01171
2
2a

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"TECNICAS PARA EL PROBLEMA DE ASIGNACION"

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRA EN INGENIERIA

(INVESTIGACION DE OPERACIONES)

P R E S E N T A

NORA MARIA DE LOURDES ORTIGOSA ZEPEDA

BAJO LA DIRECCION M. EN I. IDALIA FLORES DE LA MOTA

MEXICO, D. F. 1994

Agradecimientos.

A la M. en I. Idalia Flores de la Mota, por haber aceptado dirigir este trabajo y por el entusiasmo que manifestó durante la realización del mismo.

A los miembros del jurado

Dr. Miguel Angel Gutiérrez Andrade
M. en I. Maclovio Sauto Vallejo
M. en I. Rubén Téllez Sánchez
M. en I. Ricardo Aceves García

por el tiempo dedicado para revisar, corregir y enriquecer el trabajo original.

A mis padres, Ma. de Lourdes Zepeda de Ortigoza y Gervasio Ortigoza Guerrero, por su apoyo y aliento; y por decirme -¡Pero querías maestría!- cuando me quejaba. Gracias por hacer de mi lo que ahora soy; con su ejemplo, amor y respeto juntos lo hemos logrado.

Al M. en I. Moisés Silva González, por su ayuda que fue determinante para concluir este trabajo. Gracias por todo, donde TODO implica TODO.

A mis amigos y compañeros, de la D.E.P.F.I.

Dedicatoria.

Al Ratón con todo mi amor.

A mis papás.

Indice.

INTRODUCCION.

La Investigación de Operaciones, es la aplicación de métodos matemáticos, cuantitativos utilizados para argumentar las decisiones en todas las esferas de la actividad humana orientada hacia una finalidad.

La Investigación de Operaciones es una ponderación matemática de las futuras decisiones que permite ahorrar tiempo, fuerzas y recursos materiales, para así evitar errores graves, pues de lo contrario resultaría demasiado caro.

El objetivo de la Investigación de Operaciones radica en argumentar previa y cuantitativamente las decisiones óptimas.

Los problemas prototipo de la Investigación de Operaciones son:

1.) Asignación
2.) Inventario
3.) Reemplazo
4.) Líneas de espera
5.) Secuenciación y coordinación
6.) Trayectorias
7.) Competencia
8.) Búsqueda

En este trabajo se presentan técnicas para resolver el

Problema de Asignación. El Problema de Asignación implica la asignación de recursos a trabajos que deben ejecutarse. El objetivo es asignar los recursos a los trabajos de manera que se optimice el rendimiento total o el costo total.

En el primer capítulo se presenta el problema de asignación, como un caso particular del problema de transporte. Los modelos primal y dual del problema de asignación. Así como un ejemplo.

El segundo capítulo propone el método de ramificación y acotamiento, para resolver el problema de asignación. Y un ejemplo, en donde se aplica el algoritmo de ramificación y acotamiento.

Una técnica muy interesante para resolver el Problema de Asignación es el Problema de Acoplamiento pero en gráficas bipartitas. Los conceptos y teoremas para este tema se presentan en el capítulo 3.

Conociendo la terminología del problema de acotamiento, pueden abordarse los capítulos cuatro y cinco. En el capítulo cuatro se supone que un trabajador i puede o no realizar un trabajo j ($\forall$ i,j = 1,...,n). Por tal motivo en la matriz de eficiencia sólo puede haber 0's y 1's; 0 si el trabajador no puede realizar el trabajo y 1 si el trabajador puede realizar el trabajo. Mientras que en el quinto capítulo todos los trabajadores pueden realizar todos los trabajo y además se conoce la eficiencia de cada trabajador en cada uno de los trabajos.

En estos dos últimos capítulos se proponen el algoritmo Húngaro, el algoritmo de Kuhn-Munkres y el algoritmo de las Dos Fases. Y para cada uno de ellos un ejemplo en el què se muestra el funcionemiento de éstos.

Para finalizar, en los apéndices se proporcionan los programas fuente de los algoritmos Húngaro y de Kuhn-Munkres, editados en

Turbo Pascal. En éstos el orden de la matriz y la matriz de eficiencia pueden darse por teclado o por archivo. Si el caso es el último, el archivo debe estar en código ASCII y el primer elemento debe ser el orden de la matriz. Debido a la capacidad de la memoria de la computadora, donde se elaboraron los programas, el tamaño máximo de la matriz para el algoritmo Húngaro es de 100x100 y para el algoritmo de Kuhn-Munkres es 60x60.

Capítulo 1.

EL PROBLEMA DE ASIGNACION.

Una clase importante de los problemas de Programación Lineal es la del Problema de Transporte. En los problemas de transporte, se consideran m *puntos localizados en un mapa, donde el origen* i *tiene una provisión de* a_i *unidades de un determinado producto. También en el mapa existen* n *puntos de destino, donde el destino* j *requiere* b_j *unidades del producto. Se conoce además el costo unitario* c_{ij} *por transportar el producto del origen* i *al destino* j.

El problema de transporte consiste en determinar los embarques factibles de los orígenes a los destinos que minimicen el costo total de transporte. Si la oferta total excede a la demanda total, se debe introducir un destino ficticio con demanda $b_{n+1} = \sum a_i - \sum b_j$ *y* $c_{i,n+1} = 0$ *para que así se cumpla*

$$\sum a_i = \sum b_j.$$

El modelo de programación lineal para el problema de transporte balanceado es entonces:

$$Min \sum \sum c_{ij} x_{ij}$$

s.a

$$\begin{array}{lllllllll}
x_{11} + \cdots + x_{1n} & & & & & & & = a_1 \\
& x_{21} + \cdots + x_{2n} & & & & & & = a_2 \\
& & \ddots & & & & & \\
& & & x_{m1} + \cdots + x_{mn} & & & & = a_m \\
x_{11} & + x_{21} & \cdots & + x_{m1} & & & & = b_1 \\
\ddots & & & & & & & \\
x_{1n} & + x_{2n} & + & \cdots + x_{mn} & & & & = b_n
\end{array}$$

$$\forall\ x_{ij} \geq 0 \qquad \forall\ i = 1, \ldots, m \quad j = 1, \ldots, n.$$

Mientras que la representación gráfica del problema es la que se muestra en la ilustración 1.1.

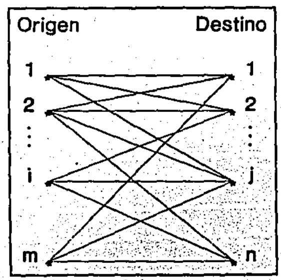

Ilustración 1.1.

También puede representarse mediante una matriz como se muestra a continuación:

		DESTINOS					**OFERTA**
		1	*2*	*3*		*n*	
ORIGENES	*1*	c_{11}	c_{12}	c_{13}	$\cdots$	c_{1n}	a_1
	2	c_{21}	c_{22}	c_{23}	$\cdots$	c_{2n}	a_2
	.						
	.						
	m	c_{m1}	c_{m2}	c_{m3}	$\cdots$	c_{mn}	a_m
DEMANDA		b_1	b_2	b_3	$\cdots$	b_n	

Un caso especial del problema de transporte surge cuando $m = n$, $a_i = 1$ *y* $b_j = 1$. *Este caso especial se llama* ***EL PROBLEMA DE ASIGNACION.***

Los problemas de asignación implican la asignación de individuos a trabajos que deben ejecutarse; la restricción que existe en este tipo de problemas es que a cada persona se le asignará un solo trabajo y a cada trabajo se le asignará una sola persona.

Si el costo de asignar al individuo i el trabajo j es c_{ij}, *entonces se tiene un problema de asignación lineal, el cual puede escribirse como un modelo matemático de la siguiente forma:*

$$Min \sum_{(i,j) \in A} x_{ij}$$

s. a

$$\sum_{(i,j) \in A} x_{ij} = 1 \quad \forall\, i \in X$$

$$\sum_{(i,j) \in A} x_{ij} = 1 \quad \forall\, j \in Y$$

Donde $X = \{ \text{individuos} \}$, $Y = \{ \text{trabajos} \}$; *de manera tal que* $|X| = |Y|$.
$A \subseteq X \times Y$ *que representa una posible asignación del individuo i al trabajo j ((i,j)).*
c_{ij} *es el costo asociado a cada elemento* $(i,j) \in A$. $c_{ij} \in N$.
$x_{ij} = 0$ *si el individuo i **no** es asignado al trabajo j.*
$x_{ij} = 1$ *si el individuo i es asignado al trabajo j.*

*Este problema es de tipo lineal, con la estructura del problema de transporte, solo que como ya se dijo la oferta en cada origen es de valor uno y la demanda en cada destino es de valor uno. Este tipo de problemas se resuelven utilizando métodos llamados **"ALGORITMOS DE ASIGNACION"** que son mucho más eficientes que el simplex o que los métodos de transporte, ya que su complejidad computacional es menor.*

El dual del problema de asignación con las restricciones de no negatividad reemplazando las restricciones de 0 ó 1 resulta ser:

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

s.a

$$u_i + v_j \leq c_{ij} \quad i,j = 1,\ldots,n.$$

$$u_i,\ v_j \text{ no restringidas } \quad 1,j = 1,\ldots,n.$$

Las condiciones de holgura complementarias están dadas por

$$(c_{ij} - u_i - v_j)x_{ij} = 0 \quad i,j, = 1, \ldots, m$$

Si puede encontrarse un conjunto de las u, v y x factibles que satisfagan la holgura complementaria, esas u, v y x serán óptimas.

En una solución dual factible están dadas por:

$$uHATi = \underset{1\leq j\leq m}{Mínimo}(c_{ij}) \quad i = 1,\ldots,m$$

$$\hat{v}j = \underset{1\leq i\leq m}{Mínimo}(c_{ij} - \hat{u}i) \quad j = 1,\ldots,m$$

Por lo cual vemos que u_i es el mínimo c_{ij} en el renglón i, y que v_j es el mínimo $c_{ij} - u_i$ en la columna j.

ALGORITMO HUNGARO.

Propósito: *Determinar la asignación óptima, dada la matriz de eficiencia, utilizando el método Húngaro.*

Descripción:

PASO 1: Dada una matriz de costos de un problema de asignación, reste en cada columna y en cada renglón el número más pequeño de esa columna o renglón, del resto de los elementos en esa columna o renglón.

PASO 2: En la nueva matriz de costos, selecciónese un cero en cada renglón y columna. Elimine durante el proceso de selección la columna y el renglón al que pertenece el cero seleccionado. Si al finalizar este paso se ha hecho una asignación completa de ceros, cada trabajador tiene asignado un trabajo y cada trabajo tiene asignado un solo trabajador, se ha encontrado la asignación óptima. En caso contrario ir al paso 3.

PASO 3:

3.1 Marque cada fila que no contiene un cero asignado.

3.2 Marque cada columna que contiene un cero (no necesariamente asignado) en la fila marcada en el paso 3.1.

3.3 Marque cada fila que contiene un cero asignado en la columna marcada en el paso 3.2.

3.4 Repita los pasos 3.2 y 3.3 hasta que no se puedan marcar mas columnas o filas.

3.5 Tache las filas no marcadas y las columnas marcadas.

3.6 Selecciónese al número más pequeño de los elementos no cubiertos por una tachadura horizontal o vertical. Reste ese elemento del resto de los no tachados y sume ese elemento a los tachados en cruz, es decir, por una tachadura horizontal y vertical. Los elementos cruzados por una sola tachadura no cambian. Regrese al paso 2.

EJEMPLO:

Encontrar la solución de costo mínimo del problema de Asignación cuya matriz de eficiencia está dada por:

	1	2	3	4	5	6
1	17	8	12	9	14	6
2	15	13	18	15	10	4
3	14	16	8	12	5	9
4	18	7	14	9	11	13
5	7	16	11	14	6	10
6	18	9	13	5	15	7

ITERACION 1:

PASO INICIAL: *Como el mínimo en el renglón 1 es 6, se resta a los elementos del renglón 1, de igual forma 4 a los elementos del renglón 2, 5 a los elementos del renglón 3, 7 a los elementos del renglón 4, 6 a los elementos del renglón 5 y 5 a los elementos del renglón 6. Se obtiene la matriz 1. Considerando la matriz 1, se sigue el mismo procedimiento restando 1 a los elementos de la columna 1 y se resta 3 a los elementos de la columna 3. Se obtiene la matriz 2 a la que llamamos matriz reducida.*

	1	2	3	4	5	6
1	11	2	6	3	8	0
2	11	9	14	11	6	0
3	9	11	3	7	0	4
4	11	0	7	2	4	6
5	1	10	5	8	0	4
6	13	4	8	0	10	2

Matriz 1.

	1	2	3	4	5	6
1	10	2	3	3	8	0
2	10	9	11	11	6	0
3	8	11	0	7	0	4
4	10	0	4	2	4	6
5	0	10	2	8	0	4
6	12	4	5	0	10	2

Matriz 2.

PASO PRINCIPAL: *Considerando la matriz 2, se trazan el mínimo de líneas sobre los renglones y columnas para cubrir todos los ceros de la matriz reducida. A los elementos* a_{11} , a_{13} , a_{21} , a_{23} , a_{41} , a_{43} , a_{61} *y* a_{63} *se les resta 3 que es el mínimo elemento no cubierto;mientras que a los elementos* a_{32} , a_{34} , a_{35} , a_{52} , a_{54} *y* a_{55} *se les suma 3 ya que están cubiertos por 2 líneas. Se obtiene la matriz 3.*

	1	2	3	4	5	6
1	7	2	0	3	8	0
2	7	9	8	11	6	0
3	8	14	0	10	3	7
4	7	0	1	2	4	6
5	0	13	2	11	3	7
6	9	4	5	0	10	2

Matriz 3.

ITERACION 2:

PASO INICIAL: Se resta 3 a los elementos de la columna 5 de la matriz 3.

	1	2	3	4	5	6
1	7	2	0	3	5	0
2	7	9	8	11	3	0
3	8	14	0	10	0	7
4	7	0	1	2	1	6
5	0	13	2	11	0	7
6	9	4	5	0	7	2

Se traza el mínimo número de líneas, como son 6 que es igual al tamaño de la matriz, se dispone de una solución óptima que es la siguiente:

De la matriz final que se obtuvo, se hace la siguiente asignación

al hombre 1 se le asigna la tarea 3
al hombre 2 se le asigna la tarea 6
al hombre 3 se le asigna la tarea 5
al hombre 4 se le asigna la tarea 2

al hombre 5 se le asigna la tarea 1
al hombre 6 se le asigna la tarea 4.

El valor de la función objetivo es 40.

Capítulo 2.

LA TECNICA DE RAMIFICACION Y ACOTAMIENTO PARA RESOLVER EL PROBLEMA DE ASIGNACION.

Existen problemas de optimización sujetos a restricciones para los que los métodos directos no existen o son ineficaces, ya sea porque la función objetivo o las restricciones no son convexas o porque las variables sólo pueden tomar valores discretos.

El método de ramificación y acotamiento surgió hace dos décadas, es una de las mejores herramientas prácticas para la solución de problemas de optimización discreta. El método consiste en particionar el problema difícil en subconjuntos más simples o *fáciles; se selecciona un subconjunto prometedor, se busca la mejor solución y se almacena esta información. Se particiona de nuevo el subconjunto en subconjuntos más simples y se repite el mismo proceso. Lo interesante de este método radica en que se pueden eliminar implícitamente grupos grandes de soluciones potenciales sin evaluarlos explícitamente.*

Supóngase que se tiene una optimización restringida difícil:

$$Min\ C^{(0)}(x) = Z(x)$$

$$s.\ a\quad g_1^{(0)}(x) \geq 0$$

$$\vdots$$

$$g_m^{(0)}(x) \geq 0$$

$$x \in X^{(0)}$$

en donde $X^{(0)}$ representa el dominio de optimización permitible para que x, represente un vector que satisfaga las restricciones y que además sea solución factible y uno que además minimiza la función objetivo es una solución óptima.

Hay cuatro razones importantes para aceptar los algoritmos de ramificación y acotamiento en los problemas de optimización discreta:

1.- El método es conceptualmente simple y fácil de entender.

2.- Es fácil de adaptar a un amplio rango de situaciones problemáticas.

3.- Es ajustable para su implantación computacional.

4.- Los métodos alternativos usualmente no están disponibles.

El problema que se resuelva por el método de ramificación y acotamiento debe cumplir las siguientes propiedades:

a.- Naturaleza Combinatoria. *Un problema combinatorio debe tener como mínimo las siguientes propiedades:*

a.1.- *Cada conjunto tiene un número finito de objetos.*

a.2.- *Cada objeto puede tomar un cierto rango de atributos.*

a.3.- *Se desarrolla una solución al problema fijando los valores de atributos para todos y cada uno de los objetos.*

a.4.- *Unicamente se permiten ciertas combinaciones de los valores de los atributos.*

b.- Ramificabilidad. *Es una propiedad del problema que implica:*

b.1.- *Construir un conjunto finito y contable que contenga todas las soluciones del problema.*

b.2.- *Particionar recursivamente un conjunto no vacío de soluciones en subconjuntos disjuntos.*

c.- Racionalidad. *Un problema racional tal que:*

c.1.- *Cada solución tiene un único valor calculado de los valores de sus atributos.*

c.2.- *La mejor solución es aquella con mayor (o menor) valor.*

d.- Acotabilidad. *Una estimación de valor de la mejor solución contenida en cualquier conjunto de soluciones se puede obtener de manera tal que:*

d.1.- *El valor actual de la mejor solución en el conjunto es inferior o igual a la estimación (así la estimación es una cota superior).*

d.2.- *Se hace un mínimo esfuerzo para obtener dicha estimación.*

d.3.- *La estimación es razonablemente cercana al valor actual.*

Para poder construir un árbol que describa todas las operaciones realizadas en el problema y efectuar una búsqueda para encontrar la mejor solución, el método de ramificación y acotamiento explota las propiedades anteriores. A este árbol se le *denomina árbol de búsqueda.*

Cada nodo del árbol está asociado a un subconjunto de T, donde T es el conjunto de todas las soluciones originales del problema. Sea T_i *un subconjunto de T, es decir, una colección de soluciones del problema. La ramificación en el proceso de particionar un subconjunto* T_i *en m subconjuntos disjuntos* $v_1, v_2, \ldots, v_m$ *tal que*

$$\bigcup_{j=1}^{m} v_j = T_i \quad y \quad v_i \cap v_j = \emptyset \qquad i \neq j.$$

El proceso de ramificación se puede visualizar como la creación o desarrollo ordenado de un árbol donde el nodo inicial representa a T, y los nodos restantes representan subconjuntos T_i *de T. A cada nodo N se le asocia un subconjunto T.*

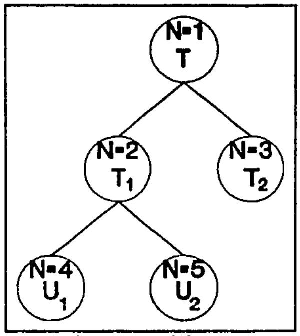

Ilustración 2.1.

La relación , entre subconjuntos, de T y la ramificación para la creación del árbol , se formaliza como sigue:

- $T_i(N)$ *es el subconjunto* T_i *de T representado por el Nodo N.*

- Un nodo intermedio es un nodo N en el cual no ha habido ramificación.

_ Un nodo final es un nodo intermedio N para el cual $T_i(N)$ *consiste de una solución única.*

_ L(N) es una cota inferior de la función objetivo Z(x) en el conjunto de las soluciones asociadas con el nodo N, esto es, $L(N) \leq Z(x)$ *para toda* $x \in T_i(N)$.

Para describir en una forma conceptual y general el método de ramificación y acotamiento se utiliza el siguiente algoritmo:

ALGORITMO BASICO DE RAMIFICACION Y ACOTAMIENTO.

Propósito: *Encontrar la solución óptima de un problema de optimización discreta.*

Descripción:

PASO 0: Se inicia con el conjunto de todas las soluciones factibles del problema en cuestión. Se forma el primer nodo del árbol.

PASO 1: Se procede a ramificar los nodos hacia nodos nuevos, utilizando alguna regla de ramificación.

PASO 2: Se determinan cotas inferiores para los nuevos nodos. Para cada nuevo nodo N se obtiene una cota inferior L(N) sobre el valor de la función objetivo para las soluciones factibles del nodo.

PASO 3: Se selecciona un nodo intermedio desde el cual se ramifica. Cada nuevo nodo se excluye o se sondea si: Se encuentra que el nodo no contiene soluciones factibles, o se ha identificado la mejor solución factible en el nodo , (L(N) corresponde al valor de su función objetivo).

PASO 4: Reconocer cuando un nodo final contiene una solución óptima. El proceso termina cuando no existen nodos restantes (no sondeados) y la solución de apoyo actual es la solución óptima, (si no existe tal solución de apoyo entonces el problema no tiene soluciones factibles). De otra manera se regresa al paso 1.

NOTA: Si en vez de minimizar se maximiza, los conceptos anteriores son válidos cambiando cotas inferiores por superiores o viceversa, así como el sentido de las desigualdades.

EL PROBLEMA DE ASIGNACION.

Este problema queda definido por una matriz de eficiencia *construida con la calificación de cada uno de los individuos a cada uno de los trabajos ofrecidos. Como ya se dijo anteriormente la matriz debe ser cuadrada.*

En este caso para seleccionar el nodo intermedio es muy simple, se elige aquella de mayor (menor) etiqueta; por lo que se refiere a la regla de obtener nuevos problemas acotantes, el algoritmo es el siguiente:

Algoritmo de Ramificación y Asignación.
Para el Problema de Asignación.

Propósito: *Determinar la asignación óptima dada la matriz de eficiencia, utilizando el método de ramificación y acotamiento.*

Descripción:

PASO 1: Partir $[a_{ij}]_{(nxn)}$ *en vectores columna j para j. Definir* $c_j = \max a_{ij}$.

PASO 2: Fijar la raíz y etiquetarla con $C = \Sigma c_j$. *Ramificar desde la raíz con n ramas i.*

PASO 3: Elegir un vector no marcado j y marcarlo con $[a_{ij}]$ *e identificar el nodo con (i,j).*

PASO 4: Si se han etiquetado todos los nodos ir al paso 7; en caso contrario ir al paso 5.

PASO 5: Seleccionar el nodo (i^*,j^*) *no etiquetado. Recorrer la rama desde la raíz hasta* $(i^*;j^*)$. *Formar* $A_{i^*j^*}$ *y* B_{j^*}:
$A_{ij} = \Sigma a_{ij}$ *con* (i_o,j_o) *en la rama* $B_j = \Sigma c_j$.

