3
25j

# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

# PROPIEDADES Y ALGORITMOS PARA EL PROBLEMA DEL AGENTE VIAJERO

# T E S I S

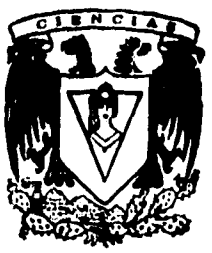
QUE PARA OBTENER EL TITULO DE:

A C T U A R I A

P R E S E N T A :

ROSALIA AGUIRRE HERNANDEZ



VNIVERSIDAD NACIONAL
AVFNºMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:
" PROPIEDADES Y ALGORITMOS PARA EL
PROBLEMA DEL AGENTE VIAJERO "

realizado por Rosalía Aguirre Hernández

con número de cuenta 9052175-7 , pasante de la carrera de actuaría

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

| | | | |
|---|---|---|---|
| M. en C. | María del Carmen | Hernández Ayuso | |
| Director de Tesis | | | |
| Propietario | | | |
| M. en C. | Beatriz | Rodríguez Fernández | |
| Propietario | | | |
| M. en C. | Victor Rafael | Pérez Pérez | 9-Mayo-96 |
| Propietario | | | |
| M. en C. | Guadalupe | Ibargüengoitia González | |
| Suplente | | | |
| Mat. | Margarita Elvira | Chávez Cano | |
| Suplente | | | |

Act. Claudia Carrillo Quiroz
Consejo Departamental de Matemáticas

A mis padres, a mis hermanos y a

Alberto por su amor y apoyo.

# Contenido

# Introducción

En el problema del agente viajero se da un conjunto de $N$ ciudades y para cada par de ciudades $i, j$ la distancia $c_{ij}$ que hay entre éstas. El problema consiste en encontrar una permutación $\pi$ de ciudades que minimice la cantidad

$$\sum_{i=1}^{N-1} c_{\pi_i \pi_{i+1}} + c_{\pi_N \pi_1}$$

Esta cantidad representa la "longitud del *tour*", por lo tanto se deben visitar las ciudades en el orden especificado por la permutación y regresar a la ciudad inicial.

El problema del agente viajero es el más famoso de los problemas combinatorios. Esto se debe, entre otras razones, a que es un problema fácil de plantear y sin embargo hasta la fecha es considerado como un problema difícil de resolver. Este problema fue uno de los primeros donde se aplicó la teoría de $NP-completez$ a principios de los 70's. A partir de entonces se ha usado como el ejemplo prototipo de los problemas combinatorios $NP-difíciles$.

Además, el problema del agente viajero ha dado pie al desarrollo de nuevos algoritmos. Por ejemplo, el método de relajación Lagrangeana se desarrolló a partir del trabajo de Held y Karp para resolver el problema del agente viajero.

Una vez que se sabe que el problema es $NP-difícil$, y por lo tanto es improbable que exista un algoritmo polinomial que encuentre la solución óptima, se buscan algoritmos aproximados eficientes. Es decir, algoritmos que encuentran una solución cercana a la óptima en un tiempo corto. Hasta la fecha, los mejores algoritmos de este tipo son aquéllos que se basan en una técnica conocida como optimización local, en la que una solución se mejora continuamente al realizar cambios locales.

El objetivo de este trabajo es estudiar las propiedades del problema del agente viajero, así como algunos algoritmos para resolverlo y comparar la

eficiencia de éstos. La tesis está estructurada de la siguiente manera. En el primer capítulo se define y se modela el problema. Además, se demuestra que es un problema difícil de resolver. En el segundo capítulo se describen algunos algoritmos que resuelven problemas combinatorios en general. Sin embargo, se mostrará que éstos no son adecuados para resolver el problema del agente viajero. En el capítulo 3 se estudian algoritmos más eficientes que se han desarrollado específicamente para resolver este problema. Por último, se realizó un manual del paquete de cómputo en donde se implementaron algunos algoritmos descritos en la tesis.

