, i_2, ..., i_r$, añadir una única restricción: $x_{12} + x_{23} + ... + x_{(r-1)r} \leq r - 1$.

5.2.1. Implementación CPLEX del método de cortes

En esta sección describimos una implementación en IBM ILOG CPLEX Optimization Studio project, Version: 12.5.1.0 del método anterior de cortes. A modo de ejemplo lo aplicaremos al problema representado en la figura 5.1

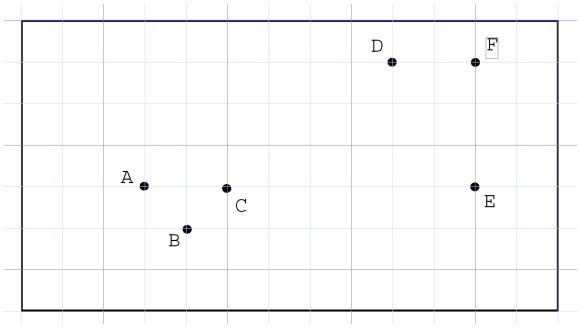


Figura 5.1: Ejemplo para la aplicación del método de cortes

Cuyos datos de entrada en el módulo CPLEX son:

```
n=5;
ncuts=10;
newcut = <0,[0 0 0 0 0 0]>;
rango={0 1 2 3 4 5};
cities = [
<A,[3,3]>
<B,[4,2]>
<C,[5,3]>
<D,[9,6]>
<E,[11,3]>
<F,[11,9]>
];
cycles = {};
```

Aquí, la lista "cycles" irá almacenando los cortes que se irán introduciendo de forma dinámica.

El código CPLEX que implementa el algoritmo es el siguiente:

```
tuple City{
   key string name;
   float coordinate[1..2];
 };
 int n=...;
 int ncuts=...; // máximo número de cortes permitido
 {int} rango=...;
 City cities[0..n]=...;
 tuple Cycle {int r; int pos[0..n];};
 {Cycle} cycles = ...;
 Cycle newcut = ...;
 float dist[0..n][0..n];
 execute distances{
   for(var i in rango)
    for(var j in rango)
     dist[i][j]=Opl.sqrt(Opl.pow(cities[i].coordinate[1]-cities[j].coordinate[1],2)
                 +Opl.pow(cities[i].coordinate[2]-cities[j].coordinate[2],2));}
 dvar boolean x[0..n][0..n];
 minimize
  sum( i, j in 0..n)
    dist[i][j]*x[i][j];
   subject to
   {
    forall( j in 0..n )
     ctllegar:
      sum( i in 0..n: i!=j )
        x[i][j] == 1;
    forall( i in 0..n )
     ctsalir:
      sum( j in 0..n: j!=i )
        x[i][j] == 1;
     forall(c in cycles)
       cuts:
         sum( i,j in 0..n: (c.pos[j]==(c.pos[i]+1) || c.pos[j]==(c.pos[i]+c.r-1))
                           && c.pos[i]!=0 && c.pos[j]!=0)
(x[i][j]+x[j][i]) <=(c.r-1);
```

```
 }
main {
var tsp = thisOplModel;
var def = tsp.modelDefinition;
var data = tsp.dataElements;
var cut=0;
while(cut<=data.ncuts)
{
var cplex1 = new IloCplex();
          // crea objeto de tipo cplex que permite resolver el problema
 // y obtener información sobre valor objetivo, solución, tiempos...
 // para conocer métodos y propiedades ligadas a este objeto
 // escribir cplex. y esperar a que aparezca ventana informativa
tsp = new IloOplModel(def,cplex1);
tsp.addDataSource(data);
tsp.generate();
var time1=cplex1.getDetTime(); // vamos a computar el tiempo de ejecución
cplex1.solve();
var time2=cplex1.getDetTime();
cplex1.exportModel("tspCut.lp");
var x=tsp.x.solutionValue;
writeln('solución actual=');
writeln('       ',x);
writeln('objetivo actual=',cplex1.getObjValue());
writeln('tiempo de ejecución ',time2-time1,' sec');
// declaración de arrays
var m=0;
var i=0;
for (var j in data.rango)
 data.newcut.pos[j]=0;
 // Vamos a construir el ciclo que contiene a la ciudad 0 en la
 // ruta actual. La variable m indica el tramo en el que nos
 // encontramos
var subtour=0;
data.newcut.r=0;
while ((m<data.n)&&(subtour==0))
for (var j in data.rango)
```

```
   if ((x[i][j]==1)&&(subtour==0))
     if (data.newcut.pos[j]!=0)
       subtour=1;// se ha producido un ciclo parcial
     else
      {
   data.newcut.pos[j]=m+1; // no se produce de momento ciclo parcial,
   m++;      // visitamos la ciudad j en el m-ésimo tramo
   data.newcut.r++;
   i=j;      // continuamos construyendo la ruta
   break;
    };
  if (subtour==1) // añadimos un corte a la formulación
   {
   data.cycles.add(data.newcut.r, data.newcut.pos);
   writeln(data.cycles);
   writeln('El último ciclo parcial añadido es', Opl.item(data.cycles,cut));
   cut++;
    tsp.end();
    cplex1.end();
   }
   else
   {
     writeln('Encontrada la solución óptima con ',cut,' cortes,
                            siendo el máximo permitido ', data.ncuts);
   break;
    }
} // cierra el while(cut<data.ncuts)
```