PASO 6: Para (i^*,j^*) *hacer* $D_{i^*j^*} = A_{i^*j^*} + B_{j^*}$. *Poner f si* $D_{i^*j^*}$ *contiene sólo un elemento de cada renglón. Ir al paso 4.*

PASO 7: Hacer v.a.f.o. (valor actual de la función objetivo) = etiqueta mayor con f; c.s.m. (cota superior máxima) = etiqueta mayor con o sin f.

PASO 8: Si c.s.m. = v.a.f.o. los nodos (i_o,j_o) *en la rama donde está la c.s.m. y las (i,j) correspondientes a los max* a_{ij}

*cuya $j \neq j_o$ constituyen la asignación óptima. **TERMINA**.*

PASO 9: Cancelar los nodos con c.s. < v.a.f.o. . m = número de nodos en la rama asignada. Ramificar desde el nodo asignado con $(m - n)$ ramas. Ir al paso 3.

EJEMPLO 2.1. *Se necesita asignar a las personas A, B, C y D en los puestos α, β, γ y δ . Si la matriz de eficiencia es:*

	α	β	γ	δ
A	3	7	4	2
B	4	3	5	4
C	2	5	6	3
D	1	3	5	3

efectuar la asignación maximizando.

Iteración 1:

Se divide la matriz en 4 vectores columna, y definimos c_j de la siguiente manera: $c_1 = 4$, $c_2 = 7$, $c_3 = 6$, $c_4 = 4$.

Se elige al vector 1, $j = 1$, donde $A_{11} = 3$, $A_{21} = 4$, $A_{31} = 2$, $A_{41} = 1$; $B_1 = \Sigma c_j = 7 + 6 + 4 = 17$.

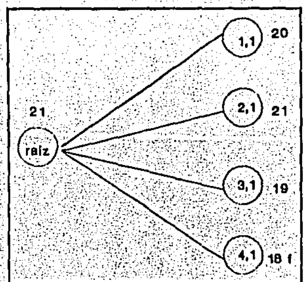

Ilustración 2.2.

$$D_{ij} = A_{ij} + B_j;$$

$D_{11} = 3 + 17 = 20$, $D_{21} = 4 + 17 = 21$, $D_{31} = 2 + 17 = 19$, $D_{41} = 1 + 17 = 18$.

Las asignaciones D_{11}, D_{21} y D_{31} no son factibles, ya que existen dos asignaciones en los renglones. Por lo tanto únicamente (4,1) es factible. Ilustración 2.2.

v.a.f.o. = 18 c.s.m = 21

Asignamos el nodo (2,1) con 4-1 =3 ramas.

Iteración 2:

La matriz ahora es:

	β	γ	δ
A	7	4	2
C	5	6	3
D	3	5	3

La matriz se divide en 3 vectores columna, con $j = 2, 3, 4$; y definimos c_j de la siguiente manera: $c_2 = 7$, $c_3 = 6$, $c_4 = 3$.

Se elige al vector $j = 2$, donde $A_{12} = 7 + 4 = 11$, $A_{32} = 5 + 4 = 9$, $A_{42} = 3 + 4 = 7$. $B = \Sigma\, c_j = 3 + 6 = 9$. $D_{12} = 11 + 9 = 20$, $D_{32} = 9 + 9 = 18$, $D_{42} = 7 + 9 = 16$. Ilustración 2.3.

v.a.f.o. = 20.

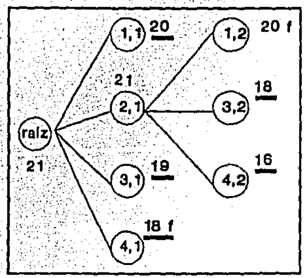

Ilustración 2.3.

Iteración 3:

La matriz ahora es:

	γ	δ
C	6	3
D	5	3

La matriz se divide en 2 vectores columna, con $j = 3, 4$. y definimos c_j de la siguiente manera: $c_3 = 6$, $c_4 = 3$.

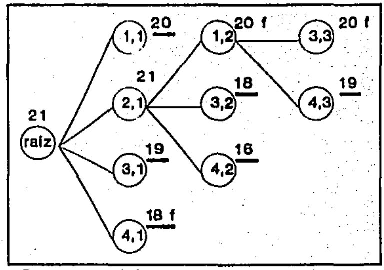

Ilustración 2.4.

Se elige al vector 3, $j = 3$, donde $A_{33} = 4 + 7 + 6 = 17$, $A_{43} = 4 + 7 + 5 = 16$, $B_3 = \Sigma\ c_j = 3$.
$D_{33} = 17 + 3 = 20$, $D_{43} = 16 + 3 = 19$. Ilustración 2.4.

v.a.f.o. = 20

∴ La asignación óptima es: a_{21}, a_{12}, a_{33} y a_{44}.

∴ El árbol que representa esta asignación es el que se muestra en la página siguiente. (Ilustración 2.5.)

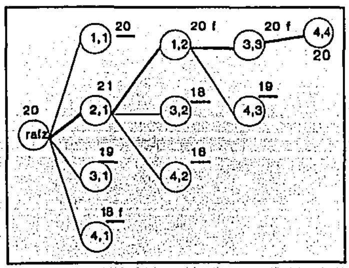

Ilustración 2.5.

Capítulo 3.

EL PROBLEMA DE ACOPLAMIENTO MAXIMO EN GRAFICAS BIPARTITAS.

Un método para determinar una asignación puede ser el de acoplamiento. En este capítulo se estudiará el problema de acoplamiento para gráficas bipartitas, para después poder abordar el problema de asignación utilizando este método en los capítulos siguientes.

Definición 3.1. *Una gráfica no dirigida $G = (V,E)$ se dice bipartita, si el conjunto de vértices V puede ser dividido en dos subconjuntos X y Y tales que todos los arcos tiene un vértice terminal en X y otro en Y. (Ilustración 3.1.)*

Definición 3.2. Un acoplamiento en una gráfica no dirigida $G = (X,Y,E)$ es un subconjunto M del conjunto E de arcos tal que dos arcos no son adyacentes en M.

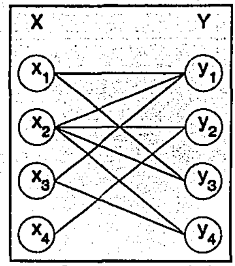

Ilustración 3.1.

En la ilustración 3.2. $x_1\,y_3$, $x_3\,y_1$, $x_4\,y_2$ $\in M$.

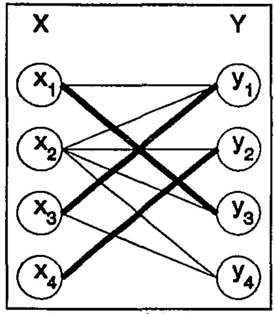

Ilustración 3.2.

El problema de acoplamiento consiste en encontrar un acoplamiento de cardinalidad máxima en una gráfica. Este problema se puede expresar como un problema de programación lineal 0-1 de la siguiente manera:

$$Max\ z = \sum_{k=1}^{m} c_k\,\zeta_{ij}$$

$$\sum \zeta_{ij} \leq 1 \qquad \forall\ i = 1,\ldots n.$$

donde: c_k es el costo asociado al arco (x_i, y_j).
$\zeta_{ij} = 1$ si (x_i, y_j) está en el acoplamiento.
$\zeta_{ij} = 0$ en otro caso

Definición 3.3. Los extremos de un arco en M se llaman acoplados bajo M. Un acoplamiento M satura un vértice v, y v se dice saturado por M si algún arco de M incide con v; en otro caso es no saturado por M. En la ilustración 3.3. x_1 , x_3 , x_4 , y_1 , y_2 y y_3 son saturados por M. Mientras que x_2 y y_4 son no saturados por M.

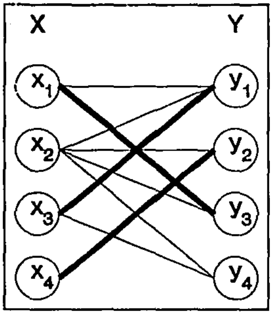

Ilustración 3.3.

Definición 3.4. Si todos los vértices de G son saturados por M, el acoplamiento es ***perfecto****. (Ilustración 3.4.)*

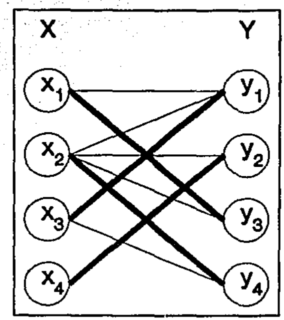

Ilustración 3.4.

Definición 3.5. Sea M un acoplamiento en G. Una ***cadena M-alternante****, es una cadena cuyos arcos están alternadamente en E\M y M. En la ilustración 3.5., los arcos delgados están en E\M y los gruesos en M.*

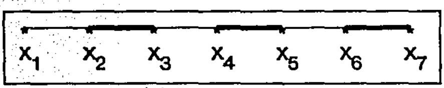

Ilustración 3.5.

Definición 3.6. Una cadena M-aumentante es una cadena M-alternante cuyos vértices inicial y final son no saturados por M. En la ilustración 3.6. x_2 y x_7 no están saturados por M. La cardinalidad de M es 3.

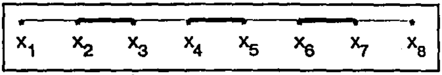

Ilustración 3.6.

La nueva cadena M-alternante ahora tiene un elemento más en el acoplamiento M, es decir, ahora su cardinalidad es 4 (Ilustración 3.7.)

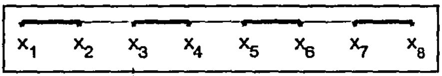

Ilustración 3.7.

*Definición 3.7. La diferencia simétrica de dos conjuntos se indica por Δ, es decir, **M Δ M*** es el conjunto de todos los elementos contenidos en M o en M^*, pero no en ambos. (Ilustración 3.8.)*

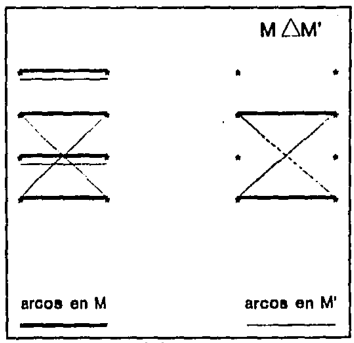

Ilustración 3.8.

<u>Teorema 3.1. (Berge 1957).</u> Un acoplamiento M en G es un acoplamiento máximo si y sólo si G no contiene una cadena M-aumentante.

Demostración:

"⇒" Sea M un acoplamiento en G, y supongamos que G contiene una cadena M-aumentante v_0, v_1, ..., v_{2m+1}. Definimos $M' \subseteq E$ por: $M' = (M \setminus \{ v_1 v_2 , v_3 v_4 , \ldots, v_{2m-1} v_{2m} \}) \cup \{ v_0 v_1 , v_2 v_3 , \ldots, v_{2m} v_{2m+1} \}$

Entonces M' es un acoplamiento en G y $|M'| = |M| + 1$. Así M no es un acoplamiento máximo.

"⇐" Supóngase que M no es un acoplamiento máximo, y sea M' un acoplamiento máximo en G. Entonces $|M'| > |M|$. Sea $H = G[M \Delta M']$ (Ilustraciones 3.9.a. y 3.9.b.)

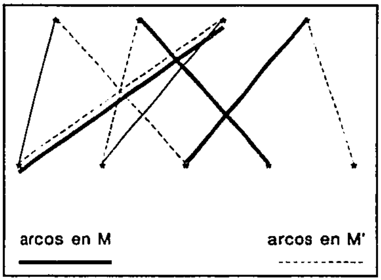

Ilustración 3.9.a.

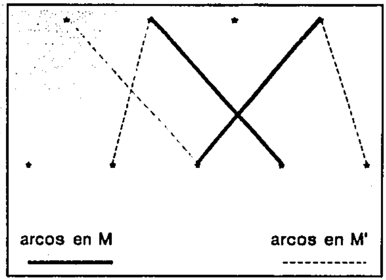

Ilustración 3.9.b.

Cada vértice de H tiene grado 1 ó 2 en H, dado que éstos pueden incidir con al menos un arco de M y un arco de M'. Así cada componente de H es un ciclo par con arcos alternados en M y M', o una cadena con arcos alternadamente en M y M'. Como $|M'| > |M|$, H contiene más arcos de M' que de M, y por lo tanto alguna cadena compuesta de H debe iniciar y terminar con arcos de M'. El origen y el fin de P, son saturados por M' en H, y no saturados por M en G. Así P es una cadena M-aumentante en G.

■

Definición 3.8. Para cualquier conjunto S de vértices en G definimos el ***conjunto vecindad de S en G*** *como el conjunto de todos los vértices adyacentes a vértices en S, se denota por $N_G(S)$. (Ilustración 3.10.)*

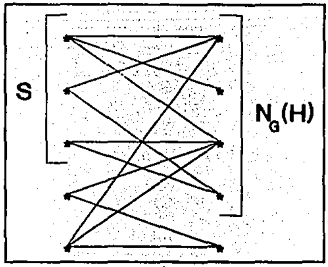

Ilustración 3.10.

En muchas aplicaciones se desea encontrar un acoplamiento de G que sature todos los vértices en X, un ejemplo puede ser el Problema de Asignación de Personal. Las condiciones necesarias y suficientes para la existencia de tal acoplamiento fueron dadas primero por Hall en el siguiente teorema.

<u>Teorema 3.2. (Hall 1935).</u> Sea G una gráfica bipartita, con bipartición (X,Y). Entonces G contiene un acoplamiento que satura todos los vértices en X si y sólo si $|N(S)| \geq |S|$.

<u>Demostración:</u>

<u>"⇒"</u> Supongamos que G contiene un acoplamiento M que satura todo vértice en X, y sea S un subconjunto de X. Como los vértices en S son acoplados por M, con vértices distintos en $N(S)$, claramente tenemos $|N(S)| \geq |S|$.

<u>"⇐"</u> Supongamos que G es una gráfica bipartita que satisface $|N(S)| \geq |S|$, pero que G no contiene un acoplamiento que sature todos los vértices en X. Obtenemos entonces una contradicción. Sea M^* un acoplamiento máximo en G. Para nuestra suposición, M^* no satura todos los vértices en X. Sea u un vértice no saturado por M en X, y Z denota el conjunto de todos los vértices conectados a u por cadenas M^*-alternantes. Como M^* es un acoplamiento máximo, se obtiene del teorema 3.1, que u es el único vértice no saturado por

M^* en Z. $S = Z \cap X$ y $T = Z \cap Y$.
(Ilustración 3.11.).

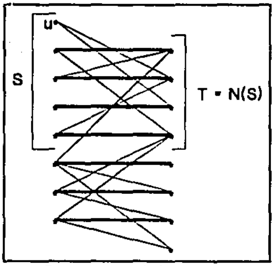

Ilustración 3.11.

Podemos ver que los vértices en $S \setminus \{u\}$ son acoplados bajo M^* con vértices en T. Entonces $|T| = |S| - 1$ y $N(S) \supseteq T$. En efecto, tenemos que $N(S) = T$, dado que todo vértice en $N(S)$ está conectado a u por una cadena M^*-alternante. Pero $|T| = |S| - 1$ y $N(S) = T$ implican que $|N(S)| = |S| - 1 < |S|$, contradiciendo que $|N(S)| \geq |S|$ para toda $S \subseteq X$.

■

La demostración de este teorema nos ofrece las bases de un algoritmo bueno para encontrar un acoplamiento máximo en una gráfica bipartita.

Definición 3.9. *El grado $d_G(v)$ de un vértice en G es el número de arcos de G incidentes con v.*

Definición 3.10. *Una gráfica G es regular si todos los vértices de G son de igual grado.*
respectivamente. Por definición de $N(S)$, $E_1 \subseteq E_2$ y por lo tanto

$$k|N(S)| = |E_2| \geq |E_1| = k|S|.$$

Definición 3.11. Si G es regular con $d(v_i) = k$ para todos los vértices v_i en G, entonces G es llamada k-regular. (Ilustración 3.12.)

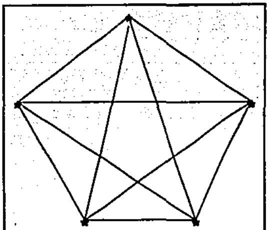

Ilustración 3.12.

<u>Corolario 3.1.</u> Si G es una gráfica k-regular, con $k > 0$, entonces G tiene un acoplamiento perfecto.

<u>Demostración:</u> Sea G una gráfica k-regular con bipartición (X,Y). Como G es k-regular, $k|X| = E = k|Y|$ y como $k > 0$, $|X| = |Y|$. Sea ahora S un subconjunto de X y denotemos por E_1 y E_2 los subconjuntos de arcos incidentes con vértices en S y N(S), S e obtiene que $|N(S)| \geq |S|$ y por lo tanto, por el teorema 3.2, G tiene un acoplamiento M que satura todos los vértices en X. Por lo tanto $|X| = |Y|$, y M es un acoplamiento perfecto.

■

El corolario 3.1. en algunas ocasiones se conoce como el **Teorema del Matrimonio**, puede ser expresado en la siguiente forma: si una chica en una aldea conoce exactamente K chicos y cada chico conoce exactamente k chicas, entonces cada chica puede casarse con un chico que ella conoce y cada chico puede casarse con una chica que el conoce. (A cada chica se le asigna un chico).

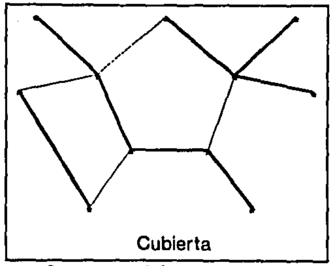

Ilustración 3.13.

Definición 3.12. Una cubierta de una gráfica G es un subconjunto K de V tal que todo arco de G tiene al menos un vértice en K. (Ilustración 3.13)

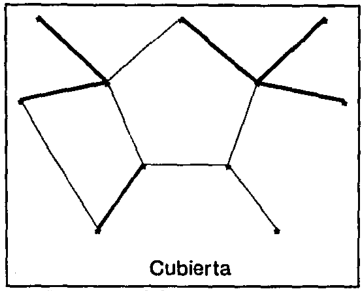

Ilustración 3.14.

*Definición 3.13. Una cubierta K es una **cubierta mínima** si G no contiene una cubierta K′ con $|K'| \le |K|$. (Ilustración 3.14)*

Podemos asegurar que si K es una cubierta de G, y M es un acoplamiento de G, entonces K contiene al menos un vértice de cada arco en M. Se cumple que para cualquier acoplamiento M y cualquier cubierta K, $|M| \le |K|$. Cumpliéndose además que si M^* es un acoplamiento máximo y K′ es una cubierta mínima, entonces $|M^*| \le |K'|$. La igualdad se cumple sólo cuando la gráfica es bipartita, es decir cuando $|M^*| = |K'|$. Este resultado se debe a Köing (1931) y está relacionado con el teorema de Hall. Antes de enunciarlo veamos un lema.

Lema 3.1. ***Sea M un acoplamiento y K una cubierta tal que*** $|M| = |K|$***. Entonces M es un acoplamiento máximo y K es una cubierta mínima.***

Demostración: Si M^* es un acoplamiento máximo y K' es una cubierta mínima, como $|M^*| \leq |K'|$ entonces $|M| \leq |K'| \leq |K|$. Como $|M| = |K|$ se obtiene que $|M| = |M^*|$ y $|K| = |K'|$.

■

Teorema 3.3. (Kõing 1931). ***En una gráfica bipartita el número de arcos en un acoplamiento máximo es igual al número de vértices en una cubierta mínima.***

Demostración: Sea G una gráfica bipartita con bipartición (X,Y) y sea M^* un acoplamiento máximo de G. Denotemos por U el conjunto de vértices no saturados por M^* en X, y sea Z el conjunto de todos los vértices conectados por cadenas M^*-alternantes a vértices de U. Los conjuntos $S = Z \cap X$ y $T = Z \cap Y$. Entonces tenemos que todos los vértices en T son saturados por M^* y $N(S) = T$. Definimos $K = (X \backslash S) \cup T$. Todos los arcos de G deben tener al menos uno de sus extremos en K. Por otra parte, había un arco con un extremo en S y otro en $Y \backslash T$, contradiciendo $N(S) = T$. Por lo tanto K es una cubierta de G y tenemos que $|M^*| = |K|$. Por el lema 3.1., K es una cubierta mínima.

■

Definición 3.14. La componente de una gráfica es impar o par de acuerdo al número impar o par de vértices que tiene. Denotamos por $o(G)$ el número de componentes impares de G.

Una condición necesaria y suficiente para que una gráfica tenga un acoplamiento perfecto fue obtenida por Tutte y se presenta en el siguiente teorema.

<u>Teorema 3.4. (Tutte 1947).</u> G tiene un acoplamiento perfecto si y sólo si $o(G - S) \leq |S|$. $\forall\ S \subset V$.

<u>Demostración:</u>

<u>"⇒"</u> Supongamos primero que G tiene un acoplamiento perfecto M. Sea S un subconjunto propio de V, y sean G_1 , G_2 , ..., G_n *las componentes impares de G-S. Porque* G_i *es impar, algún vértice* u_i *de* G_i *es impar, entonces algún vértice* u_i *de* G_i *debe estar acoplado bajo M con un vértice* v_i *de S. (Ilustración 3.15.)*

Por lo tanto, como $\{v_1 , v_2 , ..., v_n \} \subseteq S$,
$o(G- S) = n = N| \{v_1 , v_2 , ..., v_n \}| \leq |S|$.

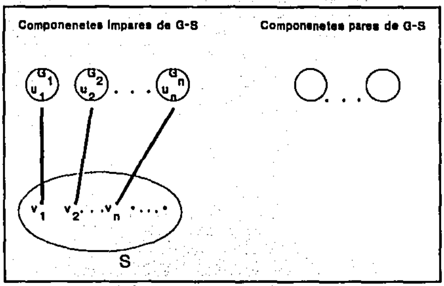

Ilustración 3.15.

<u>"⇐"</u> Supongamos que G satisface $|N(S)| \geq |S|$, *pero que no contiene un acoplamiento perfecto. Entonces G es una subgráfica extendida de una gráfica* G^* *que no tiene un acoplamiento perfecto. Dado que G-S es una subgráfica extendida de* G^*-S *tenemos que* $o(G^* - S) \leq o(G - S)$ *y por* $o(G - S) \leq |S|$

$\therefore\ o(G^* - S) \leq |S|$ *para toda* $S \subset V(G^*)$.