# Capítulo 1

# El Problema del Agente Viajero.

En este capítulo se define el problema del agente viajero. Además se enuncian algunos conceptos de teoría de gráficas con el propósito de plantearlo como un problema de gráficas. Después se estudia el problema de programación matemática para mostrar que el problema del agente viajero también se puede modelar como un problema de este tipo y en particular como uno de programación entera binario.

Una vez definido y modelado el problema se demuestra que éste pertenece a la clase de problemas $NP-completos$, lo cual significa que es un problema difícil de resolver. Por último se discuten las diferentes alternativas que se pueden tomar, para resolver un problema, al saber que éste pertenece a la clase $NP-completa$.

## 1.1 Conceptos Básicos de Teoría de Gráficas.

En esta sección se definen algunos conceptos de teoría de gráficas, los cuales se utilizan a lo largo del trabajo.

Una *gráfica* $G$ es una pareja $(V, E)$, donde $V$ es un conjunto finito de *nodos o vértices* y los elementos de $E$ son subconjuntos de $V$ de cardinalidad dos, llamados *aristas*. Una gráfica se denota como $G = (V, E)$ y se representa con puntos asociados a los vértices y líneas asociadas con las aristas. Los vértices de $V$ se denotan como $v_1, v_2, ..., v_k$. Una *gráfica dirigida o digráfica* es una gráfica con direcciones asignadas a las aristas. Es decir, una digráfica $D$

es una pareja $(V, A)$ donde $V$ es un conjunto de vértices y $A$ es un conjunto de parejas ordenadas de vértices llamados *arcos*. La gráfica

$$G = (\{v_1, v_2, v_3, v_4\}, \{[v_1, v_2], [v_2, v_3], [v_3, v_4], [v_4, v_1], [v_1, v_3]\})$$

y la digráfica

$$D = (\{v_1, v_2, v_3, v_4\}, \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1), (v_1, v_3)\})$$

se muestra en la figura 1.1.

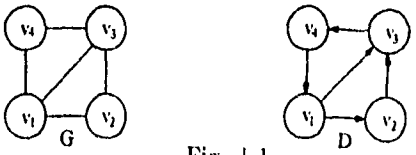



Fig. 1.1

Obsérvese que se usan corchetes para denotar las aristas y paréntesis para denotar los arcos.

Si $G = (V, E)$ es una gráfica y $e = [v_1, v_2]$ está en $E$, entonces se dice que $v_1$ es *adyacente* a $v_2$ ( y viceversa) y $e$ es *incidente* tanto a $v_1$ como a $v_2$. Una gráfica es *completa* si cada par de vértices son adyacentes. El *grado* de un vértice $v$ en una gráfica $G$ es el número de aristas incidentes a $v$. Por ejemplo; en la gráfica de la figura 1.1, el vértice $v_3$ de la gráfica $G$ tiene grado 3.

Una *gráfica parcial* de $G = (V, E)$ es la gráfica $G_p = (V, E_p)$ donde $E_p \subset E$. Es decir, es una gráfica constituida por todos los vértices y algunas aristas de $G$. En la figura 1.2 se muestra una gráfica $G_p$ que es gráfica parcial de $G$.

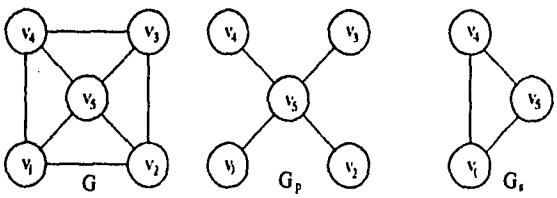



Fig. 1.2

Una subgráfica de $G$ es una gráfica $G_s = (V_s, E_s)$ donde $V_s \subset V$ y $[v_i, v_j] \in E_s$ si y sólo si $v_i, v_j \in V_s$ y $[v_i, v_j] \in E$. Es decir, es una gráfica formada por un subconjunto de vértices de $G$ y todas las aristas de $G$ que unen los vértices de este subconjunto. En la figura 1.2, $G_s$ representa una subgráfica de $G$.