Cuando aplicamos el algoritmo anterior al problema de la figura 5.1 se resuelve el problema de asignación y se forman dos ciclos tal y como aparecen en la figura 5.2.

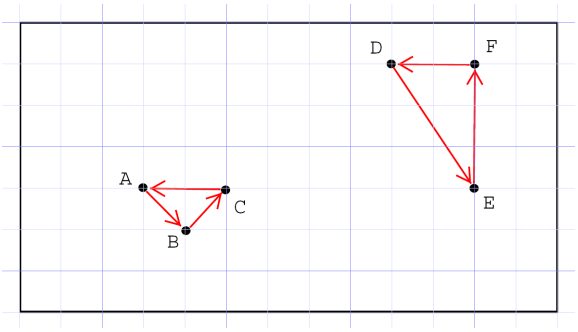


Figura 5.2: Solución inicial al resolver el problema de asignación

Tras la inclusión de varios cortes se obtiene el circuito que aparece en la figura 5.3.

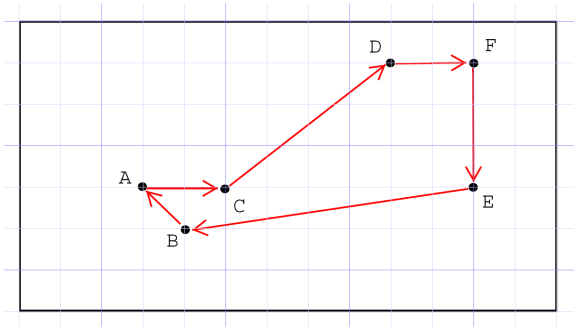


Figura 5.3: Solución final tras introducir 4 cortes

La salida de la aplicación del algoritmo de cortes muestra los cortes añadidos hasta obtener la solución final. Como se puede observar, el valor objetivo óptimo de los problemas relajados va creciendo a medida que introducimos los cortes hasta obtener la solución óptima.

```
solución actual=
        [[0 1 0 0 0 0]
         [0 0 1 0 0 0]
```

```
         [1 0 0 0 0 0]
         [0 0 0 0 1 0]
         [0 0 0 0 0 1]
         [0 0 0 1 0 0]]
objetivo actual=18.03952967567417
tiempo de ejecución 0.2359285354614258 sec
 {<3 [3 1 2 0 0 0]>}
El último ciclo parcial añadido es <3 [3 1 2 0 0 0]>
solución actual=
        [[0 1 0 0 0 0]
         [1 0 0 0 0 0]
         [0 0 0 0 1 0]
         [0 0 0 0 0 1]
         [0 0 1 0 0 0]
         [0 0 0 1 0 0]]
objetivo actual=22.03952967567417
tiempo de ejecución 0.2748842239379883 sec
 {<3 [3 1 2 0 0 0]> <2 [2 1 0 0 0 0]>}
El último ciclo parcial añadido es <2 [2 1 0 0 0 0]>
solución actual=
        [[0 1 0 0 0 0]
         [0 0 0 0 1 0]
         [1 0 0 0 0 0]
         [0 0 0 0 0 1]
         [0 0 1 0 0 0]
         [0 0 0 1 0 0]]
objetivo actual=23.69638392516655
tiempo de ejecución 0.2814064025878906 sec
 {<3 [3 1 2 0 0 0]> <2 [2 1 0 0 0 0]>
     <4 [4 1 3 0 2 0]>}
El último ciclo parcial añadido es <4 [4 1 3 0 2 0]>
solución actual=
        [[0 0 0 0 1 0]
         [1 0 0 0 0 0]
         [0 1 0 0 0 0]
         [0 0 0 0 0 1]
         [0 0 1 0 0 0]
```

```
         [0 0 0 1 0 0]]
objetivo actual=24.03952967567417
tiempo de ejecución 0.5718231201171875 sec
 {<3 [3 1 2 0 0 0]> <2 [2 1 0 0 0 0]>