En particular, conviniendo $S = \emptyset$, *vemos que* $O(G^*) = 0$ *y así* $V(G^*)$ *es par.*

Denótese por U el conjunto de vértices de grado v-1 en G^*. *Como* G^* *claramente tiene un acoplamiento perfecto U = V, podemos* suponer que $G^* - U$ es una unión disjunta de gráficas completas. Supóngase, por el contrario, que alguna componente $G^* - U$ no es completa. Entonces, en esta componente existen vértices x, y y z tales que $xy \in E(G^*)$, $yz \in E(G^*)$ y $xz \notin E(G^*)$. Sin embargo, como $y \notin U$, existe un vértice w en $G^* - U$ tal que $yw \notin E(G^*)$. Tal situación se muestra en la ilustración 3.16.

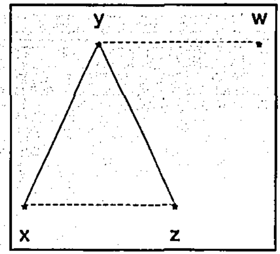

Ilustración 3.16.

Como G^* es una gráfica máxima que no contiene un acoplamiento perfecto, $G^* + e$ tiene un acoplamiento perfecto para todo $e \notin E(G^*)$. Sean M_1 y M_2 acoplamientos perfectos en $G^* + xz$ y $G^* + yw$, respectivamente, y denótese por H la subgráfica de $G^* \cup \{xz, yw\}$ inducida por $M_1 \Delta M_2$. Como cada vértice de H tiene grado 2, H es una unión disjunta de ciclos. Todos estos ciclos son pares, dado que arcos de M_1 alternan con arcos de M_2 a lo largo de ellos. Distinguimos dos casos:

Caso 1: xz y yw están en componentes diferentes de H. Entonces si yw está en el ciclo C de H, los arcos de M_1 en C junto con los arcos de M_2 que no están en C, constituyen un acoplamiento perfecto en G^*, contradiciendo la definición de G^*. (Ilustración 3.17.)

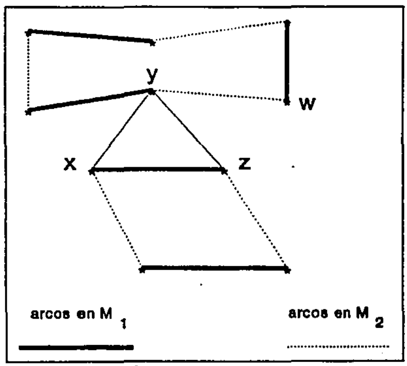

Ilustración 3.17.

Caso 2: xz y yw están en la misma componente C de H. Por simetría de x y z, suponemos que los vértices x, y, w y z están en tal orden en C. Entonces los arcos de M_1 en la sección yw...z de C, junto con los arcos yz y los arcos que no están en M_2 en la sección yw...z de C constituyen un acoplamiento perfecto en G^*, otra vez contradiciendo la definición de G^*. (Ilustración 3.18.).

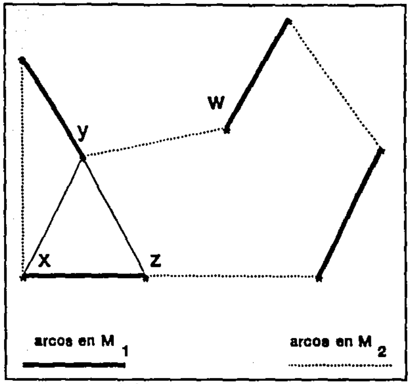

Ilustración 3.18.

Como en ambos casos, 1 y 2, conducen a contradicciones y se obtiene que G^* - U está efectivamente en una unión disjunta de gráficas completas.

Como $o(G^* - S) \leq |S|$, entonces $o(G^* - U) \leq |U|$. Así a lo más $|U|$ de las componentes de G^* - U son impares. Pero entonces G^* tiene un acoplamiento perfecto: un vértice en cada componente impar de G^* - U está acoplado con un vértice de U; los vértices sobrantes en U y en componentes de G^* - U son entonces acoplados como se muestra en la ilustración 3.19. Como se supuso que G^* no tiene un acoplamiento perfecto obtenemos la contradicción deseada. ■

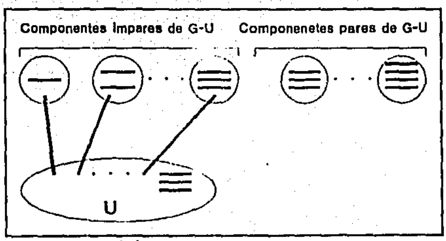

Ilustración 3.19.

Capítulo 4.

EL PROBLEMA DE ASIGNACION DE PERSONAL ENCONTRANDO UN ACOPLAMIENTO PERFECTO.

Como ya se ha mencionado anteriormente, el objetivo de este trabajo es resolver el problema de asignación. Recordamos que el problema de asignación consiste en: asignar n trabajadores x_1, x_2, ..., x_n a n trabajos y_1, y_2, ..., y_n. Cada trabajador está capacitado para realizar uno o más de estos trabajos.

Construimos una gráfica bipartita G con bipartición (X,Y), donde $X = \{ x_1, x_2, \ldots, x_n \}$, $Y = \{ y_1, y_2, \ldots, y_n \}$. El arco $x_iy_j \in G$ si y sólo si el trabajador x_i está capacitado para realizar el *trabajo y_j. El problema se convierte en determinar si en G existe un acoplamiento perfecto, si ésto no es posible entonces se debe encontrar un acoplamiento de cardinalidad máxima.*

Presentaremos un algoritmo que encuentra un acoplamiento G que satura todos los vértices X o que encuentra un subconjunto S de X tal que $|N(S)| < |S|$.

*La idea básica del algoritmo es iniciar con un acoplamiento arbitrario M. Si M satura todos los vértices en X, entonces el acoplamiento es del tipo requerido. Si no, se elige un vértice u $\in$ X no saturado por M y se busca una cadena M-aumentante con origen en u. Este método encuentra tal cadena P si existe, en este caso **M** = M Δ E(P) es un acoplamiento más grande que M, y por lo tanto satura más vértices en X. Este procedimiento se repite hasta que tal cadena no existe, entonces S = Z $\cap$ X y se satisface $|N(S)| < |S|$.*

Definición 4.1. Las siguientes tres definiciones de árbol no dirigido, ilustración 4.1., son equivalentes:

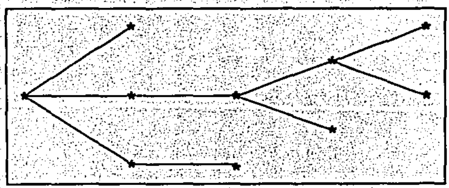

Ilustración 4.1.

4.1.1. Una gráfica conectada de n vértices y n - 1 arcos.

4.1.2. Una gráfica conectada sin circuitos.

4.1.3. Una gráfica en la que cada par de vértices son conectados por una única cadena elemental.

Definición 4.2. Sea M un acoplamiento en G, y sea u un vértice no saturado por M en X. Un árbol H $\subseteq$ G se llama árbol alternante con raíz u si i.) u $\in$ V(H) y ii.) para todo vértice en H, la cadena (u,v) en H es una cadena M-alternante.

La búsqueda para una cadena M-aumentante con origen u requiere del crecimiento de un árbol alternante con raíz en u. Este procedimiento fue sugerido por Edmonds (1965). Inicialmente, H consiste de exactamente u. El árbol entonces crece de forma tal que, sucede:

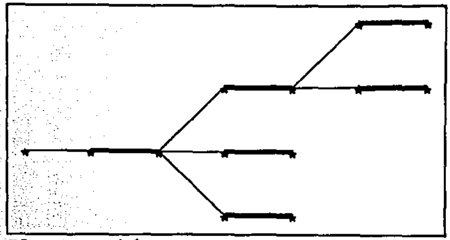

Ilustración 4.2.

i.) Todos los vértices de H excepto u son saturados y acoplados por M. (Ilustración 4.3.a.)

ii.) H contiene un vértice no saturado por M diferente de u. (Ilustración 4.3.b.)

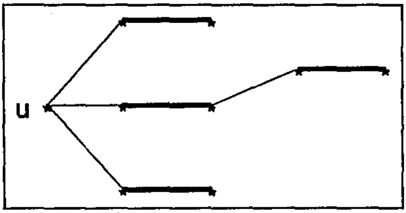

Ilustración 4.3.a.

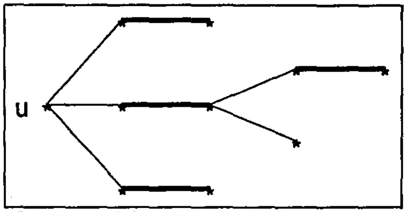

Ilustración 4.3.b.

Si se da el caso i.) entonces, sean $S = V(H) \cap X$ *y* $V(H) \cap Y$, *tenemos que* $N(S) \neq T$; *así* $N(S) = T$ *ó* $N(S) \supset T$.

a.) Si $N(S) = T$, *entonces como los vértices en* $S \setminus \{u\}$ *están acoplados con los vértices en T,* $|N(S)| = |S| - 1$, *indicando que G no tiene todos los vértices saturados en X.*

b.) Si $N(S) \supset T$, *existe un vértice* $y \in Y \setminus T$ *adyacente al vértice x en S. Como todos los vértices de H excepto u están acoplados por M, x = u ó x está acoplado con un vértice de H. Entonces* $xy \notin M$. *Si y está saturado por M, con* $yz \in M$, *h crece añadiendo los vértices y y z y los arcos yz y xz (Ilustración 4.4.a). Retrocediendo así al caso i.). Si y no está saturado por*

M, H crece añadiendo el vértice y y el arco xy, resultando el caso ii.). La cadena (u,y) de H es entonces una cadena M-aumentante con origen u (Ilustración 4.4.b.).

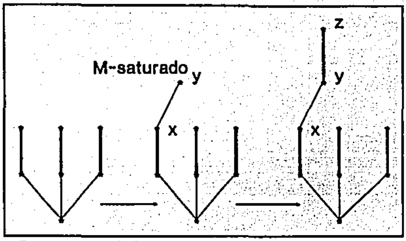

Ilustración 4.4.a.

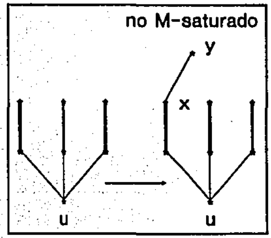

Ilustración 4.4.b.

*El algoritmo antes descrito se conoce como el **Método Húngaro** y puede resumirse de la siguiente manera:*

ALGORITMO DE ACOPLAMIENTO BIPARTITA O METODO HUNGARO.

Propósito: *Determinar un acoplamiento perfecto en una gráfica bipartita.*

Descripción:

PASO 0: Iniciar con un acoplamiento arbitrario M.

*PASO 1: Si M satura todos los vértices en X, **ALTO**, un acoplamiento perfecto ha sido encontrado. En otro caso, sea $u \in X$ un vértice no saturado por M. Sean $S = \{u\}$, $T = \emptyset$ y $P = \{u\}$.*

PASO 2: Si $N(S) = T$ entonces $|N(S)| < |S|$, como $|T| = |S| - 1$ ALTO, por el teorema de Hall, no existe un acoplamiento que sature todo vértice en X. De otra forma, sea $y \in N(S) \setminus T$, $P = P \cap \{y\}$.

PASO 3: Si y es saturado por M, sea $yz \in M$. $S = S \cup \{z\}$, $T = T \cup \{y\}$ y $P = P \cup \{z\}$ y vaya al paso 2. (Observe que $|T| = |S| - 1$ y se mantiene después del reemplazo). En otro caso, sea P *una cadena M-aumentante (u,y). Reemplace M por M' = M⊊E(P) y vaya al paso 1.*

EJEMPLO 4.1. Supongamos que tenemos 5 hombres x_1, x_2, x_3, x_4 y x_5 a los que les vamos a signar una tarea y_1, y_2, y_3, y_4 y y_5. Cuál es la mejor asignación si:

x_1 sólo puede hacer las tareas y_1, y_2 y y_5.
x_2 sólo puede hacer las tareas y_1 y y_3.
x_3 sólo puede hacer las tareas y_3 y y_4.
x_4 sólo puede hacer las tareas y_1 y y_5.
x_5 sólo puede hacer las tareas y_3 y y_5.

Este problema lo podemos ver como un problema de acoplamiento en una gráfica bipartita de la siguiente manera:

Determinar un acoplamiento máximo en la gráfica con bipartición (X,Y), donde X = { x_1, x_2, x_3, x_4, x_5 } y Y ={ y_1, y_2, y_3, y_4, y_5 }. (Ilustración 4.5.)

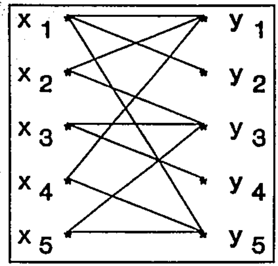

Ilustración 4.5.

PASO 0: Iniciamos con un acoplamiento arbitrario M = { (x_1, y_1), (x_3, y_3), (x_5, y_5)} (Ilustración 4.6.)

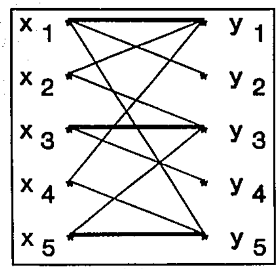

Ilustración 4.6.

ITERACION 1:

PASO 1: Como M no satura todos los vértices en X, sea $u = x_2 \in X$ *un vértice no saturado.* $S = \{ x_2 \}$, $T = \emptyset$ *y* $P = \{ x_2 \}$.

PASO 2: $N(S) = \{ y_1 , y_3 \}$, $T \subset N(S)$. *Sea* $y_1 \in N(S) - T$, $P = \{ x_2, y_1 \}$.

PASO 3: Como y_1 *es saturado por M,* $x_1 y_1 \in M$. $S = \{ x_2 , x_1 \}$, $T = \{ y_1 \}$, $P = \{ x_2, y_1 , x_1 \}$.

PASO 2: $N(S) = \{ y_1 , y_3 , y_2 \}$, $T \subset N(S)$. *Sea* $y_2 \in N(S) - T$, $P = \{ x_2, y_1 , x_1 , y_2 \}$.

PASO 3: Como y_2 *no es saturado por M, una cadena aumentante ha sido encontrada. (Ilustración 4.7.a.). La nueva cadena alternante es la que se muestra en la ilustración 4.7.b.*

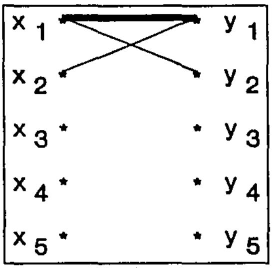

Ilustración 4.7.a.

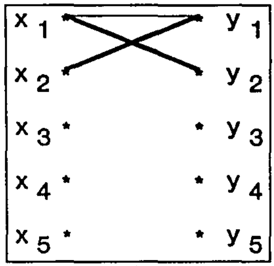

Ilustración 4.7.b

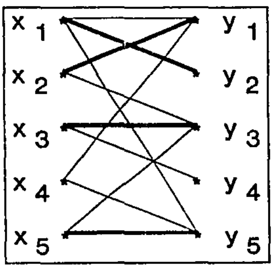

Ilustración 4.8.

El nuevo acoplamiento ahora es $M = \{ x_1 y_2 , x_2 y_1 , x_3 y_3 , x_5 y_5 \}$. *(Ilustración 4.8.)*

ITERACION 2:

PASO 1: Como M no satura todos los vértices en X, sea u = x_4 $\in X$ *un vértice no saturado.* $S = \{ x_4 \}$, $T = \emptyset$ *y* $P = \{ x_4 \}$.

PASO 2: $N(S) = \{ y_1 , y_5 \}$, $T \subset N(S)$. *Sea* $y_1 \in N(S) - T$, $P = \{ x_4, y_1 \}$.

PASO 3: Como y_1 *es saturado por M,* $x_2 y_1 \in M$. $S = \{ x_4 , x_2 \}$, $T = \{ y_1 \}$, $P = \{ x_4, y_1 , x_2 \}$.

PASO 2: $N(S) = \{ y_1 , y_5 , y_3 \}$, $T \subset N(S)$. *Sea* $y_3 \in N(S) - T$, $P = \{ x_4, y_1 , x_2 , y_3 \}$.

PASO 3: Como y_3 *es saturado por M,* $x_3 y_3 \in M$. $S = \{ x_4 , x_2 , x_3\}$, $T = \{ y_1 , y_3 \}$, $P = \{ x_4, y_1 , x_2 , y_3 , x_3 \}$.

PASO 2: $N(S) = \{ y_1 , y_5 , y_3 , y_4\}$, $T \subset N(S)$. *Sea* $y_4 \in N(S) - T$, $P = \{ x_4, y_1 , x_2 , y_3 , x_2 , y_4\}$.

PASO 3: Como y_4 *no es saturado por M, una cadena aumentante ha sido encontrada. (Ilustración 4.9.a.). La nueva cadena alternante es la que se muestra en la ilustración 4.9.b.*

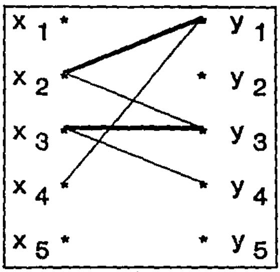

Ilustración 4.9.a.

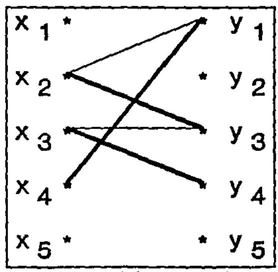

Ilustración 4.9.b.

El nuevo acoplamiento ahora es $M = \{ x_1 y_2 , x_2 y_3 , x_3 y_4 , x_4 y_1 , x_5 y_5 \}$. *(Ilustración 4.10.)*

ITERACION 3:

PASO 1: Como M satura todos los vértices en X, un acoplamiento perfecto ha sido encontrado.

a x_1 *se le asigna el trabajo* y_2
a x_2 *se le asigna el trabajo* y_3
a x_3 *se le asigna el trabajo* y_4
a x_4 *se le asigna el trabajo* y_1
a x_5 *se le asigna el trabajo* y_5

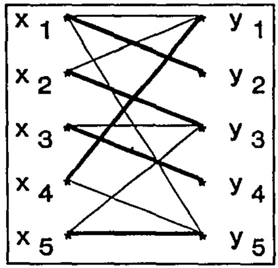

Ilustración 4.10.

Capítulo 5.

EL PROBLEMA DE ASIGNACION OPTIMA.

Consideremos ahora el problema de asignación pero tomando en cuanta la eficiencia de los trabajadores en varios trabajos (capacidad posiblemente, la ganancia de la compañía). En este caso, nos interesa una asignación en la que la efectividad de los trabajadores sea máxima. El problema de encontrar tal asignación se conoce como el ***PROBLEMA DE ASIGNACION OPTIMA.***

Considérese una gráfica bipartita completa pesada con bipartición (X,Y), donde $X = \{ x_1 , x_2 , \ldots , x_n \}$ y $Y = \{ y_1 , y_2 , \ldots , y_n \}$ y el arco $x_i y_j$ tiene peso $w_{ij} = w(x_i y_j)$ es la *eficiencia del trabajador x_i en el trabajo y_j. El problema de asignación óptima es equivalente a encontrar un acoplamiento perfecto en esta gráfica con pesos, a tal acoplamiento le llamaremos **acoplamiento óptimo**.*

Para resolver el problema de asignación es posible enumerar los $n!$ acoplamientos perfectos y encontrar un óptimo entre ellos. Sin embargo, para una n grande, tal procedimiento sería ineficiente. Presentaremos aquí un algoritmo bueno para encontrar un acoplamiento óptimo en una gráfica bipartita completa con pesos.

*Definición 5.1. Un **vértice etiquetado factible** es una función con valor real en el conjunto de vértices $X \cup Y$ tal que para toda $x \in X$ y $y \in Y$:*

$$l(x) + l(y) \geq w(xy)$$

*(El número real $l(v)$ se llama **etiqueta del vértice v**). Un vértice factible etiquetado es así una etiqueta del vértice, tal que la suma de las etiquetas de los dos extremos de un arco es al menos tan grande como el peso de los arcos. No importa cual sea el peso de los arcos, ya que siempre existe un vértice factible etiquetado, así la función es dada por:*

$$l(x) = \max w(xy) \quad \text{si } x \in X \qquad (1)$$
$$l(y) = 0 \qquad \text{si } y \in Y \qquad (2)$$

Si l es un vértice etiquetado, denotamos por :

$$E_l = \{ xy \in E \mid l(x) + L(y) = w(xy) \}$$

La subgráfica extendida de G con arcos en el conjunto E_l se refiere como a la subgráfica igualdad correspondiente al vértice factible etiquetado l, y se denota por G_l. La conección entre subgráficas iguales y acoplamientos óptimos se prueba en el siguiente teorema.

Teorema 5.1 Sea l un vértice factible etiquetado de G. Si G_l contiene un acoplamiento perfecto M^*, entonces M^* es un acoplamiento óptimo de G.

Demostración:

Supóngase que G_l contiene un acoplamiento perfecto M^*. Como G_l es una subgráfica extendida de G, M^* es cualquier acoplamiento perfecto de G, entonces

$$w(M) = \sum w(e) = \sum_{e \in M} l(v) \qquad v \in V \quad (i)$$

como cada $e \in M$ pertenece a la subgráfica extendida de G, M^* es cualquier acoplamiento perfecto de G, entonces

$$w(M) = \sum w(e) \leq \sum_{e \in M} l(v) \qquad v \in V \qquad (ii)$$

Se tiene por i.) y ii.) que $w(M^*) \geq w(M)$ así M^* es un acoplamiento óptimo.

■

El teorema 5.1. es la base de un algoritmo dado por Kuhn (1955) y Munkres (1957), para encontrar un acoplamiento óptimo en una gráfica bipartita completa con pesos.

Iniciando con un vértice factible arbitrario etiquetado l, por ejemplo utilizando (1) y (2) determinamos G_l, elegimos un acoplamiento arbitrario M en G_l y aplicamos el Método Húngaro. Si un acoplamiento perfecto se encuentra en G_l, entonces por el teorema 5.1. este acoplamiento es óptimo. En otro caso el método Húngaro termina con un acoplamiento M que no es perfecto. Modificamos entonces l a un vértice factible etiquetado l' con la propiedad de que tanto M' como H están contenidos en G_1, . Tales modificaciones en el vértice factible etiquetado se hacen tan pronto como sea necesario, hasta que un acoplamiento perfecto se encuentra en alguna subgráfica igualdad.

ALGORITMO DE KUHN-MUNKRES.