Una *cadena*, en una gráfica $G$, es una sucesión de vértices

$$W = [v_1, v_2, ..., v_k]$$

$k \geq 1$, tal que $[v_j, v_{j+1}] \in E$ para $j = 1, ..., k-1$. A $W$ se le conoce como cadena de $v_1$ a $v_k$. Un *ciclo* es una cadena $W$ donde $v_1 = v_k$. Considérese la gráfica $G$ de la figura 1.1, $[v_1]$, $[v_1, v_3, v_1, v_2, v_3]$ y $[v_3, v_1, v_2, v_3]$ son ejemplos de cadenas; la última es un ciclo. Una gráfica es *conexa* si existe una cadena entre cualquier par de vértices de la gráfica.

Si la gráfica es dirigida, la sucesión de vértices $W$, donde $(v_j, v_{j+1}) \in A$, para $j = 1, ..., k-1$, se conoce como *camino* de $v_1$ a $v_k$. Un *circuito* es un camino cerrado, es decir un camino $W$ donde $v_1 = v_k$. El camino $[v_1, v_3, v_4, v_1]$ de la gráfica $D$, en la figura 1.1, es un circuito. Un *circuito hamiltoniano* de una gráfica $G$ es un circuito que incluye todos los vértices de $G$. En el contexto del problema del agente viajero a este circuito tambi'en se le conoce como *tour*. Una gráfica es hamiltoniana si contiene un circuito hamiltoniano. La gráfica $D$ de la figura 1.1 contiene el circuito hamiltoniano $v_1, v_2, v_3, v_1, v_1$.

Un *camino elemental* es un camino en donde no se repiten los vértices. Si los arcos del camino no se repiten se tiene un *camino simple*. Análogamente se define *cadena simple* y *cadena elemental*.

Los conceptos tratados a continuación se aplican tanto a gráficas como a digráficas. Un *árbol* $T = (V_T, E_T)$ es una gráfica conexa sin ciclos. En la figura 1.3, la gráfica $T$ constituye un *árbol expandido* de $G$. Es decir, un árbol formado con todos los vértices de $G$.

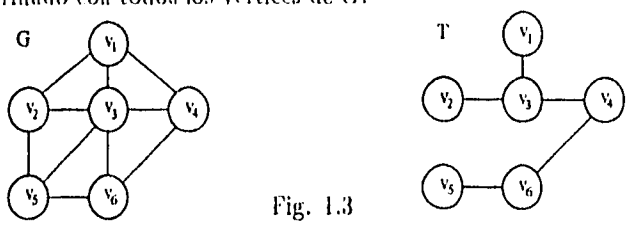



Fig. 1.3

Sea $T = (V_T, E_T)$ una gráfica. Las siguientes afirmaciones son equivalentes:

1. $T$ es un árbol
2. $T$ es conexo y tiene $| V_T | - 1$ aristas
3. $T$ no tiene ciclos, pero si se agrega una arista a $T$ se genera un único ciclo.

Una *gráfica ponderada* es una gráfica $G = (V, E)$ junto con una función de pesos $W$ que va de $E$ a $\mathbb{R}^+$. Los pesos representan costos o distancias.

### 1.1.1 El Problema del Agente Viajero.

Los conceptos presentados con anterioridad permiten enunciar el problema del agente viajero. Supóngase que un agente viajero necesita hacer un viaje redondo a través de una colección de $p\,(\geq 3)$ ciudades. ¿Qué ruta debe elegir para minimizar la distancia total recorrida? Este problema puede representarse gráficamente. Supóngase que $G$ es una digráfica ponderada conexa cuyos vértices $v_i$ $(1 \leq i \leq p)$ representan ciudades, y $c_{ij}$ es el peso del arco $(v_i, v_j)$ que representa la distancia que hay entre las ciudades $v_i$ y $v_j$. El problema del agente viajero pregunta por el circuito Hamiltoniano de menor peso. Si la gráfica no es dirigida, es decir, la distancia $c_{ij}$ no depende de la dirección en la que se viaje, entonces el problema se conoce como *problema simétrico del agente viajero.*

## 1.2 Programación Matemática.

El problema de *programación matemática* consiste en elegir la mejor opción de un conjunto de parámetros en presencia de ciertas restricciones para alcanzar un fin determinado. De manera abstracta el problema de programación matemática es el siguiente:

$$\begin{aligned} &\min(\max) f(x) \\ &\text{s.a} \\ &\qquad g_i(x) \geq b_i \quad \forall i = 1, ..., m \\ &\qquad h_j(x) = c_j \quad \forall j = 1, ..., n \end{aligned}$$

donde $x$ es un vector de variables de decisión, y $f(x)$, $g_i(x)$, y $h_j(x)$ son funciones generales.