     <4 [4 1 3 0 2 0]>
     <4 [4 3 2 0 1 0]>}
El último ciclo parcial añadido es <4 [4 3 2 0 1 0]>
solución actual=
        [[0 0 1 0 0 0]
         [1 0 0 0 0 0]
         [0 0 0 1 0 0]
         [0 0 0 0 0 1]
         [0 1 0 0 0 0]
         [0 0 0 0 1 0]]
objetivo actual=25.09083264970256
tiempo de ejecución 0.6963605880737305 sec
Encontrada la solución óptima con 4 cortes, siendo el máximo permitido 10
```

5.3. Otras aplicaciones

Veamos como aplicar el Problema del Circuito del Viajante a otros problemas en diferentes áreas [4].

5.3.1. Visualización de datos

En este apartado se pretende, a partir de una matriz de contingencia, conseguir una reordenación, tanto de sus filas como de sus columnas, de forma que su representación gráfica se pueda analizar de una forma más eficaz con la ayuda del problema del Circuito del Viajante.

Bertin en 1999 [2] introdujo matrices de permutación para analizar datos multivariantes con un tamaño de muestra medio o pequeño.

Para la reordenación de los datos se definen:

1. D_r como la distancia euclídea entre dos filas
2. D_c como la distancia euclídea entre dos columnas
3. $X_p = \pi(X)$ son las posibles permutaciones de filas y columnas
4. ϕ como la suma de las distancias de filas o columnas adyacentes en una permutación

Se busca un π^* que minimice $\phi(\pi)$, es decir, formulamos el problema de optimización siguiente:

$$\phi(\pi^*) := \min \phi(\pi)$$

$$\text{Sujeto a: } \pi \in X_p$$

Resolver este problema es equivalente a resolver dos TSP independientes: Primero resolvemos el problema que determina las distancias mínimas entre filas adyacentes, y seguidamente entre columnas.

A continuación vemos un conjunto de datos antes y después de modificarlo para mejorar su visualización basándonos en el TSP:

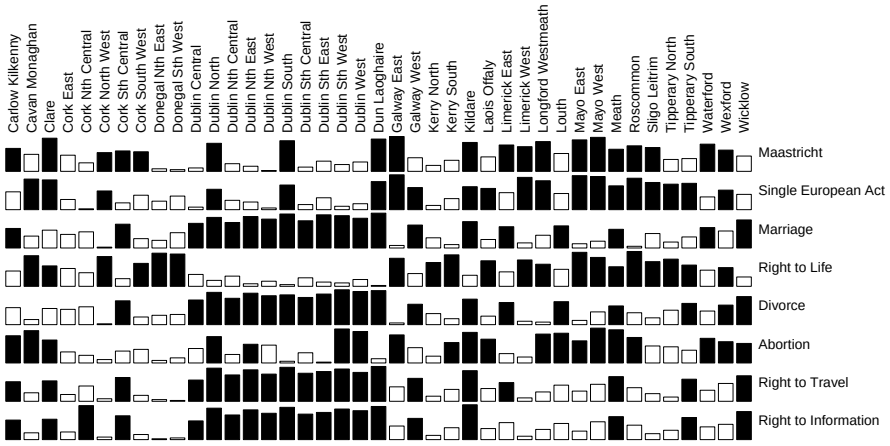


Figura 5.4: Vemos unos datos sobre diversos temas problemáticos en varias ciudades que son difíciles de analizar y no es fácil sacar alguna conclusión válida de este gráfico