Propósito: *Encontrar un acoplamiento óptimo en una gráfica bipartita completa pesada.*

Descripción:

PASO 0: Iniciar con un vértice factible etiquetado l, determinar G_1, elegir un acoplamiento arbitrario M en G_1.

PASO 1: Si X es saturado por M, entonces M es un acoplamiento perfecto (como $|X| = |Y|$) entonces por el teorema 5.1. es también un acoplamiento óptimo; en este caso ALTO. En otro caso sea u un vértice no saturado por M. Sea $S = \{u\}$ y $T = \emptyset$.

PASO 2: Si $N_{G1}(S) \supset T$, ir al paso 3. En otro caso $N_{G1}(S) = T$. Calcular

$$\alpha_i = \min_{x \in S\ y \notin T} \{l(x) + l(y) - w(xy)\}$$

PASO 3: Elegir un vértice y en $N_{G1}(S) \setminus T$. Como en el procedimiento del método Húngaro, considerar si es saturado por M o no. Si y es saturado por M, con $yz \in M$, reemplazar S por $S \cup \{z\}$ y T por $T \cup \{y\}$, e ir al paso 2. En otro caso, sea P una cadena M-aumentante (u,y) en G_{G1}, reemplazar M por $M = M \vartriangle e(P)$ e ir al paso 1.

Ejemplo 5.1. Encontrar un acoplamiento óptimo para resolver el problema de asignación si la matriz de eficiencia es:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	17	8	12	9	14	6
x_2	15	13	18	15	10	4
x_3	14	16	8	12	5	9
x_4	18	7	14	9	11	13
x_5	7	16	11	14	6	10
x_6	18	9	13	5	15	7

PASO 0: Se calculan $l(x_i)$ *y* $l(y_j)$ $\forall\ i,j = 1,\ldots,6$ *donde*

$$l(x_i) = \max_{y \in Y} w(xy) \quad si\ x \in X$$
$$l(y_j) = 0 \quad si\ y \in Y$$

$l(x_1) = \max\ \{17, 8, 12, 9, 14, 6\} = 17 = w\ (x_1, y_1)$
$l(x_2) = \max\ \{15, 13, 18, 15, 10, 4\} = 18 = w\ (x_2, y_3)$
$l(x_3) = \max\ \{14, 16, 8, 12, 5, 9\} = 16 = w\ (x_3, y_2)$
$l(x_4) = \max\ \{18, 7, 14, 9, 11, 13\} = 18 = w\ (x_4, y_1)$
$l(x_5) = \max\ \{7, 16, 11, 14, 6, 10\} = 16 = w\ (x_5, y_2)$
$l(x_6) = \max\ \{18, 9, 13, 5, 15, 7\} = 18 = w\ (x_6, y_1)$
$l(y_j) = 0 \quad \forall\ j = 1,\ldots,6$

La subgráfica G_1 *asociada es la que se muestra en la ilustración 5.1.a., mientras el acoplamiento arbitrario se muestra en la ilustración 5.1.b.*

ITERACION 1:

PASO 1: *X no está M-saturado. Sea* $u = x_4$ *un vértice expuesto.* $S = \{ x_4 \}$, $P = \{ x_4 \}$ *y* $T = \emptyset$.

PASO 2: $N_{G1}(S) = \{ y_1 \}$, $T \subset N_{G1}(S)$

PASO 3: $N_{G1}(S) \backslash T = \{ y_1 \}$, y_1 *es M-saturado* $y_1 x_1 \in M$.
$\therefore\ S = S \cup \{ x_1 \} = \{ x_4, x_1 \}$, $T = T \cup \{ y_1 \} = \{ y_1 \}$,
$P = \{ x_4, y_1, x_1 \}$

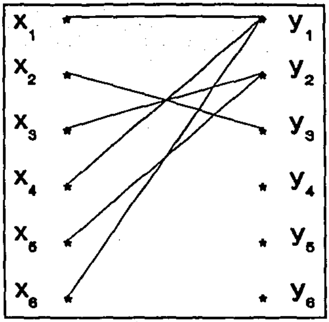

Ilustración 5.1.a

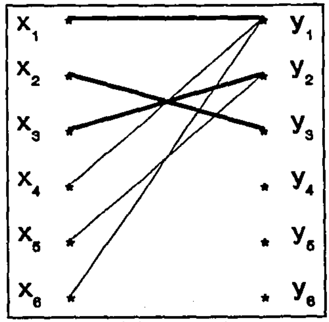

Ilustración 5.1.b.

PASO 2: $N_{G1}(S) = \{ y_1 \}$, $T = N_{G1}(S)$. *Se calcula*

$$\alpha_1 = \min_{x \in S\, y \notin T} \{l(x) + l(y) - x(xy)\}$$

$\alpha_1 = \{ (18+0-7) = 11,\ (18+0-4) = 4,\ (18+0-9) = 9,\ (18+0-11) = 7,\ (18+0-13) = 5,\ (17+0-8) = 9,\ (17+0-12) = 5,\ (17+0-9) = 8,\ (17+0-14) = 3,\ (17+0-14) = 3,\ (17+0-6) = 11 \} = 3$

Se actualizan las etiquetas utilizando la siguiente regla:

$l(v) - \alpha_1$	*si* $v \in S$
$l(v) + \alpha_1$	*si* $v \in T$
$l(v)$	*en otro caso*

$l(x_1) = 17 - 3 = 14 = w(x_1\ y_5)$

$l(x_2) = 18$

$l(x_3) = 16$

$l(x_4) = 18 - 3 = 15$

$l(x_5) = 16$

$l(x_6) = 18$

$l(y_1) = 0 + 3 = 3$

$l(y_j) = 0$

$j = 2, \ldots, 6.$

La nueva subgráfica es la que se muestra en la ilustración 5.2.

$G_1 = G_1 \cup (x_1 \; y_5)$.

$N_{G1}(S) = \{ y_1, y_5 \}$, $T \subset N_{G1}(S)$

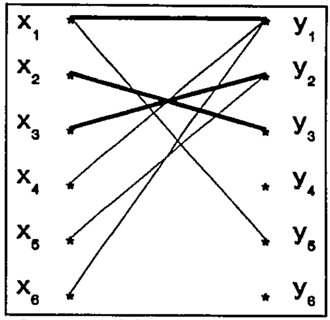

Ilustración 5.2.

PASO 3: $N_{G1}(S) \setminus T \{ y_5 \}$, y_5 *no es M-saturado.*
$\therefore$ *una cadena M-aumentante ha sido encontrada* $P = \{ x_4, y_1, x_1, y_5 \}$ *(Ilustración 5.3.a) . La nueva cadena M-alternante se muestra en la ilustración 5.3.b.*

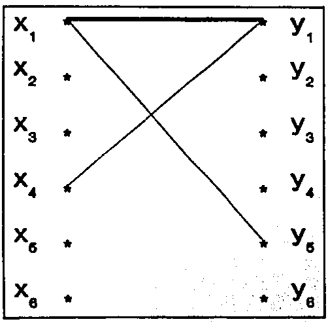

Ilustración 5.3.a.

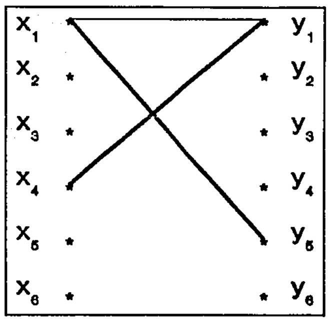

Ilustración 5.3.b.

Por lo tanto el nuevo acoplamiento en G_1 *se muestra en la Ilustración 5.4.*

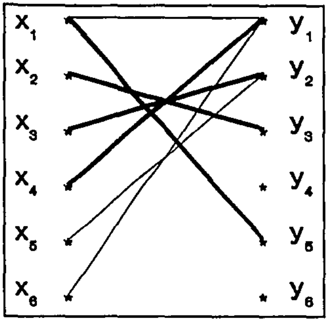

Ilustración 5.4.

ITERACION 2:

PASO 1: X no está M-saturado. Sea $u = x_5$ un vértice expuesto. $S = \{ x_5 \}$, $P = \{ x_5 \}$ y $T = \varnothing$.

PASO 2: $N_{G1}(S) = \{ y_2 \}$, $T \subset N_{G1}(S)$

PASO 3: $N_{G1}(S) \setminus T = \{ y_2 \}$, y_2 es M-saturado $y_2 \; x_3 \in M \therefore$ $S = S \cup \{ x_5 \} = \{ x_5 , x_3 \}$, $T = T \cup \{ y_2 \} = \{ y_2 \}$, $P = \{ x_5 , y_2 , x_3 \}$

PASO 2: $N_{G1}(S) = \{ y_2 \}$, $T = N_{G1}(S)$. Se calcula

$$\alpha_1 = \min_{x \in S \; y \notin T} \{ l(x) + l(y) - x(xy) \}$$

$$\alpha_1 = \{ (16+3-7) = 12, (16+0-11) = 2, (16+0-6) = 10, (16+0-10) = 6, (16+3-14) = 5, (16+0-8) = 8, (16+0-12) = 4, (16+0-5) = 11, (16+0-9) = 7, \} = 2$$

Se actualizan las etiquetas utilizando la siguiente regla:

$l(v) - \alpha_1$	si $v \in S$
$l(v) + \alpha_1$	si $v \in T$
$l(v)$	en otro caso

$l(x_1) = 14$ $\qquad$ $l(y_1) = 3$

$l(x_2) = 18$ $\qquad$ $l(y_2) = 0 + 2 = 2$

$l(x_3) = 16 - 2 = 14 = w(x_3\ y_1)$

$l(x_4) = 15$ $\qquad$ $l(y_j) = 0$

$l(x_5) = 16 - 2 = 14 = w(x_5\ y_4)$ $\qquad$ $j = 3, \ldots 6.$

$l(x_6) = 18$

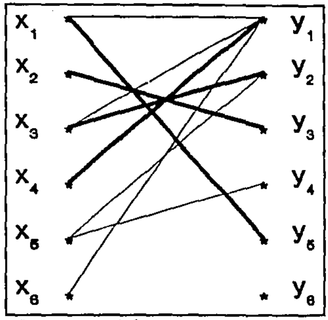

Ilustración 5.4.

La nueva subgráfica es la que se muestra en la ilustración 5.4.

$G_l = G_l \cup (x_3\ y_1) \cup (x_5\ y_4).$

$N_{Gl}(S) = \{y_2, y_4, y_1\},$

$T \subset N_{Gl}(S)$

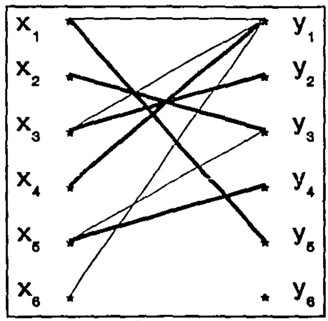

Ilustración 5.5.

PASO 3: $N_{Gl}(S)\backslash T = \{y_4, y_1\}$

y_4 *no es M-saturado y como es vecino de* x_5 *entonces podemos acoplar* x_5 *con* y_4*, ya que* x_5 *es un vértice expuesto, ignorando la cadena P que se estaba construyendo. El nuevo acoplamiento se muestra en la ilustración 5.5.*

ITERACION 3:

PASO 1: X no está M-saturado. Sea $u = x_6$ *un vértice expuesto.* $S = \{x_6\}$, $P = \{x_6\}$ *y* $T = \emptyset$.

PASO 2: $N_{Gl}(S) = \{y_1\}$, $T \subset N_{Gl}(S)$

PASO 3: $N_{G1}(S)\backslash T = \{ y_1 \}$, y_1 es M-saturado $y_1 x_4 \in M$ $\therefore$
$S = S \cup \{ x_4 \} = \{ x_6, x_4 \}$, $T = T \cup \{ y_1 \} = \{ y_1 \}$,
$P = \{ x_6, y_1, x_4 \}$

PASO 2: $N_{G1}(S) = \{ y_1 \}$, $T = N_{G1}(S)$. Se calcula

$$\alpha_1 = \min_{x \in S,\, y \notin T} \{ l(x) + l(y) - x(xy) \}$$

$\alpha_1 = \{ (18+2-9) = 11,\ (18+0-13) = 5,\ (18+0-15) = 3,\ (18+0-7) = 11,\ (15+2-7) = 10,\ (15+0-14) = 1,\ (15+0-9) = 6,\ (15+0-11) = 4,\ (15+0-13) = 2 \} = 1$

Se actualizan las etiquetas utilizando la siguiente regla:

$$\begin{array}{ll} l(v) - \alpha_1 & \text{si } v \in S \\ l(v) + \alpha_1 & \text{si } v \in T \\ l(v) & \text{en otro caso} \end{array}$$

$l(x_1) = 14$ $\quad$ $l(y_1) = 3 + 1 = 4$
$l(x_2) = 18$ $\quad$ $l(y_2) = 2$
$l(x_3) = 14$
$l(x_4) = 15 - 1 = w(x_4\, y_3)$ $\quad$ $l(y_j) = 0$
$l(x_5) = 14$ $\quad$ $j = 3, \ldots 6.$
$l(x_6) = 18 - 1 = 17$

La nueva subgráfica es la que se muestra en la ilustración 5.6. $G_1 = G_1 \cup (x_4\, y_3)$

$N_{G1}(S) = \{ y_1, y_3 \}$, $T \subset N_{G1}(S)$

PASO 3: $N_{G1}(S)\backslash T = \{ y_3 \}$, y_3 es M-saturado $y_3 x_2 \in M$ $\therefore$
$S = S \cup \{ x_2 \} = \{ x_6, x_4, x_2 \}$, $T = T \cup \{ y_3 \} = \{ y_1, y_3 \}$, $P = \{ x_6, y_1, x_4, y_3, x_2 \}$

PASO 2: $N_{G1}(S) = \{ y_1, y_3 \}$, $T = N_{G1}(S)$.

Se calcula

$$\alpha_1 = \min_{x \in S\ y \notin T} \{l(x) + l(y) - x(xy)\}$$

$\alpha_1 = \{ (17+2-9) = 10,\ (18+0-5) = 12,\ (18+0-7) = 10,\ (14+2-7) = 9,\ (14+0-9) = 5,\ (14+0-11) = 3,\ (14+0-13) = 1,\ (18+2-13) = 7,\ (18+0-15) = 3,\ (18+02-10) = 8,\ (18+0-4) = 14 \} = 1$

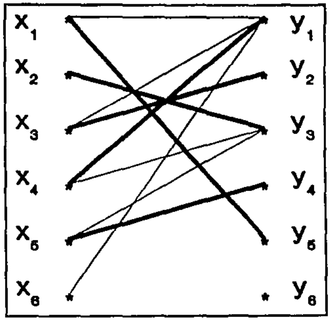

Ilustración 5.6.

Se actualizan las etiquetas utilizando la siguiente regla:

$$\begin{array}{ll} l(v) - \alpha_1 & \text{si } v \in S \\ l(v) + \alpha_1 & \text{si } v \in T \\ l(v) & \text{en otro caso} \end{array}$$

$l(x_1) = 14$

$l(x_2) = 18 - 1 = 17$

$l(x_3) = 14$

$l(x_4) = 14 - 1 = 13 = w(x_4\ y_6)$

$l(x_5) = 14$

$l(x_6) = 17 - 1 = 16$

$l(y_1) = 4 + 1 = 5$

$l(y_2) = 2$

$l(y_3) = 1$

$l(y_j) = 0$

$j = 4, \ldots 6.$

$G_1 = G_1 \cup (x_4\ y_6)$. Ver Ilustración 5.7.

$N_{G1}(S) = \{ y_1, y_6, y_3 \}$,

$T \subset N_{G1}(S)$

PASO 3: $N_{G1}(S) \setminus T\ \{ y_6 \}$, y_6 *no es M-saturado.*

∴ una cadena M-aumentante ha sido encontrada $P = \{ x_6 , y_1 , x_4 , y_3 , x_2 , y_6 \}$.

Pera el arco $(x_2 \; y_6) \nexists$ *en* G_1, *ya que* y_6 *sólo es vecino de* x_4. *Como* x_4 *si está en la cadena, entonces podemos modificarla de la siguiente forma* $P = \{ x_6 , y_1 , x_4 , y_6 \}$. *La cadena aumentante se encuentra en la ilustración 5.8.a. y la nueva cadena alternante en la ilustración 5.8.b.*

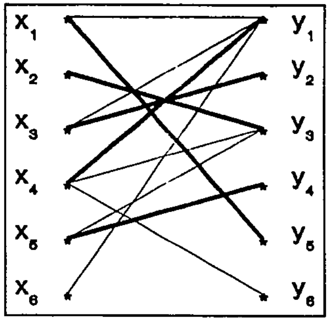

Ilustración 5.7.

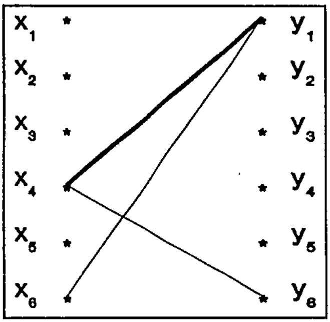

Ilustración 5.8.a.

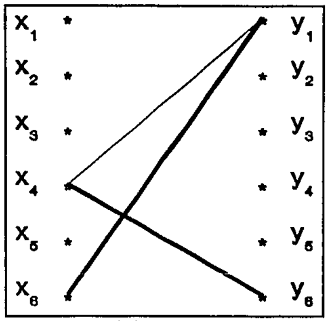

Ilustración 5.8.b.

Y el nuevo acoplamiento se muestra en la ilustración 5.9.

ITERACION 4:

PASO 1: X es M-saturado, entonces M es un acoplamiento perfecto y óptimo.

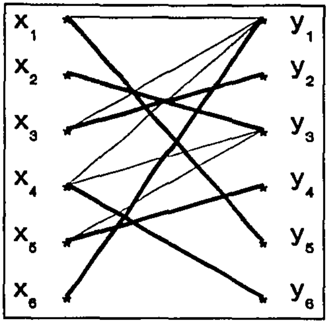

Ilustración 5.9.

Por lo tanto a x_1 se le asigna el trabajo y_5
a x_2 se le asigna el trabajo y_3
a x_3 se le asigna el trabajo y_2
a x_4 se le asigna el trabajo y_6
a x_5 se le asigna el trabajo y_4
a x_6 se le asigna el trabajo y_1

Ahora presentaremos un algoritmo para determinar un algoritmo para determinar un acoplamiento máximo pero, de peso mínimo.

Este algoritmo consiste de dos fases. La primera fase, determina la subgráfica factible mínima ; mientras que la segunda fase determina el acoplamiento máximo de peso mínimo con la subgráfica factible mínima.

ALGORITMO DE DOS FASES.

Propósito: *Determinar un acoplamiento máximo de peso mínimo, en una gráfica bipartita completa pesada.*

Descripción:

Fase 1: *Considere la matriz de eficiencia de la gráfica bipartita. Un arco se considera factible sólo cuando se satisface la relación $w_i + w_j \geq l_{ij}$, donde l_{ij} es el peso del arco (i,j) en la matriz de eficiencia.*

PASO 1: Determine el elemento más pequeño l_{ij} en el renglón i. Haga $w_i = \min(l_{ij})$. Repita este paso para todos los renglones.

PASO 2: Determine el valor más pequeño de los valores $(l_{ij} - w_i)$ en todas las columnas. Haga $w_j = \min(l_{ij} - w_i)$. Repita este paso para todas las columnas.

PASO 3: Identifique todos los arcos para los cuales $l_{ij} = w_i + w_j$. Incluya estos arcos en la subgráfica G'.

Fase 2:

PASO 1: Determine la subgráfica factible mínima G' usando la fase 1.

PASO 2: Forme la matriz de adyacencia S para la subgráfica factible mínima G'.

PASO 3: Verifique en todo renglón de S si alguna columna tiene una adyacencia (entrada diferente de cero); si existe, elimine todas las otras adyacencias en esa columna.

PASO 4: Verifique en toda columna de S si algún renglón tiene una adyacencia; si existe elimine al otras adyacencias en ese renglón.

PASO 5: Verifique el número de adyacencias en cada renglón y en cada columna.

a. Si cada renglón tiene exactamente una adyacencia, incluya todas estas adyacencias en el conjunto del acoplamiento y PARE.

b. Si en todo renglón y columna hay al menos una adyacencia y existe un subconjunto de renglones y columnas con más de una *adyacencia, entonces seleccione arbitrariamente una adyacencia en un renglón (o columna), borre las demás adyacencias en tal renglón (o columna) y regrese al paso 3.*

c. Si existe algún renglón (o columna sin adyacencias, marque el renglón (o columna) y vaya al paso 6.

PASO 6: Regrese a la matriz original del paso 2. Inicie con un vértice correspondiente al renglón (o columna) marcado, trace una cadena conexa.

PASO 7: Reduzca los pesos de todos los vértices renglón que no pertenezcan a la cadena por 1, e incremente los pesos de todos los vértices columna que pertenezcan a la cadena por 1.

PASO 8: Determine una nueva subgráfica factible y regrese al paso 2.

Ejemplo 5.2. Determinar el acoplamiento máximo de peso mínimo para la matriz de eficiencia:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	17	8	12	9	14	6
x_2	15	13	18	15	10	4
x_3	14	16	8	12	5	9
x_4	18	7	14	9	11	13
x_5	7	16	11	14	6	10
x_6	18	9	13	5	15	7

FASE I:

PASO 1: Se determina el elemento más pequeño l_{ij} en el renglón i y hacemos $w_i = \min_j l_{ij}$.

$w_1 = 6$, $w_2 = 4$, $w_3 = 5$, $w_4 = 7$, $w_5 = 6$, $w_6 = 5$.

Se resta w_i al renglón i, obteniendo la matriz con elementos $l_{ij} - w_i$.

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	11	2	6	3	8	0
x_2	11	9	14	11	6	0
x_3	9	11	3	7	0	4
x_4	11	0	7	2	4	6
x_5	1	10	5	8	0	4
x_6	13	4	8	0	10	2

PASO 2: Determinamos el más pequeño de los valores $l_{ij} - w_i$, en toda la columna j y hacemos $v_j = \min_i (l_{ij} - w_i)$
$v_1 = 1$, $v_2 = 0$, $v_3 = 3$, $v_4 = 0$, $v_5 = 0$, $v_6 = 0$.