Existen muchas clases específicas de este problema, las cuales se obtienen al poner restricciones sobre las funciones bajo consideración y sobre los valores que puedan tomar las variables de decisión. En general, estos problemas se pueden dividir en dos categorías: aquéllos con variables de decisión continuas y aquéllos con variables discretas, los últimos conocidos como *problemas combinatorios.*

Dentro de los problemas continuos el que mejor se ha resuelto es el problema de *programación lineal* $(PL)$, en el cual $f(x)$, $g_i(x)$ y $h_j(x)$ son funciones lineales de las variables de decisión:

$$\min(\max) \sum_{j=1}^{n} c_j x_j$$

s.a

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \forall i = 1, ..., m$$
$$x_j \geq 0 \quad \forall j = 1, ..., n$$

Este problema se puede escribir en forma matricial como:

$$\min cx$$

s.a

$$Ax \geq b$$
$$x \geq 0$$

donde $A \in M_{m \times n}$ de entradas $a_{ij}$ con $i = 1, ..., m$ y $j = 1, ..., n$.

En los problemas combinatorios se busca un objeto de un conjunto finito, o infinito numerable, como un entero, un conjunto, una permutación, o una gráfica. A continuación se darán ejemplos de problemas combinatorios.

- **Problema de Asignación.** Un conjunto de $n$ personas están disponibles para realizar $n$ tareas. Si la persona $i$ realiza la tarea $j$, se genera un costo de $c_{ij}$ unidades. El problema consiste en encontrar una asignación $\{\pi_1, ..., \pi_n\}$ que minimice $\sum_{i=1}^{n} c_{i\pi_i}$, donde $\pi_i$ es la tarea realizada por la persona $i$. Aquí, la solución está representada por la permutación $\{\pi_1, ..., \pi_n\}$ de los números $\{1, ..., n\}$.

- **El Problema de la Mochila 0-1.** Un conjunto de $n$ artículos están disponibles para colocarse en una mochila con capacidad de $C$ unidades. El artículo $i$ tiene un valor de $v_i$ unidades y ocupa $c_i$ unidades de capacidad. El problema consiste en determinar el subconjunto $I$ de artículos que deberán incluirse en la mochila para maximizar $\sum_{i \in I} v_i$ sujeto a $\sum_{i \in I} c_i \leq C$. Aquí, la solución está dada por el subconjunto $I \subseteq \{1, ..., n\}$.

- **El Problema de Recubrimiento.** Una familia de $m$ subconjuntos contiene $n$ artículos, de tal manera que el subconjunto $S_i$ contiene $n_i$ $(\leq n)$ artículos. El problema consiste en seleccionar $k$ $(< m)$ subconjuntos $\{S_{i_1}, ..., S_{i_k}\}$ tales que $| \cup_{j=1}^{k} S_{i_j} | = n$ y que minimice $\sum_{j=1}^{k} c_{i_j}$, donde $c_i$ es el costo de seleccionar el subconjunto $S_i$. En este caso, la solución está representada por una familia de subconjuntos $\{ S_{i_1}, ..., S_{i_k}\}$.

- **El Problema del Árbol Expandido de Peso Mínimo.** Dada una gráfica $G$ no dirigida con pesos específicos en las aristas, determinar el subconjunto de aristas que formen una gráfica conexa acíclica que tenga al menos una arista incidente con cada vértice de $G$, y que minimice el peso total de las aristas. Aquí, la solución está dada por una subgráfica de $G$.