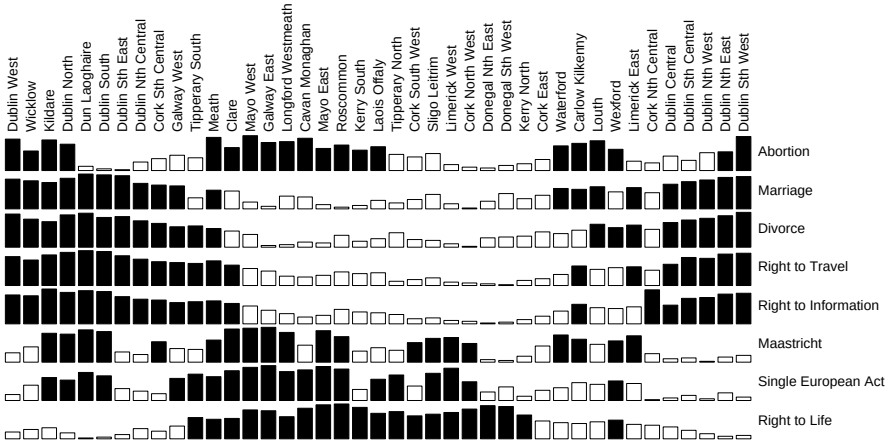


Figura 5.5: Se muestra la misma gráfica pero con una reordenación de los datos, y vemos que mejora en gran medida su interpretabilidad

5.3.2. Problema de placa de circuitos impresos PCB

Es una de las utilidades más ingeniosas que puede resolverse mediante el problema del circuito del viajante: la creación de placas de circuitos impresos [5].

Este problema se enfoca en dos subproblemas, el orden óptimo de taladrar las placas y los caminos óptimos que comunican los chips. En los problemas de perforado hemos de tomar los nodos o ciudades como las posiciones a perforar, y las distancias entre ellas como el tiempo que necesita la máquina en trasladarse de una a otra. El punto inicial y final será un punto adicional donde permanece la perforadora en el periodo de inactividad. Si estas máquinas no son correctamente programadas el tiempo que tarda en desplazarse de un orificio a otro puede ser significativo con lo que la producción de placas no sería eficiente.

En el problema de conexión de chips la idea es minimizar la cantidad de cable necesaria para minimizar todos los puntos de una placa sin que haya interferencias. Como los chips son de pequeño tamaño, no se pueden poner más de dos cables en un único pin. Tomando como nodos los pins y la cantidad de cable necesaria para unirlos como la distancia, el problema es equivalente al del viajante de comercio.

Bibliografía

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network flows. Theory, Algorithms and Applications*, Prentice Hall, 1993.

[2] J. Bertin, *Graphics and graphic information processing*, Morgan Kaufmann Publishers Inc., 1999.

[3] Dell'Amico M. Burkard, R. and S. Martello, *Assignment Problems: Revised Reprint*, SIAM, 2009.

[4] M. Hahsler and K. Hornik, *TSP – Infrastructure for the Traveling Salesperson Problem*, Journal of Statistical Software **23** (2007).

[5] J. Lenstra and A.R. Kan, *Some simple applications of the travelling salesman problem*, Operational Research Quarterly (1975).

[6] S. Martello, *Jenó Egerváry: from the origins of the Hungarian algorithm to satellite communication*, Central European Journal of Operations Research **18** (2010), 47–58.

[7] C. H. Papadimitriou, *The Euclidean traveling salesman problem is NP-complete*, Theoretical Computer Science (1977).