Se resta v_j a cada columna y se obtiene la matriz:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	10	2	3	3	8	0
x_2	10	9	11	11	6	0
x_3	8	11	0	7	0	4
x_4	10	0	4	2	4	6
x_5	0	10	2	8	0	4
x_6	12	4	5	0	10	2

PASO 3: Los arcos que cumplen con la relación $l_{ij} = w_i + v_j$ son: $x_1 y_6$, $x_2 y_6$, $x_3 y_3$, $x_3 y_5$, $x_4 y_2$, $x_5 y_1$, $x_5 y_5$, $x_6 y_4$.

FASE II:

PASO 1 y PASO 2: La subgráfica factible mínima G' tiene como matriz de adyacencia:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	0	0	0	0	0	1
x_2	0	0	0	0	0	1
x_3	0	0	1	0	1	0
x_4	0	1	0	0	0	0
x_5	1	0	0	0	1	0
x_6	0	0	0	1	0	0

PASO 3: En el renglón 1 existe la adyacencia simple a_{16} quitamos la adyacencia a_{26} de la columna 6.

En el renglón 6 existe la adyacencia simple a_{64} .

PASO 4: En la columna 1 existe la adyacencia simple a_{15} quitamos la adyacencia a_{55} del renglón 5.

En la columna 2 existe la adyacencia simple a_{42}.

En la columna 3 existe la adyacencia simple a_{33} quitamos la adyacencia a_{35} del renglón 5.

PASO 5: c. No existe adyacencia en el renglón 2, marcamos el renglón 2.

PASO 6: Con la matriz de adyacencias original formamos una cadena que inicie en el vértice x_2 .

$$P = \{ x_2 , y_6 , x_1 \}$$

PASO 7: Cambiamos las w_i y las v_j utilizando la siguiente relación:

$$\begin{array}{lll} w_i = w_i - 1 & & si\ x_i \in P \\ v_j = v_j + 1 & & si\ y_j \in P \\ v & & en\ otro\ caso \end{array}$$

$w_1 = 5,\ w_2 = 3,\ w_3 = 5,\ w_4 = 7,\ w_5 = 6,\ w_6 = 5.$

$v_1 = 1,\ v_2 = 0,\ v_3 = 3,\ v_4 = 0,\ v_5 = 0,\ v_6 = 1.$

PASO 8: $l_{12} = w_1 + v_2 = W(x_1 y_2) = 8.$

En la subgráfica G' agregamos el arco $x_1 y_2$, es decir en la matriz de adyacencias $a_{12} = 1$.

PASO 2: La nueva matriz de adyacencias es:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	0	1	0	0	0	1
x_2	0	0	0	0	0	1
x_3	0	0	1	0	1	0
x_4	0	1	0	0	0	0
x_5	1	0	0	0	1	0
x_6	0	0	0	1	0	0

PASO 3: En el renglón 2 existe la adyacencia simple a_{26} quitamos la adyacencia a_{16} de la columna 6.

En el renglón 1 existe la adyacencia simple a_{12} quitamos la adyacencia a_{42} de la columna 2.

En el renglón 6 existe la adyacencia simple a_{64}.

***PASO 4:** En la columna 1 existe la adyacencia simple a_{15} quitamos la adyacencia a_{55} del renglón 5.*

En la columna 3 existe la adyacencia simple a_{33} quitamos la adyacencia a_{35} del renglón 5.

***PASO 5c:** La columna 5 no tiene adyacencias. Marcamos la columna 5.*

***PASO 6:** Con la matriz de adyacencias asociada a la subgráfica formamos una cadena que inicie en y_5.*

$P = \{ y_5 , x_5 , y_1 \}$.

***PASO 7:** Cambiamos las w_i y las v_j utilizando la siguiente relación:*

$$
\begin{array}{lll}
w_i = w_i - 1 & & \text{si } x_i \in P \\
v_j = v_j + 1 & & \text{si } y_j \in P \\
v & & \text{en otro caso}
\end{array}
$$

$w_1 = 5,\ w_2 = 3,\ w_3 = 5,\ w_4 = 7,\ w_5 = 5,\ w_6 = 5.$

$v_1 = 2,\ v_2 = 0,\ v_3 = 3,\ v_4 = 0,\ v_5 = 1,\ v_6 = 1.$

***PASO 8:** La subgráfica factible es la misma.*

***PASO 2:** La matriz de adyacencias asociada a la subgráfica es la misma.*

***PASO 6:** Se considera la misma cadena: $P = \{ y_5 , x_5 , y_1 \}$.*

***PASO 7:** Cambiamos las w_i y las v_j utilizando la siguiente relación:*

$$
\begin{array}{lll}
w_i = w_i - 1 & & \text{si } x_i \in P \\
v_j = v_j + 1 & & \text{si } y_j \in P \\
v & & \text{en otro caso.}
\end{array}
$$

$w_1 = 5,\ w_2 = 3,\ w_3 = 5,\ w_4 = 7,\ w_5 = 4,\ w_6 = 5.$

$v_1 = 3,\ v_2 = 0,\ v_3 = 3,\ v_4 = 0,\ v_5 = 2,\ v_6 = 1.$

***PASO 8:** La subgráfica factible es la misma.*

***PASO 2:** La matriz de adyacencias asociada a la subgráfica es la misma.*

***PASO 6:** Se considera la misma cadena: $P = \{ y_5 , x_5 , y_1 \}$.*

***PASO 7:** Cambiamos las w_i y las v_j utilizando la siguiente* relación:

$$w_i = w_i - 1 \qquad \text{si } x_i \in P$$
$$v_j = v_j + 1 \qquad \text{si } y_j \in P$$
$$v \qquad \text{en otro caso}$$

$w_1 = 5,\ w_2 = 3,\ w_3 = 5,\ w_4 = 7,\ w_5 = 3,\ w_6 = 5.$

$v_1 = 4,\ v_2 = 0,\ v_3 = 3,\ v_4 = 0,\ v_5 = 3,\ v_6 = 1.$

PASO 8: La subgráfica factible es la misma.

PASO 2: La matriz de adyacencias asociada a la subgráfica es la misma.

PASO 6: Se considera la misma cadena: $P = \{ y_5, x_5, y_1 \}$.

PASO 7: Cambiamos las w_i y las v_j utilizando la siguiente relación:

$$w_i = w_i - 1 \qquad \text{si } x_i \in P$$
$$v_j = v_j + 1 \qquad \text{si } y_j \in P$$
$$v \qquad \text{en otro caso}$$

$w_1 = 5,\ w_2 = 3,\ w_3 = 5,\ w_4 = 7,\ w_5 = 2,\ w_6 = 5.$

$v_1 = 5,\ v_2 = 0,\ v_3 = 3,\ v_4 = 0,\ v_5 = 4,\ v_6 = 1.$

PASO 8: $l_{45} = w_4 + v_5 = W(x_4\, y_5) = 8.$

En la subgráfica G′ agregamos el arco $x_4\, y_5$, es decir en la matriz de adyacencias $a_{45} = 1$.

PASO 2: La nueva matriz de adyacencias es:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	0	1	0	0	0	1
x_2	0	0	0	0	0	1
x_3	0	0	1	0	1	0
x_4	0	1	0	0	1	0
x_5	1	0	0	0	1	0
x_6	0	0	0	1	0	0

PASO 3: En el renglón 2 existe la adyacencia simple a_{26} quitamos la adyacencia a_{16} de la columna 6.

En el renglón 1 existe la adyacencia simple a_{12} quitamos la adyacencia a_{42} de la columna 2.

En el renglón 1 existe la adyacencia simple a_{64}.

PASO 4: En la columna 1 existe la adyacencia simple a_{51} quitamos la adyacencia a_{55} del renglón 5.

En la columna 3 existe la adyacencia simple a_{33} quitamos la adyacencia a_{35} del renglón 5.

En la columna 3 existe la adyacencia simple a_{33}.

PASO 5a: En cada renglón y columna hay exactamente una adyacencia. Estas adyacencias son el acoplamiento buscado ALTO.

El vértice x_1 lo acoplamos con el vértice y_2.
El vértice x_2 lo acoplamos con el vértice y_6.
El vértice x_3 lo acoplamos con el vértice y_3.
El vértice x_4 lo acoplamos con el vértice y_5.
El vértice x_5 lo acoplamos con el vértice y_1.
El vértice x_6 lo acoplamos con el vértice y_4.

El peso del acoplamiento es 43.

Conclusiones.

Como se ha dicho en esta tesis, el problema de asignación es un problema en el que se presenta la necesidad de asignar hombres a trabajos, de manera tal que exista una correspondencia biunívoca; por lo tanto la matriz de eficiencia debe de ser cuadrada y los elementos de ésta deben de ser enteros positivos.

El problema de asignación se supone lineal, y las técnicas propuestas son: ramificación y acotamiento, método húngaro, método de Kuhn-Munkres y método de las dos fases. Siendo los más eficientes son el método húngaro y el método de Kuhn-Munkres, ya que su complejidad es $O(n^3)$, presentados en los dos útimos capítulos.

Para los métodos húngaro y de Kuhn-Munkres se dan los listados de los programas fuente editados en Turbo-Pascal, y están basados en tales métodos. En el programa de Kuhn-Munkres se consideran los casos en los que:

1) *existe una cadena aumentante,*
2) *existe una cadena aumentante, pero ésta no se utiliza totalmente, ya que de hacerlo así se entra en un bucle y por tanto, nunca se logra la solución del problema,*
3) *existe una cadena aumentante y ésta no es factible.*

Estos dos últimos casos no están contemplados en los algoritmos (teoría), ya que por ser un problema gráfico éstos son fáciles de detectar.

Hay matrices de eficiencia, en las que sus elementos provocan que los programas no funcionen, ello debido a que no tienen las características necesarias para abordarse en forma gráfica y por tanto es recomendable utilizar el método de ramificación y acotamiento.

Para el problema de asignación no existe una cota superior, en lo que se refiere al orden de la matriz, pero para los programas si.

Considero que debido a que el problema se está abordando gráficamente no existen programas que resuelvan el problema de asignación por esta vía.