Muchos problemas combinatorios se pueden formular como problemas de *programación entera* ($PE$), en donde las variables de decisión toman valores enteros. Por ejemplo, en el caso del problema de la mochila $0-1$ se definen las variables de decisión como

$$x_i = \begin{cases} 1 & \text{si el artículo } i \text{ se incluye} \\ 0 & \text{en otro caso} \end{cases} \qquad i = 1, ..., n$$

y el problema se reduce al siguiente problema de programación entera binario:

$$\begin{aligned} \max & \sum_{i=1}^{n} x_i v_i \\ \text{s.a} & \\ & \sum_{i=1}^{n} x_i c_i \leq C \\ & x_i = 0, 1 \quad \forall i = 1, ..., n \end{aligned}$$

En general, es difícil resolver el problema de programación entera. Sin embargo, en muchos problemas combinatorios que se plantean de esta manera es posible usar la técnica de relajación Lagrangeana con el propósito de generar cotas inferiores para la solución óptima. La importancia de estas cotas se discutirá en el capítulo 2.

### 1.2.1 Formulación del Problema del Agente Viajero.

A continuación se formulará el problema del agente viajero como un problema de programación entera binario. Se definen las variables de decisión de la siguiente manera:

$$x_{ij} = \begin{cases} 1 & \text{si se viaja de la ciudad } i \text{ a la ciudad } j \\ 0 & \text{en otro caso} \end{cases}$$
$$\text{para} \quad i, j = 1, ..., n;\ i \neq j$$

Las restricciones son las siguientes:

1. Se debe entrar en cada ciudad exactamente una vez. Es decir,

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1,...,n \quad i \neq j \tag{1.1}$$

2. Se debe salir de cada ciudad exactamente una vez. Es decir,

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1,...,n \quad i \neq j \tag{1.2}$$

3. No se permiten *subtours* o *loops* como se muestra en la figura 1.4. Se mostrará que esta restricción puede escribirse de la siguiente manera:

$$a_i - a_j + nx_{ij} \leq n - 1 \quad \text{donde } a_i, a_j \in \mathbb{R} \quad 2 \leq i \neq j \leq n \tag{1.3}$$

Se mostrará que si $(x, a)$ es una solución factible para el problema del agente viajero, entonces $x$ es la asignación de un *tour*. Esto es equivalente a demostrar que si $x$ no es la asignación de un *tour*, entonces $x$ no satisface las restricciones del problema del agente viajero. Sea $i_1,...,i_k = i_1$ un *subtour* que no contiene al vértice 1, es decir $i_j \neq 1$ $\forall 1 \leq j \leq k$. Supóngase que este *subtour* es una solución factible para el problema del agente viajero. Entonces, del conjunto de restricciones (1.3) se tiene que:

$$\begin{aligned} a_{i_1} - a_{i_2} + n &\leq n - 1 \\ a_{i_2} - a_{i_3} + n &\leq n - 1 \\ &\vdots \\ a_{i_{k-1}} - a_{i_k} + n &\leq n - 1 \end{aligned}$$

si se suman estas restricciones se tiene:

$$(k-1)n \leq (k-1)(n-1)$$

lo cual es una contradicción. Por lo tanto, si $x$ forma *subtours* entonces no satisface el conjunto de restricciones (1.3) y por lo tanto no es una solución factible para el problema del agente viajero.
Ahora, se mostrará que si $x$ es la asignación de un *tour* que satisface

el conjunto de restricciones (1.1) y (1.2) entonces existe $a \in \mathbb{R}^n$, tal que $(x, a)$ es una solución factible para el problema del agente viajero. Se define

$$a_r = \begin{cases} k & \text{si la ciudad } r \text{ se visita en } k - \acute{e}simo \text{ lugar} \\ n & \text{si } i = n \end{cases}$$

entonces, si $x_{ij} = 1$ se tiene que

$$a_i - a_j + nx_{ij} = k - (k+1) + n = n - 1$$

por otro lado, si $x_{ij} = 0$ se tiene que

$$a_i - a_j + nx_{ij} = a_i - a_j \leq n - 2 < n - 1$$

Por lo tanto, $a_i - a_j + nx_{ij} \leq n - 1 \quad \forall x_{ij}, 2 \leq i \neq j \leq n$. Entonces, $(x, a)$ es una solución factible para el problema del agente viajero.

De acuerdo a lo anterior, el problema del agente viajero se puede plantear de la siguiente manera:

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

$$s.a$$

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &= 1 \quad \forall i = 1, \cdots, n. \quad i \neq j \\ \sum_{i=1}^{n} x_{ij} &= 1 \quad \forall j = 1, \cdots, n \quad i \neq j \\ a_i - a_j + nx_{ij} &\leq n - 1 \quad 2 \leq i \neq j \leq n, \quad a_i, a_j \in \mathbb{R} \\ x_{ij} &= 0, 1 \quad \forall 1 \leq i \neq j \leq n \end{aligned}$$

El conjunto de restricciones (1.3) que impide la formación de *subtours*, en la solución óptima del problema del agente viajero, se puede formular de otra manera que posteriormente será de mayor utilidad. Sea $S$ cualquier subconjunto propio de $V$. Si las aristas correspondientes a $x_{ij} = 1$ con ambos extremos en $S$ son menos que $| S |$ entonces no se forman *subtours*, es decir a lo más debe haber $| S | - 1$ aristas. Por lo tanto, para eliminar *subtours* se debe cumplir la siguiente desigualdad:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq | S | - 1 \quad \forall S \subset V, S \neq \emptyset \tag{1.4}$$

Por ejemplo, considérese la gráfica de la figura 1.4.

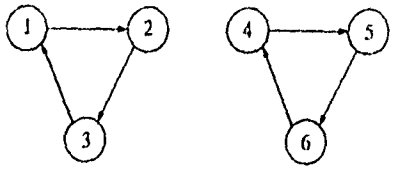



Fig. 1.4

Sea $S = \{1,2,3\}$ ( o $S = \{4,5,6\}$ ). El lado izquierdo de la desigualdad (1.4) es 3, mientras que $\mid S \mid -1 = 2$. Por lo tanto la desigualdad no se cumple y como consecuencia se forman *subtours*. La formulación del problema del agente viajero en este caso es:

$$\min Z = \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$$

*s.a*

$$\begin{aligned}
\sum_{j \in V} x_{ij} &= 1, i \in V \\
\sum_{i \in V} x_{ij} &= 1, j \in V \\
\sum_{i \in S} \sum_{j \in S} x_{ij} &\leq \mid S \mid -1, \forall S \subset V, S \neq \emptyset \\
x_{ij} &= 0,1 \quad \forall i, j \in V
\end{aligned}$$

Si se considera el problema simétrica del agente viajero, donde $c_{ij} = c_{ji} \quad \forall i, j \in V$ , la formulación es la siguiente:

$$\min Z = \sum_{i \in V} \sum_{j > i} c_{ij} x_{ij}$$

*s.a*

$$\begin{aligned}
\sum_{j < i} x_{ij} + \sum_{j > i} x_{ij} &= 2, i \in V \\
\sum_{i \in S} \sum_{j \in S, j > i} x_{ij} &\leq \mid S \mid -1, \forall S \subset V, S \neq \emptyset \\
x_{ij} &= 0,1 \quad \forall i, j \in V, j > i
\end{aligned}$$

## 1.3 Teoría de NP-Completez.

Existen muchos problemas que no se pueden resolver con las técnicas disponibles de manera exacta y eficiente. Algunos de estos problemas podrían ser resueltos con algoritmos eficientes que aún no se descubren. Sin embargo, es muy probable que muchos de ellos no puedan ser resueltos eficientemente. Entonces, es de gran utilidad identificar este tipo de problemas con el propósito de no invertir tiempo en buscar algoritmos que no existen. La teoría de NP-completez proporciona técnicas para identificar este tipo de problemas.