Apéndice 1.

```
UNIT CAPTURA;
INTERFACE
uses crt;
const dim = 100;
type bin = 0..1;
     vector = array [1..dim] of bin;
     matriz = array [1..dim] of vector;
     conj_compa = SET OF byte;

Procedure Pregunta(var preg : boolean);

Function Checa_Valor (cad : string; valmin, valmax : integer) : boolean;

Procedure Mensaje;

Procedure LeeEntero(posx, posy, valmin, valmax : integer;
               var numero : bin; var renglon : byte; var columna
               : byte);

Procedure LeeMatriz( var matr : matriz; dimen :  byte; ind : byte);

IMPLEMENTATION

Procedure pregunta(var preg : boolean);
 var cch : char;
  begin
    gotoxy(32,2);
    write ('¿ESTA BIEN ESCRITA LA MATRIZ? ([S] / [N]) ');
    repeat
      cch:= Upcase(readKey);
    until cch in ['S','N'];
    if cch = 'S' then preg:= true
    else preg:= false;
    gotoxy(32,2);
    write ('                              ');
  end;

Function checa_valor (cad : string; valmin, valmax : integer) : boolean;
 var nn, num : integer;
  begin
    val(cad,num,nn);
    checa_valor:= TRUE;
    If nn = 0 Then
     Begin
      if num < valmin Then checa_valor:= FALSE;
      if num > valmax Then checa_valor:= FALSE
     End
    Else checa_valor:= FALSE;
  end;

Procedure mensaje;
 var cch : char;
  begin
    write(#7);
    gotoxy(10,24);
    write('VALOR FUERA DE RANGO, PARA CONTINUAR
          OPRIMA CUALQUIER TECLA');
    cch:= readkey;
    gotoxy(10,24);
    write('                                        ');
  end;

Procedure LeeEntero(posx, posy, valmin, valmax : integer;
               var numero : bin; var renglon : byte;
               var columna : byte);
 var cad : string;
     cch : char;
     long : integer;
  begin
    str(numero,cad);
    long:= length(cad);
    if long > 3 then cad:= copy(cad,1,3);
    gotoxy(posx,posy);
    write('   ');
    gotoxy(posx,posy);
    write(cad);
    repeat
      cch:= readkey;
      case Ord(cch) of
       48..57: begin
               if long < 3 then
                begin
                  inc(long);
                  cad:= cad + cch;
                  gotoxy(posx,posy); write (cad);
                  if long = 3 then cch:= chr(13);
                end;
              end;
       8     : if long > 0 then
               begin
                 dec(long);
                 cad:= copy(cad,1,long);
```

```
            gotoxy(posx + long, posy);
            write(' ');
          end
        else write(#7);
  13    : begin
          if long = 0 then mensaje
          else
           begin
             if checa_valor(cad,valmin,valmax)
             then inc(columna)
             else
               begin
                 mensaje;
                 cch:= ' ';
               end;
           end;
        end;
  0     : begin
          cch:= readkey;
          if ((long >0) and  (checa_valor(cad,valmin,valmax)))
           then
            case cch of
              #72 : dec(renglon);
              #80 : inc(renglon);
              #77 : inc(columna);
              #75 : dec(columna);
          end;
          cch:= chr(13);
        end;
  end;
 until ord(cch) = 13; {hasta que teclee return}
 val(cad,numero,long);
end;

Procedure LeeMatriz( var matr : matriz; dimen : byte; ind : byte);
 var nn, valmin,  valmax  : integer;
     posx, posy, columna, renglon, cont : byte;
     termina : boolean;
 begin
   valmin:= 0;
   if ind = 0 then valmax:= 1 else valmax:= 999;
   for cont:= 1 to dimen do
    for nn:= 1 to dimen do
      matr[cont,nn]:= 0; {llenado por renglón}
   clrscr; posx:= 5; posy:= 3;
   for cont:= 1 to dimen do
     begin
       gotoxy(posx,posy); write('y',cont);
       inc(posx,4);
     end;
   posx:= 1; posy:=4;
   for cont:= 1 to dimen do
     begin
       gotoxy(posx,posy); write('x',cont);
       inc(posy);
     end;
   posx:= 5; posy:= 4;
   clrscr;
   for cont:= 1 to dimen do {escribo matriz}
    begin
      posx:= 5;
      for nn:= 1 to dimen do
       begin
         gotoxy(posx,posy);
         write(matr[cont,nn]);
         inc(posx,4);
       end;
      inc(posy);
   end;
   termina:= false;
   columna:= 1; renglon:= 1;
   repeat
    posx:= 1 + columna*4;
    posy:= 3 + renglon;
    LeeEntero(posx,posy,valmin,valmax, matr[renglon; columna],
              renglon,columna);
    if columna > dimen then
      begin
       columna:= 1; inc(renglon)
      end;
    if renglon > dimen then
     begin
      pregunta(termina);
      renglon:= 1;
     end;
    if renglon < 1  then renglon:= dimen;
    If columna < 1 Then
     begin
      columna:= dimen; dec(renglon)
     end
   until termina;
 end;
END.
```

```
PROGRAM HUNGARO; (*Caso sin costos*)

uses crt, CAPTURA;

Procedure Presentacion;
  Begin
    GotoXY(3,10);  TextMode(c40);  TextColor(Blue);
    TextBackground(LightGray);
    ClrScr;  GotoXY(8,13);
    Write('ALGORITMO HUNGARO');
    TextColor(Blue+Blink);
    GotoXY(1,25);
    Write('Oprima RETURN para continuar...');
    ReadLn;
    ClrScr;
    TextMode(c80); TextColor(Yellow); TextBackground(Blue);
    ClrScr;
    GotoXY(3,10);
    Write('Este programa resuelve el problema de ACOPLAMIENTO
          MAXIMO en');
    Write(' una gráfica ');
    GotoXY(41,11);Write('————————————');GotoXY(26,14);
    Write('BIPARTITA '); GotoXY(26,15); Write('————————');
    GotoXY(1,25);
    Write('Oprima RETURN para continuar...');
    ReadLn; ClrScr; GotoXY(3,3);
    Write('Para poder utilizar este programa, se puede introducir la
          MATRIZ');
    Write(' de'); GotoXY(3,5);
```

```
Write('ADYACENCIA desde el teclado o por medio de un archivo en
     código ASCII.');
GotoXY(3,8);
Write('a) Si es por ARCHIVO, el formato del archivo es el siguiente;
     el pri-');
GotoXY(6,10);
Write('mer renglón del archivo debe contener sólo un número. Dicho
     número ');
GotoXY(6,12);
Write('representa el orden de una matriz cuadrada (NxN). El resto del
     ar-');
GotoXY(6,14);
Write('chivo debe contener "N" renglones con"N" números cada uno,
     separa-');
GotoXY(6,16);
Write('dos por un espacio entre sí.');
GotoXY(3,19);
Write('b) Si es por TECLADO, el programa guiará al usuario en la
     introduc-');
GotoXY(6,21);
Write('ción de los valores necesarios para el planteamiento del
     problema.');
GotoXY(1,24);
TextColor(Yellow+Blink);
Write('Nota: El ORDEN MAXIMO de la matriz de adyacencia debe
     ser 100, (N ≤ 100).');
TextColor(Yellow);
GotoXY(1,25);
Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr;
GotoXY(3,10);
Write('Si al introducir los datos por el teclado se comete(n) algun(os)
     erro');
Write('r(es),');
GotoXY(3,12);
Write('se debe continuar introduciendo los datos que falten. El
     programa pregunta');
Write('rá');
GotoXY(3,14);
Write('si se desean hacer modificaciones, y se modificará lo que el
     usuario le in-');
GotoXY(3,16);
Write('dique al programa.');
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr;
GotoXY(3,10);
Write('Se informa al usuario que si el problema es de N > "15", las
     matrices ');
GotoXY(3,12);
Write('asociadas al problema no se reportarán en las iteraciones
     intermedias,');
GotoXY(3,14);
Write('solamente en el reporte de los resultados finales. Ello debido al
     limi-');
GotoXY(3,16);
Write('tado espacio en la pantalla. ');
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr;
GotoXY(15,3);
TextColor(Yellow+Blink);
Write('NOTACION UTILIZADA EN ESTE PROGRAMA:');
GotoXY(15,4);
Write('———— ———— — ——— ————');
TextColor(Yellow);
GotoXY(3,7);
Write('"S"   denota el subconjunto de los vértices NO SATURADOS
     del conjunto X.');
GotoXY(3,9);
Write('"N(S)" denota a los VECINOS de S.');
GotoXY(3,11);
Write('"P"   denota a la CADENA ALTERNANTE o
     AUMENTANTE.');
GotoXY(3,13);
Write('"T"   es subconjunto de N(S) y es subconjunto de P.');
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
End;

Procedure Aborta( tm : byte );
 Var
  cc : char;
  ij : integer;
 Begin
  TextMode(c40);
  TextColor(Yellow+Blink);
  TextBackground(Blue);
  ClrScr;
  GotoXY(1,12);
  Write('¡Se cancela la ejecución del programa!');
  GotoXY(10,15);
  Write('¡  N = ',tm,' > 100  !');
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  cc:= #0;
  While cc <> #13 do
   begin
        For ij:= 1 to 2 do Write(#7);
        readLn(cc);
   end;
  TextMode(c80); ClrScr;
  Halt
 End;

Procedure  Leer_Matriz_Adyacencia( var nn : matriz; var tm : byte );
 var i,j : word;
x,y,rpan : byte;
    c : char;
   arch: TEXT;
   nomb: string;
  Begin
   clrscr;
   GotoXY(1,13);
   Write('¿Deseas dar la matriz por teclado o por archivo?  (T/A)   ');
   x:= WhereX; y:= WhereY;
   Repeat
    GotoXY(x,y);
    Readln(c)
   Until c in ['A','a','T','t'];
   case c of
    'A', 'a': begin
              GotoXY(1,18);
                Write('Dame el nombre del archivo y su trayectoria :
                 ');
              Readln(nomb);
                assign(arch,nomb);
                reset(arch);
                readln(arch,tm);
              If tm > 100 Then Aborta(tm);
                 for i:= 1 to tm do
                   for j:= 1 to tm do
                     read(arch,nn[i,j]);
                        close(arch)
```

```pascal
        end;
  'T', 't': begin
          GotoXY(8,17);
            Write('Dame el tamaño de la matriz  ( a lo más
              100x100)    ');
            Readln(tm);
          If tm > 100 Then Aborta(tm);
          LeeMatriz(nn,tm,0);
            end
end;
If tm <= 15 Then
 Begin
  ClrScr;
  GotoXY(15,1);
  Write('La matriz de ADYACENCIA es:');
  GotoXY(7,3);
  rpan:= 3;
  For j := 1 to tm do Write('y',j,'  ');  rpan:= 4; Writeln;
  For i := 1 to tm do
    begin
      Write(' x',i:2);
      GotoXY(5,rpan);
      For j := 1 to tm do Write(nn[i,j]:4);
      Writeln;
      rpan: = rpan + 1
    end;
  GotoXY(3,23);
  Write('¿Desea hacer algun(os) cambio(s) en la matriz? (s/n)  ');
  x:= WhereX;  y:= WhereY;
  Repeat
    GotoXY(x,y);
    Read(c)
  Until c in ['S','s','N','n'];
  If c in ['S','s'] Then
    Begin
      Repeat
        ClrScr;
        GotoXY(15,10);
        Writeln('¿Qué elemento quiere cambiar?');
        GotoXY(15,15);
        Write('Renglón = ? ');  Read(i);
        GotoXY(15,18);
        Write('Columna = ? ');  ReadLn(j);
        GotoXY(20,22);
        Write('Nuevo valor de la entrada ',i:3,',',j:3,' = ?  ');
        Read(nn[i,j]);
        ClrScr;
        GotoXY(10,15);
        Write('¿Desea hacer algún otro cambio en la matriz? (s/n)  ');
        x:= WhereX;  y:= WhereY;
        c := ' ';
        While NOT (c in ['S','s','N','n']) do
          Begin
            GotoXY(x,y);
            c := ReadKey;
          End
      Until c in ['N','n'];
      ClrScr;
      GotoXY(15,1);
      Write('La matriz de ADYACENCIA es:');
      GotoXY(7,2);
      rpan:= 2;
      For j := 1 to tm do Write('y',j,'  ');  Writeln; rpan:= 3;
      For i := 1 to tm do
        begin
          Write(' x',i:2);
          GotoXY(5,rpan);
          For j := 1 to tm do Write(nn[i,j]:4);
          Writeln; rpan:= rpan + 1
        end;
      GotoXY(1,25); Write('Oprima RETURN para continuar...');
      ReadLn(c);
      ClrScr
    End
 End
End;

Procedure    Inicia_Mat_Acopla( var m1 : matriz; tatam : byte );
 var i,j : byte;
  Begin
    For i:= 1 to tatam do
      For j:= 1 to tatam do m1[i,j] := 0;
  End;


Procedure  Inic_Vec_Aco( var vec : vector; long : byte );
 var i : byte;
  Begin
   For i:= 1 to long do vec[i] := 0
  End;

Procedure Paso_0( m_adya : matriz; var m_acop : matriz; t1 : byte;
                  var vacox : vector );
 var i,j : byte;
   vvx,vvy: vector;
  Begin
    For i:= 1 to t1 do vvx[i] := 0;
    For j:= 1 to t1 do vvy[j] := 0;

    For i:= 1 to t1 do
     For j:= 1 to t1 do
      If (m_adya[i,j] = 1) AND (vvx[i] = 0) AND (vvy[j] = 0) then
         begin
           m_acop[i,j] := 1;
           vvx[i] := 1;
           vvy[j] := 1;
           j:= t1
         end;
    For i:= 1 to t1 do
     If vvx[i] = 1 then vacox[i] := 1
  End;

Function Paso1A( vecto : vector; longi : byte ): byte;
 var i,aux : byte;
  Begin
    i:= 1; aux := 0;
    While i <= longi do
     If vecto[i] = 0
      Then
       begin
        aux := i;
        i := longi + 1
       end
      Else i := i + 1;
    If aux = 0 Then Paso1A := 0
     Else Paso1A := aux
  End;

Function  CardiS( con_S : vector; tal : byte ): byte;
 var i : byte;
  Begin
    i := 1;
    While  (con_S[i] <> 0) AND (i <= tal) do i := i + 1;
    If i > tal Then CardiS := tal
```

```
   Else CardiS := i - 1
  End;

Procedure Los_Vecinos_de_S( var NdeS : vector; conj_S : vector;
                        mat_ad : matriz;   ttam : byte );
 var cardS, i, j, k : byte;
    cardN : byte;
  Begin
   cardS := CardiS(conj_S,ttam);
   cardN := CardiS(NdeS,ttam);
   cardN := cardN + 1;
   For i:= 1 to cardS do
    For j := 1 to ttam do
     If mat_ad[conj_S[i],j] = 1
      Then
       begin
        k:= 1;
        While (NdeS[k] < > j) AND (k < = (cardN - 1)) do
          k := k + 1;
        If k = cardN Then
          begin
           NdeS[cardN] := j;
           cardN := cardN + 1
          end
       end
  End;


Function  Compara_Conjuntos( conj1, conj2 : vector; taam : byte):
                        Boolean;
 var i, car1, car2 : byte;
   con1, con2 : conj_compa;
  Begin
   car1 := CardiS(conj1,taam);  car2 := CardiS(conj2,taam);
   If car1 = car2 Then
     begin
       con1 := []; con2 := [];
       For i := 1 to car1 do
         begin
           con1 := con1 + [conj1[i]];
           con2 := con2 + [conj2[i]]
         end;
       If con1 = con2 then Compara_Conjuntos := TRUE
       Else Compara_Conjuntos := FALSE
     end
    Else  Compara_Conjuntos := FALSE
  End;

Procedure   Unir_Conj(var reunion : vector;  eleme, es : byte );
 var card, i : byte;
  Begin
   card := CardiS(reunion,es);
   i:= 1;
   While (reunion[i] < > eleme) AND (i < = card) do i := i + 1;
   If i > card Then reunion[i] := eleme
  End;

Procedure   Unir_Traye(var reunion : vector;  eleme, es : byte );
 var card : byte;
  Begin
   card := CardiS(reunion,es);
   reunion[card+1] := eleme
  End;

Procedure  FormarNS_T( NdeSS,TT : vector ;  var DifNT : conj_compa;
                  tam1 : byte);
 var i, carNS, carT : byte;
    conNS, conT : conj_compa;
  Begin
   carNS := CardiS(NdeSS,tam1);
   carT := CardiS(TT,tam1);
   conNS := [];  conT := [];
   For i := 1 to carNS do conNS := conNS + [NdeSS[i]];
   For i := 1 to carT do conT := conT + [TT[i]];
   DifNT := conNS - conT
  End;

Function  Buscar_Col( ren, inic : byte; mmaat : matriz; siz1 : byte ): byte;
 var ii, aux : byte;
  Begin
   ii := inic; aux := 0;
   While ii < = siz1 do
    begin
     if mmaat[ren,ii] = 0 Then ii := ii + 1
     Else
      begin
       aux := ii;
       ii := siz1 + 1
      end
    end;
   Buscar_Col := aux
  End;

Function  Buscar_Ren( col, prim : byte; maatt : matriz; ziz : byte ): byte;
 var ii, aux : byte;
  Begin
   ii := prim; aux := 0;
   While ii < = ziz do
    begin
     if maatt[ii,col] = 0 Then ii := ii + 1
     Else
      begin
       aux := ii;
       ii := ziz + 1
      end
    end;
   Buscar_Ren := aux
  End;

Procedure  Acoplar_P_M_Aco( cadena : vector; var acoplamiento :
                     matriz; esp : byte);
 var carCad, jj : byte;
  Begin
   carCad := CardiS(cadena,esp);   jj := 1;
   While jj < = carCad do
    begin
     acoplamiento[cadena[jj],cadena[jj+1]] := 1;   jj := jj + 2
    end
  End;

Procedure  Acoplar_ady_M_Aco( adyac : matriz; var acopla : matriz;
                    tan : byte);
 var i,j : byte;
    vvx,vvy: vector;
  Begin
   For i:= 1 to tan do vvx[i] := 0;
   For j:= 1 to tan do vvy[j] := 0;
   For i := 1 to tan do
    For j := 1 to tan do
     vvx[i] := vvx[i] + acopla[i,j];
   For j := 1 to tan do
    For i := 1 to tan do
     vvy[j] := vvy[j] + acopla[i,j];
   For i:= 1 to tan do
```

```
    If vvx[i] = 0 Then
      For j:= 1 to tan do
        If (adyac[i,j] = 1) AND (vvy[j] = 0) Then
            begin
              acopla[i,j] := 1;
            vvx[i] := 1;
            vvy[j] := 1;
              j:= tan
            end;
  End;

Function Actuali_VecA( aacop : matriz; var vec : vector;
                      lon1 : byte ): byte;
 var k, l, auxi : byte;
  Begin
    For k := 1 to lon1 do vec[k] := 0;
    For k := 1 to lon1 do
      For l := 1 to lon1 do
        If aacop[k,l] = 1 Then
          begin
            vec[k] := 1;
            l := lon1
          end;
    auxi := 0;
    For k := 1 to lon1 do
      If vec[k] = 0 Then
      begin
        auxi := k;
        k := lon1
      end;
    If auxi = 0 Then Actuali_VecA := 0
    Else Actuali_VecA := auxi
  End;

Procedure Salida(var acco : matriz; TH : boolean; ttam : byte);
 var c, d, rp : byte;
     cc : char;
  Begin
     clrscr;
     If TH = TRUE Then
       Begin
         TextMode(c40); TextColor(Yellow); TextBackground(Blue);
         ClrScr;    GotoXY(3,10);
         Write('El ACOPLAMIENTO NO es MAXIMO (Teorema ');
         GotoXY(3,12);
         Write(' de Hall); los vértices saturados se ');
         GotoXY(3,14);
         Write(' localizan en la siguiente matriz de ');
         GotoXY(15,16);
         Write('acoplamiento:');
         For c:= 1 to 2 do Write(#7)
       End
     Else
       Begin
         TextMode(c40); TextColor(Yellow); TextBackground(Blue);
         ClrScr; GotoXY(8,10);
         Write('El ACOPLAMIENTO es MAXIMO;');
         GotoXY(6,14);
         Write('la matriz de acoplamiento es:')
       End;
     GotoXY(1,25); Write('Oprima RETURN para continuar...');
     Read(cc);
     TextMode(c80);
     If TH = FALSE Then For c:= 1 to 2 do Write(#7);
     TextColor(Yellow); TextBackground(Blue);
     ClrScr; GotoXY(20,1);
     Write(' ╔══════════════════════════════════╗ ');
     GotoXY(20,2);
     Write(' ║ LA MATRIZ DE ACOPLAMIENTO ES: ║ ');
     GotoXY(20,3);
     Write(' ╚══════════════════════════════════╝ ');
     GotoXY(8,5); rp:= 5;
     For d := 1 to ttam do Write('y',d,'  ');  Writeln; rp:= 6;
     For c := 1 to ttam do
      begin
        Write(' x',c:2);    GotoXY(5,rp);
          For d := 1 to ttam do  Write(acco[c,d]:4);  Writeln;
        rp:= rp +1
      end;
     Readln;
     TextMode(c80);
     ClrScr
  End;


VAR mat_ady, mat_aco : matriz;
  tam, u, xi, yj, Repa, Co, Re : byte;
  vec_acox, No_Saturado, Vecinos, P_trayec : vector;
  c0 : CHAR;
  Vecinos_de_S: vector;
  iguales, Teo_Hall, PAS2, pertenece : boolean;
  Dife_N_T : conj_compa;

BEGIN

  Presentacion;
  Leer_Matriz_Adyacencia(mat_ady, tam);
  clrscr;
  GotoXY(1,13);
  Write('¿Desea ver TODAS las iteraciones del algoritmo?  (S/N)  ');
  Co:= WhereX; Re:= WhereY;
  Repeat
    GotoXY(Co,Re);
    Read(c0)
  Until c0 in ['S','s','N','n'];
  IF c0 IN ['s','S'] THEN
    BEgin
     Inicia_Mat_Acopla(mat_aco, tam);
     Inic_Vec_Aco(vec_acox, tam);
     Inic_Vec_Aco(No_Saturado,tam);
     Inic_Vec_Aco(Vecinos,tam);
    Inic_Vec_Aco(P_trayec,tam);
    Inic_Vec_Aco(Vecinos_de_S,tam);
    Paso_0(mat_ady, mat_aco, tam, vec_acox);
    ClrScr;   GotoXY(33,1);
    Write(' ╔═══════════╗ ');  GotoXY(33,2);
    Write(' ║  PASO 0  ║ ');  GotoXY(33,3);
    Write(' ╚═══════════╝ ');  GotoXY(3,5);   Write('S    = ϕ');
    GotoXY(3,7); Write('N(S) = ϕ'); GotoXY(3,9); Write('P    = ϕ');
    GotoXY(3,11);  Write('T    = ϕ'); GotoXY(1,25);
    Write('Oprima RETURN para continuar...');
    ReadLn;  ClrScr;    GotoXY(33,1);
    Write(' ╔═══════════╗ '); GotoXY(33,2);
    Write(' ║  PASO 0  ║ ');  GotoXY(33,3);
    Write(' ╚═══════════╝ ');
    If tam <= 15 Then
      Begin
        GotoXY(10,5);
        Write('La matriz de ADYACENCIAS asociada a la subgráfica
             es:');
        GotoXY(7,7); Repa:= 7;
        For Co := 1 to tam do Write('y',Co,'  ');  Writeln;  Repa := 8;
        For Re := 1 to tam do
          begin
```

```
      Write(' x',Re:2); GotoXY(5,Repa);
      For Co := 1 to tam do Write(mat_ady[Re,Co]:4);
      Writeln; Repa:= Repa + 1;
    end;
   GotoXY(1,25); Write('Oprima RETURN para continuar...');
  ReadLn; ClrScr; GotoXY(33,1);
  Write(' ╔══════════╗ ');  GotoXY(33,2);
  Write(' ║  PASO 0  ║ '); GotoXY(33,3);
  Write(' ╚══════════╝ ');  GotoXY(10,5);
  Write('La matriz de ACOPLAMIENTO para la subgráfica es:');
  GotoXY(7,7); Repa:= 7;
  For Co := 1 to tam do Write('y',Co,'  ');  Writeln;  Repa:= 8;
  For Re := 1 to tam do
    begin
      Write(' x',Re:2); GotoXY(5,Repa);
      For Co := 1 to tam do Write(mat_aco[Re,Co]:4);
      Writeln; Repa:= Repa + 1
    end;
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  ReadLn;   ClrScr
 End;
u:= Paso1A(vec_acox,tam);
Teo_Hall := FALSE;
While ( u > 0 ) AND (Teo_Hall = FALSE) do
  Begin
   u:= Paso1A(vec_acox,tam);
   Unir_Conj(No_Saturado,u,tam);
   Unir_Traye(P_trayec,u,tam);
   Inic_Vec_Aco(Vecinos,tam);
   ClrScr;  GotoXY(33,1);
   Write(' ╔══════════╗ '); GotoXY(33,2);
   Write(' ║  PASO 1  ║ '); GotoXY(33,3);
   Write(' ╚══════════╝ '); GotoXY(3,7);
   Repa:= 7;
   Co := CardiS(No_saturado,tam);
   Write('S    = { X',No_Saturado[1]);
   For Re:= 2 to Co do
     Begin
       If ( (Re MOD 6) = 0 ) Then
         Begin
           Repa := Repa + 2;
           GotoXY(9,Repa)
         End;
       Write(',  ','X',No_Saturado[Re])
     End;
   Write(' }.');
   GotoXY(3,Repa+2);
   Repa:= Repa + 2;
   Co := CardiS(P_trayec,tam);
   Write('P    = { X',P_trayec[1]);
   For Re:= 2 to Co do
     Begin
       If ( (Re MOD 6) = 0 ) Then
         Begin
           Repa := Repa + 2;
           GotoXY(9,Repa)
         End;
       If ( (Re MOD 2) = 0 ) Then Write(',  ','X',P_trayec[Re])
       Else Write(',  ','Y',P_trayec[Re])
     End;
   Write(' }.');
   GotoXY(3,Repa+2);
   Repa:= Repa + 2;
   Write('T    = φ');
   GotoXY(3,Repa+2);
   Repa:= Repa + 2;
   Write('N(S) = φ');
```

```
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;   ClrScr;   PAS2:= TRUE;
While PAS2 = TRUE
  begin
    Los_Vecinos_de_S(Vecinos_de_S, No_Saturado, mat_ady, tam);
    iguales:= Compara_Conjuntos(Vecinos_de_S, Vecinos, tam);
    If iguales = TRUE Then
      begin
        Teo_Hall:= TRUE;
        PAS2:= FALSE;
      end
    Else
      Begin
        FormarNS_T(Vecinos_de_S,Vecinos,Dife_N_T,tam);
        yj:= 1;
        Repeat
          yj := Buscar_Col(u,yj,mat_ady,tam);
          If yj in Dife_N_T  Then
            begin
              pertenece := TRUE;
              Unir_Traye(P_trayec,yj,tam);
              ClrScr;
              GotoXY(33,1);
              Write(' ╔══════════╗ '); GotoXY(33,2);
              Write(' ║  PASO 2  ║ '); GotoXY(33,3);
              Write(' ╚══════════╝ '); GotoXY(3,7);
              Repa:= 7; GotoXY(3,Repa+2);
              Repa:= Repa + 2;
              Co := CardiS(Vecinos_de_S,tam);
              Write('N(S) = { Y',Vecinos_de_S[1]);
              For Re:= 2 to Co do
                Begin
                  If ( (Re MOD 6) = 0 ) Then
                    Begin
                      Repa := Repa + 2;
                      GotoXY(9,Repa)
                    End;
                  Write(',  ','Y',Vecinos_de_S[Re])
                End;
              Write(' }.');
              GotoXY(3,Repa+2; Repa:= Repa + 2;
              Co := CardiS(Vecinos,tam);
              If Co > 0 Then
                Begin
                  Write('T    = { Y',Vecinos[1]);
                  For Re:= 2 to Co do
                    Begin
                      If ( (Re MOD 6) = 0 ) Then
                        Begin
                          Repa := Repa + 2;
                          GotoXY(9,Repa)
                        End;
                      Write(',  ','Y',Vecinos[Re])
                    End;
                  Write(' }.');
                End
              Else Write('T    = φ');
              GotoXY(1,25);
              Write('Oprima RETURN para continuar...');
              ReadLn;
              ClrScr;
              xi := Buscar_Ren(yj,1,mat_aco,tam);
              u := xi;
              If xi > 0 Then
                begin
                  Unir_Conj(No_Saturado,xi,tam);
                  Unir_Conj(Vecinos,yj,tam);
```

```
Unir_Traye(P_trayec,xi,tam);
ClrScr; GotoXY(33,1);
Write(' ╔══════════╗ '); GotoXY(33,2);
Write(' ║  PASO 3  ║ '); GotoXY(33,3);
Write(' ╚══════════╝ '); GotoXY(3,7);
Repa:= 7;
Co := CardiS(No_saturado,tam);
Write('S    = { X',No_Saturado[1]);
For Re:= 2 to Co do
  Begin
    If ( (Re MOD 6) = 0 ) Then
      Begin
        Repa := Repa + 2;
        GotoXY(9,Repa)
      End;
    Write(',  ','X',No_Saturado[Re])
  End;
Write(' }.');
GotoXY(3,Repa+2); Repa:= Repa + 2;
Co := CardiS(Vecinos_de_S,tam);
Write('N(S) = { Y',Vecinos_de_S[1]);
For Re:= 2 to Co do
  Begin
    If ( (Re MOD 6) = 0 ) Then
      Begin
        Repa := Repa + 2;
        GotoXY(9,Repa)
      End;
    Write(',  ','Y',Vecinos_de_S[Re])
  End;
Write(' }.');
GotoXY(3,Repa+2);
Repa:= Repa + 2;
Co := CardiS(Vecinos,tam);
If Co > 0 Then
  Begin
    Write('T    = { Y',Vecinos[1]);
    For Re:= 2 to Co do
      Begin
        If ( (Re MOD 6) = 0 ) Then
          Begin
            Repa := Repa + 2;
            GotoXY(9,Repa)
          End;
        Write(',  ','Y',Vecinos[Re])
      End;
    Write(' }.');
  End
Else Write('T    = φ');
GotoXY(3,Repa+2);
Repa:= Repa + 2;
Co := CardiS(P_trayec,tam);
Write('P    = { X',P_trayec[1]);
 For Re:= 2 to Co do
  Begin
    If ( (Re MOD 6) = 0 ) Then
      Begin
        Repa := Repa + 2;
        GotoXY(9,Repa)
      End;
    If ( (Re MOD 2) = 0 ) Then
     Write(',  ','Y',P_trayec[Re])
    Else Write(',  ','X',P_trayec[Re])
  End;
 Write(' }.');
 GotoXY(1,25);
 Write('Oprima RETURN para continuar...');
```

```
              ReadLn; ClrScr
            end
          Else
            begin
              Inicia_Mat_Acopla(mat_aco,tam);
              Acoplar_P_M_Aco(P_trayec,mat_aco,tam);
              Inic_Vec_Aco(No_Saturado,tam);
              Inic_Vec_Aco(P_trayec,tam);
              Acoplar_ady_M_Aco(mat_ady,mat_aco,tam);
              ClrScr;
              GotoXY(33,1);
              Write(' ╔══════════╗ '); GotoXY(33,2);
              Write(' ║  PASO 3  ║ '); GotoXY(33,3);
              Write(' ╚══════════╝ '); GotoXY(10,5);
              Write('La matriz de ACOPLAMIENTO para la
                    subgráfica es:');
             GotoXY(7,7);
             Repa:= 7;
             For Co := 1 to tam do Write('y',Co,'  '); Writeln;
             Repa:= 8;
             For Re := 1 to tam do
              begin
                Write(' x',Re:2);
                GotoXY(5,Repa);
                For Co := 1 to tam do Write(mat_aco[Re,Co]:4);
                Writeln; Repa:= Repa + 1
              end;
             GotoXY(1,25);
             Write('Oprima RETURN para continuar...');
             ReadLn;   ClrScr;
             u:= Actuali_VecA(mat_aco,vec_acox,tam);
             PAS2 := FALSE
           end
         end
       Else
         begin
           pertenece := FALSE;
           yj := yj + 1
         end
      Until pertenece = TRUE
    End
   end
  End
 ENd

ELSE
 Begin
  Inicia_Mat_Acopla(mat_aco, tam);
  Inic_Vec_Aco(vec_acox, tam);
  Inic_Vec_Aco(No_Saturado,tam);
  Inic_Vec_Aco(Vecinos,tam);
  Inic_Vec_Aco(P_trayec,tam);
  Inic_Vec_Aco(Vecinos_de_S,tam);
  Paso_0(mat_ady, mat_aco, tam, vec_acox);
  u:= Paso1A(vec_acox,tam);
  Teo_Hall := FALSE;
 While ( u > 0 ) AND (Teo_Hall = FALSE) do
  Begin
    u:= Paso1A(vec_acox,tam);
    Unir_Conj(No_Saturado,u,tam);
    Unir_Traye(P_trayec,u,tam);
    Inic_Vec_Aco(Vecinos,tam);
    PAS2:= TRUE;
    While PAS2 = TRUE do
     begin
      Los_Vecinos_de_S(Vecinos_de_S,No_Saturado,mat_ady,tam);
      iguales:= Compara_Conjuntos(Vecinos_de_S, Vecinos, tam);
```

```
            If iguales = TRUE Then
              begin
                Teo_Hall:= TRUE;
                PAS2:= FALSE
              end
            Else
              Begin
                FormarNS_T(Vecinos_de_S,Vecinos,Dife_N_T,tam);
                yj:= 1;
                Repeat
                  yj := Buscar_Col(u,yj,mat_ady,tam);
                  If yj in Dife_N_T  Then
                    begin
                      pertenece := TRUE;
                      Unir_Traye(P_trayec,yj,tam);
                      xi := Buscar_Ren(yj,1,mat_aco,tam);
                      u := xi;
                      If xi > 0 Then
                        begin
                          Unir_Conj(No_Saturado,xi,tam);
                          Unir_Conj(Vecinos,yj,tam);
                          Unir_Traye(P_trayec,xi,tam);
                        end
                      Else
                        begin
                          Inicia_Mat_Acopla(mat_aco,tam);
                          Acoplar_P_M_Aco(P_trayec,mat_aco,tam);
                          Inic_Vec_Aco(No_Saturado,tam);
                          Inic_Vec_Aco(P_trayec,tam);
                          Acoplar_ady_M_Aco(mat_ady,mat_aco,tam);
                          u:= Actuali_VecA(mat_aco,vec_acox,tam);
                          PAS2 := FALSE
                        end
                    end
                  Else
                    begin
                      pertenece := FALSE;
                      yj := yj + 1
                    end
                Until pertenece = TRUE
              End
          end
      End
  End;
  Salida(mat_aco,Teo_Hall,tam)
END.
```

Apéndice 2.

```
UNIT CAPTU2;
INTERFACE
uses crt;
const dim = 60;
type vect = array [1..dim] of word;
    matr = array [1..dim] of vect;

Procedure Pregunta(var preg : boolean);

Function Checa_Valor (cad : string; valmin, valmax
                : integer) : boolean;

Procedure Mensaje;

Procedure LeeEntero(posx, posy, valmin, valmax :
                integer;
                var numero : word;
                var renglon : byte;
                var columna : byte);

Procedure LeeMatriz( var matrx : matr; dimen : byte; ind : byte);

IMPLEMENTATION

Procedure pregunta(var preg : boolean);

 var cch : char;
  begin
   gotoxy(32,2);
   write ('¿ESTA BIEN ESCRITA LA MATRIZ? ([S] / [N]) ');
   repeat
     cch:= Upcase(readKey);
   until cch in ['S','N'];
   if cch = 'S' then preg:= true else preg:= false;
   gotoxy(32,2); write ('                                    ');
  end;

Function checa_valor (cad : string; valmin, valmax : integer) : boolean;
 var nn, num : integer;                          (* PENDIENTE*)
  begin
   val(cad,num,nn);
   checa_valor:= TRUE;
   If nn = 0 Then
    Begin
     if num < valmin Then checa_valor:= FALSE;
     if num > valmax Then checa_valor:= FALSE
    End
   Else checa_valor:= FALSE;
  end;

Procedure mensaje;
 var cch : char;
  begin
   write(#7);
   gotoxy(10,24);
   write('VALOR FUERA DE RANGO, PARA CONTINUAR OPRIMA
        CUALQUIER TECLA');
   cch:= readkey;
   gotoxy(10,24);
   write('                                              ');
  end;

Procedure LeeEntero(posx, posy, valmin, valmax : integer;
                var numero : word;
                var renglon : byte; var columna : byte);
 var cad : string;
    cch : char;
    long : integer;
  begin
   str(numero,cad);
   long:= length(cad);
   if long > 3 then cad:= copy(cad,1,3);
   gotoxy(posx,posy);
   write('   ');
   gotoxy(posx,posy);
   write(cad);
   repeat
    cch:= readkey;
    case Ord(cch) of
     48..57: begin
             if long < 3 then
              begin
                inc(long);
                cad:= cad + cch;
                gotoxy(posx,posy); write (cad);
                if long = 3 then cch:= chr(13);
              end;
            end;
     8    : if long > 0 then
             begin
              dec(long);
              cad:= copy(cad,1,long);
```

```
          gotoxy(posx + long, posy);
          write(' ');
         end
        else write(#7);
    13  : begin
          if long = 0 then mensaje
          else
           begin
            if checa_valor(cad,valmin,valmax) then  inc(columna)
            else
            begin
              mensaje;
              cch:= ' ';
            end;
           end;
         end;
    0   : begin
          cch:= readkey;
          if ((long >0) and (checa_valor(cad,valmin,valmax))) then
          case cch of
              #72 : dec(renglon);
              #80 : inc(renglon);
              #77 : inc(columna);
              #75 : dec(columna);
          end;
          cch:= chr(13);
         end;
   end;
  until ord(cch) = 13;
  val(cad,numero,long);
 end;

Procedure LeeMatriz( var matrx : matr; dimen : byte; ind : byte);
 var  nn, valmin,  valmax  : integer;
    posx, posy, columna, renglon, cont : byte;
    termina : boolean;
 begin
  valmin:= 0;
  if ind = 0 then valmax:= 1 else valmax:= 999;
  for cont:= 1 to dimen do
   for nn:= 1 to dimen do
    matrx[cont,nn]:= 0;
  clrscr; posx:= 5; posy:= 3;
  for cont:= 1 to dimen do
   begin
    gotoxy(posx,posy); write('y',cont);
    inc(posx,4)
   end;
  posx:= 1; posy:= 4;
  for cont:= 1 to dimen do
   begin
    gotoxy(posx,posy); write('x',cont);
    inc(posy);
   end;
  posx:= 5; posy:= 4;
  clrscr;
  for cont:= 1 to dimen do
   begin
    posx:= 5;
    for nn:= 1 to dimen do
     begin
      gotoxy(posx,posy);
      write(matrx[cont,nn]);
      inc(posx,4);
     end;
    inc(posy);
   end;
```

```
  termina:= false;
  columna:= 1; renglon:= 1;
  repeat
   posx:= 1 + columna*4;
   posy:= 3 + renglon;
   LeeEntero(posx,posy,valmin,valmax,matrx[renglon,columna],renglon,
            columna);
   if columna > dimen then
    begin
     columna:= 1; inc(renglon)
    end;
   if renglon > dimen then
    begin
     pregunta(termina);
     renglon:= 1;
    end;
   if renglon < 1  then renglon:= dimen;
   If columna < 1 Then
    begin
     columna:= dimen; dec(renglon)
    end
  until termina;
 end;
END.



PROGRAM KUMU; (* Algoritmo de Kuhn - Munkres *)

uses crt, captu2;

type  bin = 0..1;
   vector = array [1..dim] of bin;
   matriz = array [1..dim] of vector;
   conj_compa = SET OF byte;

Procedure Presentacion;
 Begin
   GotoXY(3,10);  TextMode(c40); TextColor(Blue);
   TextBackground(LightGray);
   ClrScr;  GotoXY(8,13);
   Write('ALGORITMO DE KUHN-MUNKRES');
   TextColor(Blue+Blink);
   GotoXY(1,25); Write('Oprima RETURN para continuar...');
   ReadLn;
   ClrScr;
   TextMode(c80);
   TextColor(Yellow);
   TextBackground(Blue);
   ClrScr;
   GotoXY(3,10);
   Write('Este programa resuelve el problema de ACOPLAMIENTO
         OPTIMO en');
   Write(' una gráfica ');
   GotoXY(41,11);Write('————————————————');
   GotoXY(26,14);
   Write('BIPARTITA COMPLETA PESADA');
   GotoXY(26,15);
   Write('——————————————————————————');
   GotoXY(1,25); Write('Oprima RETURN para continuar...');
   ReadLn;
   ClrScr;
   GotoXY(3,3);
   Write('Para poder utilizar este programa, se puede introducir la
```

```
    MATRIZ');
Write(' de');
GotoXY(3,5);
Write('EFICIENCIA desde el teclado o por medio de un archivo en
     código ASCII.');
GotoXY(3,8);
Write('a) Si es por ARCHIVO, el formato del archivo es el siguiente:
     el pri-');
GotoXY(6,10);
Write('mer renglón del archivo debe contener sólo un número. Dicho
     número ');
GotoXY(6,12);
Write('representa el orden de una matriz cuadrada (NxN).  El resto del
     ar-');
GotoXY(6,14);
Write('chivo debe contener "N" renglones con "N" números cada uno,
     separa-');
GotoXY(6,16);
Write('dos por un espacio entre sí.');
GotoXY(3,19);
Write('b) Si es por TECLADO, el programa guiará al usuario en la
     introduc-');
GotoXY(6,21);
Write('ción de los valores necesarios para el planteamiento del
     problema.');
GotoXY(1,24);
TextColor(Yellow+Blink);
Write('Nota: El ORDEN MAXIMO de la matriz de eficiencia debe ser
     60, (N ≤ 60).');
TextColor(Yellow);
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr;
GotoXY(3,10);
Write('Si al introducir los datos por el teclado se comete(n) algun(os)
     erro');
Write('r(es).');
GotoXY(3,12);
Write('se debe continuar introduciendo los datos que falten. El
     programa pregunta');
Write('rá');
GotoXY(3,14);
Write('si se desean hacer modificaciones, y se modificará lo que el
     usuario le in-');
GotoXY(3,16);
Write('dique al programa.');
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr;
GotoXY(3,10);
Write('Se informa al usuario que si el problema es de N > "15", las
     matrices ');
GotoXY(3,12);
Write('asociadas al problema no se reportarán en las iteraciones
     intermedias,');
GotoXY(3,14);
Write('solamente en el reporte de los resultados finales. Ello debido al
     limi-');
GotoXY(3,16);
Write('tado espacio en la pantalla. ');
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr;
GotoXY(15,3);
TextColor(Yellow+Blink);
Write('NOTACION UTILIZADA EN ESTE PROGRAMA:');
GotoXY(15,4);
Write('------ -------- -- ----- ------------');
TextColor(Yellow);
GotoXY(3,7);
Write('"S"   denota el subconjunto de los vértices NO SATURADOS
     del conjunto X.');
GotoXY(3,9);
Write('"N(S)" denota a los VECINOS de S.');
GotoXY(3,11);
Write('"P"   denota a la CADENA ALTERNANTE o
     AUMENTANTE.');
GotoXY(3,13);
Write('"T"   es subconjunto de N(S) y es subconjunto de P.');
GotoXY(3,15);
Write('"LXi" denota a la etiqueta asociada al vértice Xi.');
GotoXY(3,17);
Write('"LYj" denota a la etiqueta asociada al vértice Yj.');
GotoXY(1,25); Write('Oprima RETURN para continuar...');
ReadLn;
ClrScr
End;

Procedure Aborta( tm : byte );
 Var
  cc : char;
  ij : integer;
 Begin
  TextMode(c40);
  TextColor(Yellow+Blink);
  TextBackground(Blue);
  ClrScr;
  GotoXY(1,12);
  Write('¡Se cancela la ejecución del programa!');
  GotoXY(10,15);
  Write('¡  N = ',tm,' > 60  !');
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  cc:= #0;
  While cc <> #13 do (* #13 = Carriage Return "Enter"*)
   begin
       For ij:= 1 to 2 do Write(#7);
       readLn(cc);
   end;
  TextMode(c80); ClrScr;
  Halt
 End;

Procedure Leer_Ma_Efi( var nn : matr; var tm : byte );
 var i,j : word;
    x,y,rpan : byte;
     c : char;
    arch: TEXT;
    nomb: string;

  Begin
   clrscr;
   GotoXY(1,13);
   Write('¿Desea dar la matriz de eficiencia por teclado o por archivo?
      (T/A)  ');
   x:= WhereX;  y:= WhereY;
   Repeat
    GotoXY(x,y);
    Readln(c)
   Until c in ['A','a','T','t'];

   case c of
    'A', 'a': begin
              GotoXY(1,18);
              Write('¿Cuál es el nombre del archivo y su trayectoria?
```

```
            :   ');
            Readln(nomb);
            assign(arch,nomb);
            reset(arch);
            readln(arch,tm);
            If tm > 60 Then Aborta(tm);
            for i:= 1 to tm do
             for j:= 1 to tm do
              read(arch,nn[i,j]);
            close(arch)
           end;
   'T', 't': begin
            GotoXY(15,15);
            Write('¿Cuál es el tamaño de la matriz? ( a lo más 60x60
                 ) ');
            Readln(tm);
            If tm > 60 Then Aborta(tm);
            LeeMatriz(nn,tm,1);
          end
  end;

If tm <= 15 Then
 Begin
  ClrScr;
  GotoXY(15,1);
  Write('La matriz de EFICIENCIA es:');
  GotoXY(7,3);
  rpan:= 3;
  For j := 1 to tm do Write('y',j,'  ');  rpan:= 4; Writeln;
  For i := 1 to tm do
   begin
    Write(' x',i:2);
    GotoXY(5,rpan);
    For j := 1 to tm do Write(nn[i,j]:4);
    Writeln;
    rpan:= rpan + 1
   end;
  GotoXY(3,23);
  Write('¿Desea hacer algun(os) cambio(s) en la matriz? (s/n)  ');
  x:= WhereX;  y:= WhereY;
  Repeat
   GotoXY(x,y);
   Readln(c)
  Until c in ['S','s','N','n'];

  If c in ['S','s'] Then
   Begin
    Repeat
     ClrScr;
     GotoXY(15,10);
     Writeln('¿Qué elemento quiere cambiar?');
     GotoXY(15,15);
     Write('Renglón = ? ');  Read(i);
     GotoXY(15,18);
     Write('Columna = ? ');  ReadLn(j);
     GotoXY(20,22);
     Write('Nuevo valor de la entrada ',i:3,',',j:3,' = ?  ');
     Read(nn[i,j]);
     ClrScr;
     GotoXY(10,15);
     Write('¿Desea hacer algún otro cambio en la matriz? (s/n)  ');
     x:= WhereX;  y:= WhereY;
     c := ' ';
     While NOT (c in ['S','s','N','n']) do
      Begin
       GotoXY(x,y);
       c := ReadKey;
      End
     Until c in ['N','n'];
     ClrScr;
     GotoXY(15,1);
     Write('La matriz de EFICIENCIA es:');
     GotoXY(7,2);
     rpan:= 2;
     For j := 1 to tm do Write('y',j,'  ');  Writeln; rpan:= 3;
     For i := 1 to tm do
      begin
       Write(' x',i:2);
       GotoXY(5,rpan);
       For j := 1 to tm do Write(nn[i,j]:4);
       Writeln; rpan:= rpan + 1
      end;
     GotoXY(1,25); Write('Oprima RETURN para continuar...');
     ReadLn(c);
     ClrScr
    End
   End
 End;

Procedure Inicia_Mat_Acopla( var m1 : matriz; tatam : byte);
 var i, j : byte;
  Begin
   For i:= 1 to tatam do
    For j:= 1 to tatam do
     m1[i,j]:= 0;
  End;


Procedure Inic_Vec_Aco( var vec : vector; long : byte );
 var i : byte;
  Begin
   For i:= 1 to long do vec[i] := 0
  End;


Procedure Ini_Eti( var vec : vect; long : byte );
 var i : byte;
  Begin
   For i:= 1 to long do vec[i] := 0
  End;


Procedure Ini_EtiX( var EtX : vect; MaEfi : matr;
                  var Mdy, Mco : matriz;
                  var vex : vector; tlon : byte);
 var ir,ic : byte;
   vxx, vyy : vector;
  Begin
   For ir:= 1 to tlon do
    begin
     EtX[ir]:= 0;
     For ic:= 1 to tlon do
      if MaEfi[ir,ic] > EtX[ir] then EtX[ir]:= MaEfi[ir,ic]
    end;
   For ir:= 1 to tlon do
    For ic:= 1 to tlon do
     if EtX[ir] = MaEfi[ir,ic] then Mdy[ir,ic]:= 1;
   For ir:= 1 to tlon do
    begin
     vxx[ir]:= 0; vyy[ir]:= 0
    end;
   For ir:= 1 to tlon do
    For ic:= 1 to tlon do
     if (Mdy[ir,ic] = 1) AND (vxx[ir] = 0) AND (vyy[ic] = 0) then
```

```
      begin
        Mco[ir,ic]:= 1;
        vxx[ir]:= 1;
        vyy[ic]:= 1;
        ic:= tlon
      end;
   For ir:= 1 to tlon do
     if vxx[ir] = 1 then vex[ir]:= 1
  End;


Function Paso1A( vecto : vector; longi : byte ): byte;
 var k,aux : byte;
  Begin
   aux:= 0;
   For k := 1 to longi do
     If vecto[k] = 0 Then
       begin
         aux := k;     k := longi
       end;
    Paso1A := aux
  End;


Function  CardiS( con_S : vector; ta1 : byte ): byte;
 var i : byte;
  Begin
    i := 1;
    While  (con_S[i] <> 0) AND (i <= ta1) do i := i + 1;
    If i > ta1 Then CardiS := ta1
    Else CardiS := i - 1
  End;


Function  CardAlf( con_S : vect; ta1 : byte ): byte;
 var i : byte;
  Begin
    i := 1;
    While  (i <= ta1) AND (con_S[i] <> 0) do i := i + 1;
    If i > ta1 Then CardAlf := ta1
    Else CardAlf := i - 1
  End;

Procedure  Los_Vecinos_de_S( var NdeS : vector; conj_S : vector;
                      mat_ad : matriz;   ttam : byte );
 var cardS, i, j, k : byte;
   cardN : byte;
  Begin
      cardS := CardiS(conj_S,ttam);
      cardN := CardiS(NdeS,ttam);
      cardN := cardN + 1;
      For i:= 1 to cardS do
        For j := 1 to ttam do
          If mat_ad[conj_S[i],j] = 1
            Then
             begin
               k:= 1;
               While (NdeS[k] <> j) AND (k <= (cardN - 1)) do
                     k := k + 1;
               If k = cardN Then
                  begin
                   NdeS[cardN] := j;
                   cardN := cardN + 1
                  end
             end
   End;

Function  Compara_Conjuntos( conj1, conj2 : vector; taam : byte):
                        Boolean;
 var i, car1, car2 : byte;
   con1, con2 : conj_compa;
  Begin
    car1 := CardiS(conj1,taam);  car2 := CardiS(conj2,taam);
    If car1 = car2 Then
     begin
            con1 := []; con2 := [];
         For i := 1 to car1 do
           begin
             con1 := con1 + [conj1[i]];
             con2 := con2 + [conj2[i]]
           end;
         If con1 = con2 then Compara_Conjuntos := TRUE
         Else Compara_Conjuntos := FALSE
     end
    Else  Compara_Conjuntos := FALSE
  End;

Procedure   Unir_Conj(var reunion : vector;  eleme, es : byte );
 var card, i : byte;
  Begin
    card := CardiS(reunion,es);
     i:= 1;
     While (reunion[i] <> eleme) AND (i <= card) do
            i := i + 1;
      If i > card Then
          reunion[i] := eleme
  End;

Procedure   UnirConAlf(var reunion : vect;  eleme : word;  es : byte );
 var card, i : byte;
  Begin
    card := CardAlf(reunion,es);
     i:= 1;
     While (i <= card) AND (reunion[i] <> eleme) do
            i := i + 1;
      If i > card Then
          reunion[i] := eleme
  End;

Procedure   Unir_Traye(var reunion : vector; eleme, es : byte);
var card : byte;
  Begin
    card := CardiS(reunion,es);
    reunion[card+1] := eleme
  End;

Procedure  FormarNS_T( NdeSS,TT : vector ;  var DifNT : conj_compa;
                   tam1 : byte);
 var i, carNS, carT : byte;
   conNS, conT : conj_compa;
  Begin
    carNS := CardiS(NdeSS,tam1);
    carT := CardiS(TT,tam1);
    conNS := []; conT := [];
    For i := 1 to carNS do
      conNS := conNS + [NdeSS[i]];
    For i := 1 to carT do
      conT := conT + [TT[i]];
    DifNT := conNS - conT
  End;

Function  Buscar_Col( ren, inic : byte; mmaat : matriz; siz1 : byte ): byte;
  var ii, aux : byte;
   Begin
```

```
  ii := inic; aux := 0;
  While ii <= siz1 do
    begin
     if mmaat[ren,ii] = 0 Then ii := ii + 1
     Else
      begin
       aux := ii;
       ii := siz1 + 1
      end
    end;
  Buscar_Col := aux
 End;

Function Buscar_Ren( col, prim : byte; maatt : matriz; ziz : byte ): byte;
 var ii, aux : byte;
  Begin
   ii := prim; aux := 0;
   While ii <= ziz do
    begin
     if maatt[ii,col] = 0 Then ii := ii + 1
     Else
      begin
       aux := ii;
       ii := ziz + 1
      end
    end;
   Buscar_Ren := aux
  End;

Procedure Alfas(var EX,EY : vect; Efi : matr; SS,TT : vector; dsn:byte);
 var mi, mj, carS, carT, carAlf : byte;
     setT, conS : conj_compa;
     alfs : vect;
     aux,min : word;
  Begin
   setT:= []; conS:= [];
   carT:= CardiS(TT,dsn);  carS:= CardiS(SS,dsn);
   For mi:= 1 to carT do
           setT:= setT + [TT[mi]];

   For mi:= 1 to carS do conS:= conS + [SS[mi]];
   For mi:= 1 to dim do alfs[mi]:= 0;
   For mi:= 1 to carS do
    For mj:= 1 to dsn do
     if NOT (mj in setT) then
      begin
       aux:= EX[SS[mi]] + EY[mj] - Efi[SS[mi],mj];
       if aux > 0 then UnirConAlf(alfs,aux,dim)
      end;
   min:= 65535;  carAlf:= CardAlf(alfs,dim);
   For mi := 1 to carAlf do
    if alfs[mi] < min then min:= alfs[mi];
   For mi:= 1 to dsn do
    if (mi in conS) then EX[mi]:= EX[mi] - min;
   For mi:= 1 to dsn do
    if (mi in setT) then EY[mi]:= EY[mi] + min
  End;

Procedure  Act_MatAdy( LLX : vect; var AdyM : matriz; Efic : matr;
                 llong : byte);
 var mi, mj : byte;
     vex, vey : vector;
  Begin
   For mi:= 1 to llong do
    For mj:= 1 to llong do
     if LLX[mi] = Efic[mi,mj] then
        AdyM[mi,mj]:= 1;
  End;

Procedure  Acoplar_P_M_Aco( cadena : vector; var acoplamiento :
                 matriz; esp : byte);
 var carCad, ii, jj : byte;
  Begin
   carCad := CardiS(cadena,12);
   jj := 1;
   While jj <= carCad do
    begin
     For ii:= 1 to esp do
       acoplamiento[cadena[jj],ii]:= 0;
     For ii:= 1 to esp do
       acoplamiento[ii,cadena[jj+1]]:= 0;
     acoplamiento[cadena[jj],cadena[jj+1]] := 1;   jj := jj + 2
    end
  End;

Procedure Actuali_VecA( aacop : matriz; var vec : vector;
                 lon1 : byte );
 var k, l : byte;
  Begin
   For k := 1 to lon1 do vec[k] := 0;
   For k := 1 to lon1 do
    For l := 1 to lon1 do
     If aacop[k,l] = 1 Then
       begin
        vec[k] := 1;
        l := lon1
       end;
  End;

Procedure  Salida( acco : matriz; Nefi : matr; ttam : byte);
 var  c, d, rp : byte;
      suma : longint;

  Begin
    ClrScr;
    GotoXY(20,1);
    Write('╔══════════════════════════════════════╗');
    GotoXY(20,2);
    Write('║ LA MATRIZ DE ACOPLAMIENTO OPTIMO ES: ║');
    GotoXY(20,3);
    Write('╚══════════════════════════════════════╝');
    GotoXY(8,5);
    rp:= 5;
    suma:= 0;
    For d := 1 to ttam do Write('y',d,'  ');  Writeln; rp:= 6;
    For c := 1 to ttam do
     begin
      Write(' x',c:2);
      GotoXY(5,rp);
      For d := 1 to ttam do
       Begin
        Write(acco[c,d]:4);
        If acco[c,d] = 1 Then suma:= suma + Nefi[c,d]
       End;
      Writeln;
      rp:= rp +1
     end;
    Writeln;
    Write('El PESO del ACOPLAMIENTO es: ':50,suma);
    For c:= 1 to 2 do Write(#7);
    Readln;
    TextMode(c80);
    ClrScr
  End;
```

```
VAR mat_ady, mat_aco : matriz;
   MatEfi : matr;
  tam, tam2, u, uu, u2, yj, w, ti, yjj, iteracion, Re, Co, Repa : byte;
  vec_acox, No_Saturado, Vecinos, P_trayec : vector;
  c0 : char;
  Vecinos_de_S: vector;
  iguales, PAS2, pertenece : boolean;
  Dife_N_T : conj_compa;
  LX, LY : vect;

BEGIN
  Presentacion;
  Leer_Ma_Efi(MatEfi, tam);
  tam2:= 2*tam;
  clrscr;
  GotoXY(1,13);
  Write('¿Desea ver todas las iteraciones del algoritmo? (S/N)    ');
  Co:= WhereX;  Re:= WhereY;
  Repeat
    GotoXY(Co,Re);
    Readln(c0)
  Until c0 in ['S','s','N','n'];
  IF c0 IN ['S','s'] THEN
  BEgin
  Inicia_Mat_Acopla(mat_aco, tam);
  Inicia_Mat_Acopla(mat_ady, tam);
  Inic_Vec_Aco(vec_acox, tam);
  Inic_Vec_Aco(No_Saturado,tam);
  Inic_Vec_Aco(P_trayec,tam2);
  Inic_Vec_Aco(Vecinos_de_S,tam);
  Ini_Eti(LY,tam);
  Ini_EtiX(LX,MatEfi,Mat_ady,mat_aco,vec_acox,tam);
  ClrScr;
  GotoXY(33,1);
  Write(' ╔══════════╗ ');
  GotoXY(33,2);
  Write(' ║  PASO 0  ║ ');
  GotoXY(33,3);
  Write(' ╚══════════╝ ');
  GotoXY(3,5);
  Write('S    = φ');
  GotoXY(3,7);
  Write('N(S) = φ');
  GotoXY(3,9);
  Write('P    = φ');
  GotoXY(3,11);
  Write('T    = φ');
  GotoXY(3,13); Co:= 13;
  For Re:= 1 to tam do
    Begin
     If ( (Re MOD 6) = 0 ) Then
       Begin
         Co := Co + 2;
         GotoXY(3,Co)
       End;
     Write('LX',Re,'= ',LX[Re],',  ');
    End;
   GotoXY(3,Co+2);
   Write('LYj  = 0, para TODOS los vértices de Y');
   GotoXY(1,25); Write('Oprima RETURN para continuar...');
   ReadLn;
   ClrScr;
   GotoXY(33,1);
   Write(' ╔══════════╗ ');
   GotoXY(33,2);
   Write(' ║  PASO 0  ║ ');
   GotoXY(33,3);
Write(' ╚══════════╝ ');
If tam <= 15 Then
 Begin
  GotoXY(10,5);
  Write('La matriz de ADYACENCIAS asociada a la subgráfica es:');
  GotoXY(7,7);
  Repa:= 7;
  For Co := 1 to tam do Write('y',Co,'  ');  Writeln;   Repa := 8;
  For Re := 1 to tam do
   begin
     Write(' x',Re:2);
     GotoXY(5,Repa);
     For Co := 1 to tam do Write(mat_ady[Re,Co]:4);
     Writeln; Repa:= Repa + 1;
   end;
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  ReadLn;
  ClrScr;
  GotoXY(33,1);
  Write(' ╔══════════╗ ');
  GotoXY(33,2);
  Write(' ║  PASO 0  ║ ');
  GotoXY(33,3);
  Write(' ╚══════════╝ ');
  GotoXY(10,5);
  Write('La matriz de ACOPLAMIENTO para la subgráfica es:');
  GotoXY(7,7);
  Repa:= 7;
  For Co := 1 to tam do Write('y',Co,'  ');  Writeln;  Repa:= 8;
  For Re := 1 to tam do
   begin
    Write(' x',Re:2);
    GotoXY(5,Repa);
    For Co := 1 to tam do Write(mat_aco[Re,Co]:4);
    Writeln; Repa:= Repa + 1;
   end;
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  ReadLn;
  ClrScr
 End;

u:= Paso1A(vec_acox,tam);
iteracion:= 0;

While  u > 0  do
  Begin
    iteracion:= iteracion + 1;
    Unir_Conj(No_Saturado,u,tam);
    Unir_Traye(P_trayec,u,tam2);     (* P := P U {u} *)
    Inic_Vec_Aco(Vecinos,tam);  (* T := φ *)
    ClrScr;
    GotoXY(33,1);
    Write(' ╔══════════╗ ');
    GotoXY(33,2);
    Write(' ║  PASO 1  ║ ');
    GotoXY(33,3);
    Write(' ╚══════════╝ ');
    GotoXY(3,7);
    Repa:= 7;
    Co := CardiS(No_saturado,tam);
    Write('S    = { X',No_Saturado[1]);
    For Re:= 2 to Co do
      Begin
       If ( (Re MOD 6) = 0 ) Then
        Begin
          Repa := Repa + 2;
          GotoXY(9,Repa)
```

```
      End;
    Write(',  ','X',No_Saturado[Re])
   End;
  Write(' }.');
  GotoXY(3,Repa+2);
  Repa:= Repa + 2;
  Co := CardiS(P_trayec,tam);
  Write('P    = { X',P_trayec[1]);
  For Re:= 2 to Co do
   Begin
    If ( (Re MOD 6) = 0 ) Then
     Begin
      Repa := Repa + 2;
      GotoXY(9,Repa)
     End;
    If ( (Re MOD 2) = 0 ) Then Write(',  ','X',P_trayec[Re])
    Else Write(',  ','Y',P_trayec[Re])
   End;
  Write(' }.');
  GotoXY(3,Repa+2);
  Repa:= Repa + 2;
  Write('T    = ϕ');
  GotoXY(3,Repa+2);
  Repa:= Repa + 2;
  Write('N(S) = ϕ');
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  ReadLn;
  ClrScr;
  PAS2:= TRUE;
  While PAS2 = TRUE do
   begin
    Los_Vecinos_de_S(Vecinos_de_S, No_Saturado, mat_ady, tam);
    iguales:= Compara_Conjuntos(Vecinos_de_S, Vecinos, tam);
    ClrScr;
    GotoXY(33,1);
    Write(' ╔════════╗ ');
    GotoXY(33,2);
    Write(' ║  PASO 2  ║ ');
    GotoXY(33,3);
    Write(' ╚════════╝ ');
    GotoXY(3,7);
    Repa:= 7;
    GotoXY(3,Repa+2);
    Repa:= Repa + 2;
    Co := CardiS(Vecinos_de_S,tam);
    Write('N(S) = { Y',Vecinos_de_S[1]);
    For Re:= 2 to Co do
     Begin
      If ( (Re MOD 6) = 0 ) Then
       Begin
        Repa := Repa + 2;
        GotoXY(9,Repa)
       End;
       Write(',  ','Y',Vecinos_de_S[Re])
      End;
     Write(' }.');
     GotoXY(3,Repa+2);
     Repa:= Repa + 2;
     Co := CardiS(Vecinos,tam);
     If Co > 0 Then
     Begin
      Write('T    = { Y',Vecinos[1]);
      For Re:= 2 to Co do
       Begin
        If ( (Re MOD 6) = 0 ) Then
         Begin
          Repa := Repa + 2;
          GotoXY(9,Repa)
         End;
        Write(',  ','Y',Vecinos[Re])
       End;
      Write(' }.');
     End
    Else
     Write('T    = ϕ');
    GotoXY(1,25); Write('Oprima RETURN para continuar...');
    ReadLn;
    ClrScr;
    If iguales = TRUE Then
     begin
     Alfas(LX,LY,MatEfi,No_Saturado,Vecinos,tam);
     Act_MatAdy(LX,Mat_ady,MatEfi,tam);
     uu:= Paso1A(vec_acox,tam);
     while uu <= tam do
      Begin
       yj := Buscar_Col(uu,1,mat_ady,tam);
       Repeat
        u2 := Buscar_Ren(yj,1,mat_aco,tam);
        If u2 = 0 Then
         begin
          If mat_ady[uu,yj] = 1 Then mat_aco[uu,yj]:= 1;
          yj:= tam + 1;
          Actuali_VecA(mat_aco,vec_acox,tam)
         end
        Else
         yj:= yj + 1
       Until yj > tam;
       uu:= uu + 1
      End;
     PAS2:= FALSE;
     ClrScr;
     GotoXY(33,1);
     Write(' ╔════════╗ ');
     GotoXY(33,2);
     Write(' ║  PASO 2  ║ ');
     GotoXY(33,3);
     Write(' ╚════════╝ ');
     GotoXY(3,7);
     Repa:= 7;
     GotoXY(3,Repa+2);
     Repa:= Repa + 2;
     Co := CardiS(Vecinos_de_S,tam);
     Write('N(S) = { Y',Vecinos_de_S[1]);
     For Re:= 2 to Co do
      Begin
       If ( (Re MOD 6) = 0 ) Then
        Begin
         Repa := Repa + 2;
         GotoXY(9,Repa)
        End;
       Write(',  ','Y',Vecinos_de_S[Re])
      End;
     Write(' }.');
     GotoXY(3,Repa+2);
     Repa:= Repa + 2;
     Co := CardiS(Vecinos,tam);
     If Co > 0 Then
      Begin
       Write('T    = { Y',Vecinos[1]);
       For Re:= 2 to Co do
        Begin
         If ( (Re MOD 6) = 0 ) Then
          Begin
           Repa := Repa + 2;
```

```
        GotoXY(9,Repa)
       End;
      Write(',  ','Y',Vecinos[Re])
     End;
    Write(' }.');
   End
  Else
   Write('T    = ϕ');
  GotoXY(3,Repa+2);
  Repa:= Repa + 2;
  For Re:= 1 to tam do
   Begin
    If ( (Re MOD 6) = 0 ) Then
     Begin
      Repa := Repa + 2;
      GotoXY(3,Repa)
     End;
    Write('LX',Re,'= ',LX[Re],',  ');
   End;
  GotoXY(3,Repa+2);
  Repa:= Repa + 2;
  For Re:= 1 to tam do
   Begin
    If ( (Re MOD 6) = 0 ) Then
     Begin
      Repa := Repa + 2;
      GotoXY(3,Repa)
     End;
    Write('LY',Re,'= ',LY[Re],',  ')
   End;
  GotoXY(1,25); Write('Oprima RETURN para continuar...');
  ReadLn;
  If tam <= 15 Then
   Begin
    ClrScr;
    GotoXY(33,1);
    Write(' ╔══════════╗ ');
    GotoXY(33,2);
    Write(' ║  PASO 2  ║ ');
    GotoXY(33,3);
    Write(' ╚══════════╝ ');
    GotoXY(10,5);
    Write('La matriz de ADYACENCIAS asociada a la
          subgráfica es:');
    GotoXY(7,7);
    Repa:= 7;
    For Co := 1 to tam do Write('y',Co,'  '); Writeln;
    Repa:= 8;
    For Re := 1 to tam do
     begin
      Write(' x',Re:2);
      GotoXY(5,Repa);
      For Co := 1 to tam do Write(mat_ady[Re,Co]:4);
      Writeln; Repa:= Repa + 1
     end;
    GotoXY(1,25);
    Write('Oprima RETURN para continuar...');
    ReadLn;
    ClrScr;
    GotoXY(33,1);
    Write(' ╔══════════╗ ');
    GotoXY(33,2);
    Write(' ║  PASO 2  ║ ');
    GotoXY(33,3);
    Write(' ╚══════════╝ ');
    GotoXY(10,5);
    Write('La matriz de ACOPLAMIENTO para la subgráfica
          es:');
    GotoXY(7,7);
    Repa:= 7;
    For Co := 1 to tam do Write('y',Co,'  '); Writeln;
    Repa:= 8;
    For Re := 1 to tam do
     begin
      Write(' x',Re:2);
      GotoXY(5,Repa);
      For Co := 1 to tam do Write(mat_aco[Re,Co]:4);
      Writeln; Repa:= Repa + 1
     end;
    GotoXY(1,25);
    Write('Oprima RETURN para continuar...');
    ReadLn;
    ClrScr
   End;
  Inic_Vec_Aco(No_Saturado,tam);
  Inic_Vec_Aco(P_trayec,tam2);
  Inic_Vec_Aco(Vecinos_de_S,tam);
  u:= Paso1A(vec_acox,tam)
 end
Else
 Begin
  FormarNS_T(Vecinos_de_S,Vecinos,Dife_N_T,tam);
  yj:= 1;
  Repeat
   yj := Buscar_Col(u,yj,mat_ady,tam);
   If yj in Dife_N_T  Then
    begin
     yjj:= yj;
     For w:= yjj to tam do
      begin
       ti:= Buscar_Ren(w,1,mat_aco,tam);
       If (ti = 0) AND (w in Dife_N_T) AND
          (mat_ady[u,w] = 1)
        Then
         begin
          yj:= w;
          w:= tam
         end
       Else
        ti:= 1
      end;
     If ti = 0 Then
      Begin
       Unir_Traye(P_trayec,yj,tam2);
       Acoplar_P_M_Aco(P_trayec,mat_aco,tam);
       Inic_Vec_Aco(No_Saturado,tam);
       Inic_Vec_Aco(P_trayec,tam);
       Inic_Vec_Aco(Vecinos_de_S,tam);
       Actuali_VecA(mat_aco,vec_acox,tam);
       u:= Paso1A(vec_acox,tam);
       PAS2 := FALSE;
       pertenece:= TRUE
      End
     Else
      Begin
       pertenece := TRUE;
       Unir_Traye(P_trayec,yj,tam2);)
        u := Buscar_Ren(yj,1,mat_aco,tam);
       If u > 0 Then
        begin
         Unir_Conj(No_Saturado,u,tam);
         Unir_Conj(Vecinos,yj,tam);
          Unir_Traye(P_trayec,u,tam2);
          ClrScr;
```

```
GotoXY(33,1);
Write(' ╔══════════╗ ');
GotoXY(33,2);
Write(' ║  PASO 3  ║ ');
GotoXY(33,3);
Write(' ╚══════════╝ ');
GotoXY(3,7);
Repa:= 7;
Co := CardiS(No_saturado,tam);
Write('S    = { X',No_Saturado[1]);
For Re:= 2 to Co do
 Begin
  If ( (Re MOD 6) = 0 ) Then
   Begin
    Repa := Repa + 2;
    GotoXY(9,Repa)
   End;
  Write(',  ','X',No_Saturado[Re])
 End;
Write(' }.');
GotoXY(3,Repa+2);
Repa:= Repa + 2;
Co := CardiS(Vecinos_de_S,tam);
Write('N(S) = { Y',Vecinos_de_S[1]);
For Re:= 2 to Co do
  Begin
   If ( (Re MOD 6) = 0 ) Then
    Begin
     Repa := Repa + 2;
     GotoXY(9,Repa)
    End;
  Write(',  ','Y',Vecinos_de_S[Re])
 End;
Write(' }.');
GotoXY(3,Repa+2);
Repa:= Repa + 2;
Co := CardiS(Vecinos,tam);
If Co > 0 Then
 Begin
  Write('T    = { Y',Vecinos[1]);
  For Re:= 2 to Co do
   Begin
    If ( (Re MOD 6) = 0 ) Then
     Begin
      Repa := Repa + 2;
      GotoXY(9,Repa)
     End;
    Write(',  ','Y',Vecinos[Re])
   End;
  Write(' }.');
 End
Else Write('T    = ϕ');
GotoXY(3,Repa+2);
Repa:= Repa + 2;
Co := CardiS(P_trayec,tam);
Write('P    = { X',P_trayec[1]);
For Re:= 2 to Co do
 Begin
  If ( (Re MOD 6) = 0 ) Then
   Begin
    Repa := Repa + 2;
    GotoXY(9,Repa)
   End;
  If ( (Re MOD 2) = 0 ) Then Write(',
   ','Y',P_trayec[Re])
  Else Write(',  ','X',P_trayec[Re])
 End;
           Write(' }.');
           GotoXY(1,25);
           Write('Oprima RETURN para  continuar...');
           ReadLn; ClrScr
          end
         Else
          begin
           Acoplar_P_M_Aco(P_trayec,mat_aco,tam);
           Inic_Vec_Aco(No_Saturado,tam);
           Inic_Vec_Aco(P_trayec,tam);
           Inic_Vec_Aco(Vecinos_de_S,tam);
           Actuali_VecA(mat_aco,vec_acox,tam);
           u:= Paso1A(vec_acox,tam);
           PAS2 := FALSE
          end
        End
       end
      Else
       begin
        pertenece := FALSE;
        yj := yj + 1
       end
    Until pertenece = TRUE
    End
   end  (* PASO 2 *)
  End; (* PASO 1 *)
 ENd
ELSE
 BEgin
  Inicia_Mat_Acopla(mat_aco, tam);
  Inicia_Mat_Acopla(mat_ady, tam);
  Inic_Vec_Aco(vec_acox, tam);
  Inic_Vec_Aco(No_Saturado,tam);
  Inic_Vec_Aco(P_trayec,tam2);
  Inic_Vec_Aco(Vecinos_de_S,tam);
  Ini_Eti(LY,tam);
  Ini_EtiX(LX,MatEfi,Mat_ady,mat_aco,vec_acox,tam);
  u:= Paso1A(vec_acox,tam);
  iteracion:= 0;
  While  u > 0  do
   Begin
    iteracion:= iteracion + 1;
    Unir_Conj(No_Saturado,u,tam);
    Unir_Traye(P_trayec,u,tam2);
    Inic_Vec_Aco(Vecinos,tam);
    PAS2:= TRUE;
    While PAS2 = TRUE do
     begin
      Los_Vecinos_de_S(Vecinos_de_S, No_Saturado, mat_ady, tam);
      iguales:= Compara_Conjuntos(Vecinos_de_S, Vecinos, tam);
      If iguales = TRUE Then
       begin
        Alfas(LX,LY,MatEfi,No_Saturado,Vecinos,tam);
        Act_MatAdy(LX,Mat_ady,MatEfi,tam);
       uu:= Paso1A(vec_acox,tam);
       while uu <= tam do
        Begin
         yj := Buscar_Col(uu,1,mat_ady,tam);
         Repeat
         u2 := Buscar_Ren(yj,1,mat_aco,tam);
         If u2 = 0 Then
          begin
           If mat_ady[uu,yj] = 1  Then mat_aco[uu,yj]:= 1;
           yj:= tam + 1;
           Actuali_VecA(mat_aco,vec_acox,tam)
          end
         Else yj:= yj + 1
```

```
      Until yj > tam;
      uu:= uu + 1
    End;
   PAS2:= FALSE;
   Inic_Vec_Aco(No_Saturado,tam);
   Inic_Vec_Aco(P_trayec,tam2);
   Inic_Vec_Aco(Vecinos_de_S,tam); (* N(S) = vacio *)
   u:= Paso1A(vec_acox,tam)
  end
Else
  Begin
   FormarNS_T(Vecinos_de_S,Vecinos,Dife_N_T,tam);
   yj:= 1;
   Repeat
    yj := Buscar_Col(u,yj,mat_ady,tam);
    If yj in Dife_N_T  Then
      begin
       yjj:= yj;
       For w:= yjj to tam do
        begin
         ti:= Buscar_Ren(w,1,mat_aco,tam);
         If (ti = 0) AND (w in Dife_N_T) AND
           (mat_ady[u,w] = 1)
         Then
          begin
           yj:= w;
           w:= tam
          end
         Else ti:= 1
        end;
       If ti = 0 Then
        Begin
         Unir_Traye(P_trayec,yj,tam2);
         Acoplar_P_M_Aco(P_trayec,mat_aco,tam);
         Inic_Vec_Aco(No_Saturado,tam);
         Inic_Vec_Aco(P_trayec,tam);
         Inic_Vec_Aco(Vecinos_de_S,tam);
         Actuali_VecA(mat_aco,vec_acox,tam);
         u:= Paso1A(vec_acox,tam);
         PAS2 := FALSE;
         pertenece:= TRUE
        End
       Else
        Begin
         pertenece := TRUE;
         Unir_Traye(P_trayec,yj,tam2);
         u := Buscar_Ren(yj,1,mat_aco,tam);
         If u > 0 Then
          begin
           Unir_Conj(No_Saturado,u,tam);
           Unir_Conj(Vecinos,yj,tam);
           Unir_Traye(P_trayec,u,tam2);
          end
         Else
          begin
           Acoplar_P_M_Aco(P_trayec,mat_aco,tam);
           Inic_Vec_Aco(No_Saturado,tam);
           Inic_Vec_Aco(P_trayec,tam);
           Inic_Vec_Aco(Vecinos_de_S,tam);
           Actuali_VecA(mat_aco,vec_acox,tam);
           u:= Paso1A(vec_acox,tam);
           PAS2 := FALSE
          end
        End
      end  (* yj en Dife_N_T *)
     Else
      begin
          pertenece := FALSE;
          yj := yj + 1
        end
      Until pertenece = TRUE
     End
   end
  End;
 ENd;
Salida(mat_aco,MatEfi,tam)
END.
```