A continuación se definen algunos conceptos que se utilizarán más adelante. Un *problema* se especifica con la descripción general de sus parámetros y las propiedades que debe tener la solución. Como ejemplo considérese el problema del agente viajero. Los parámetros de este problema son las ciudades $1, 2, ..., n$ y para cada par de ciudades $i, j$ la distancia $c_{ij}$ entre ellas. La solución es una permutación $(\pi_1, \pi_2, ..., \pi_n)$ de ciudades que minimicen $\sum_{i=1}^{n-1} c_{\pi_i \pi_{i+1}} + c_{\pi_m \pi_1}$.

Un *ejemplo* de un problema se obtiene al especificar los valores de todos los parámetros del problema. Por ejemplo, un *ejemplo* para el problema del agente viajero está dada por $\{1,2,3,4\}, c_{12} = 10, c_{13} = 5, c_{14} = 9, c_{23} = 6, c_{24} = 9$ y $c_{34} = 3$. La permutación $(1,2,4,3)$ es una solución de este *ejemplo* con costo total de 23 unidades.

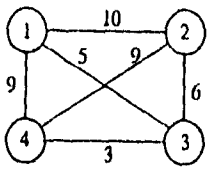



Fig.1.5 *ejemplo* de problema del agente viajero

Un *algoritmo* es un conjunto de instrucciones o reglas bien definidas que se usan para obtener un resultado específico a partir de unos datos de entrada específicos en un número finito de pasos.

Se está interesado en encontrar el algoritmo más "eficiente" que resuelva un problema. La noción de eficiencia involucra todos los recursos de cómputo que se necesitan para ejecutar un algoritmo. Sin embargo, el algoritmo "más eficiente" generalmente significa el más rápido. El tiempo requerido por un algoritmo depende del tamaño del *ejemplo* del problema. El tamaño refleja la cantidad de datos de entrada necesarios para describir el *ejemplo*. Generalmente esta medida se hace de manera informal. Por ejemplo, para el

problema del agente viajero el número de ciudades se toma como el tamaño del *ejemplo* aunque existan otros datos de entrada como la distancia entre cada par de ciudades.

La *función de complejidad de tiempo* proporciona el mayor tiempo requerido por un algoritmo para resolver un *ejemplo* de un problema con un determinado tamaño. Los algoritmos tienen una gran variedad de funciones de complejidad de tiempo y determinar cuáles algoritmos son suficientemente eficientes y cuáles son muy ineficientes depende de cada situación. Sin embargo, éstos se han dividido en algoritmos de tiempo polinomial y algoritmos de tiempo exponencial.

Se dice que una función $f(n)$ es de orden $g$, $O(g(n))$, si existe una constante $c$ tal que $\mid f(n) \mid < c \mid g(n) \mid$ para $n \geq 0$. Un *algoritmo de tiempo polinomial* se define como aquél cuya función de complejidad de tiempo sea $O(p(n))$ para alguna función polinomial $p$ y donde $n$ es el tamaño del *ejemplo* del problema. Los problemas de la clase $P$ son aquéllos para los cuales existe un algoritmo polinomial que los resuelva. Cualquier algoritmo cuya función de complejidad de tiempo no pueda ser acotada de esta manera se conoce como *algoritmo exponencial*. Cabe hacer notar que esta definición incluye ciertas funciones no polinomiales, como $n^{\log n}$, que no se consideran como funciones exponenciales. Un ejemplo de un algoritmo exponencial para el problema del agente viajero consiste en enumerar todas las posibles soluciones para determinar cuál es la óptima. Un *ejemplo* del problema del agente viajero con $N$ ciudades tiene $(N-1)!$ posibles soluciones ( si el problema es simétrico tiene $(N-1)!/2$ ). Supóngase que se tiene una computadora que lista todas las posibles soluciones de un problema con 20 ciudades en una hora. Usando esta fórmula, se requieren 20 horas para resolver un problema con 21 ciudades, 17.5 días para resolver un problema de 22 ciudades y aproximadamente 6 siglos para uno de 25 ciudades.