Bibliografía.

Acerson, K. WordPerfect 5.5 Manual de Referencia, Ed. McGraw-Hill.

Acoff, S. Fundamentos de Investigación de Operaciones, Ed. Limusa Noriega.

Bazaraa, J. Linear Programing and Network Flows, Ed. Wiley.

Bondy, J. Graph Theory With Applications, Ed. Macmillan Press.

Chachra, P. Applications of Graphs Theory Algorithms. Ed Noth Hollan.

Christofides, N. Graph Theory an Algorithmic Approach, Ed. Academic Press.

Flores de la Mota, I. Apuntes de Programación Entera, Departamento de Ingeniería de Sistemas UNAM.

Gupta, C. Fundamental of Operations Research for Management, Ed. Prentice Hall.

Hillier, F. Introducción a la Investigación de Operaciones, Ed. McGraw-Hill.

Jaufred, Moreno, Acosta. Métodos de Optimización, Ed Representaciones y Servicios de Ingeniería.

Joyanes, L. Programación en Turbo Pascal, Ed. McGraw-Hill.

Mincberg, M. WordPerfect A su Alcance, Ed. McGraw-Hill.

O'Brien, S. Turbo Pascal 5.5 The Complete Reference, Ed. McGraw-Hill.

Prawda, J. Métodos y Modelos de Investigación de Operaciones. Vol. 1, Ed. Limusa Noriega.

Ventsel, E. Investigación de Operaciones Problemas, Principios y Metodología, Ed. MIR Moscú.