Sin embargo, hay algoritmos exponenciales que son muy útiles en la práctica, ya que el tiempo de complejidad se define como una medida del peor caso, por lo que muchas instancias de un problema pueden necesitar menos tiempo. Por ejemplo, el algoritmo simplex para resolver problemas de programación lineal es un algoritmo exponencial, pero en la práctica es muy rápido. Otro ejemplo es el algoritmo de ramificación y acotamiento para el problema de la mochila. Sin embargo estos ejemplos son raros. Por otro lado, aunque un algoritmo tenga tiempo de complejidad $n^{100}$ ó $10^{99}n^2$ no se pueden considerar eficientes. Pero en la práctica los problemas que se resuelven en tiempo polinomial están acotados por polinomios de grado 2 ó 3 y no involucran coeficientes grandes. Entonces, se dice que un problema

es *insoluble* si no existe ningún algoritmo polinomial que lo resuelva.

### 1.3.1 Clase NP, NP-completa y CoNp.

La teoría de NP-completez proporciona técnicas para demostrar que un problema es tan "difícil" como un gran número de problemas que son conocidos como "difíciles". La principal técnica que se usa para demostrar que dos problemas están relacionados es "reducir" un problema en otro al transformar cada *ejemplo* de un problema en un *ejemplo* equivalente de otro problema. Para ello se considerarán problemas de decisión, es decir, problemas en donde la respuesta es *si* o *no*. El objetivo de restringirse a estos problemas es hacer la teoría más simple. Muchos problemas combinatorios se pueden transformar en problemas de decisión. Por ejemplo, el problema de decisión para el problema del agente viajero es :

*Dada una gráfica $G = (V, E)$ con pesos $cij$ en las aristas $[i, j]$ y $C$ un número fijo, ¿existe un circuito Hamiltoniano en $G$ de peso menor o igual a $C$?*

Se introducirá el concepto de reducción polinomial. Supóngase que se tiene un problema $\pi_1$ que puede ser resuelto por un algoritmo $A$. Si cada instancia de un problema $\pi_2$ se puede transformar en un *ejemplo* de $\pi_1$ en un tiempo polinomial, entonces claramente se puede usar este hecho y el algoritmo $A$ para resolver $\pi_2$. Si el número de instancias del problema $\pi_1$ es mayor o igual que el número de instancias transformadas del problema $\pi_2$, éstas se pueden ver como casos particulares de las instancias de $\pi_1$. Entonces, es razonable afirmar que el problema $\pi_1$ es tan difícil (o posiblemente más difícil) que $\pi_2$.

Los problemas de decisión pueden tener diferentes grados de dificultad. Pero si se tuviera una solución candidata, sería fácil verificar si con ésta se demuestra que la respuesta al problema de decisión es *sí*. Para los problemas de decisión que están en $NP$ no se requiere que cada *ejemplo* del problema pueda ser resuelto en un tiempo polinomial por algún algoritmo. Sólo se requiere que para las instancias del problema en la cual la respuesta al problema de decisión es *sí*, exista una solución con la que se pueda verificar la respuesta en tiempo polinomial. Específicamente, la *clase $NP$* incluye aquellos problemas de decisión que pueden ser resueltos en tiempo polinomial si se "adivina" la solución con la que se puede demostrar que el problema de decisión es *sí*. Las letras $NP$ significan *polinomial no determinístico*. Es decir, los problemas en $NP$ se resuelven en tiempo polinomial no determinístico en el sentido de que se pueden generar soluciones candidatas con

una alta probabilidad de adivinar alguna que sirva para demostrar que la respuesta al problema de decisión es *sí*.

El complemento de un problema de decisión se obtiene al intercambiar los papeles de las respuestas *sí* y *no*. Por ejemplo, el complemento del problema de decisión para el problema del agente viajero es el siguiente:

*Dada una gráfica* $G = (V, E)$ *con pesos* $cij$ *en las aristas* $[i, j]$ *y* $C$ *un número fijo, ¿no existe ningún circuito Hamiltoniano en* $G$ *de peso menor o igual a* $C$*?*

Obsérvese que las instancias para las cuales la respuesta al problema de decisión es *sí* tienen como respuesta *no* cuando se considera el complemento del problema de decisión.

El complemento de los problemas de la clase $NP$ pertenecen a la clase $CoNP$. ¿Cómo se pueden resolver problemas que pertenecen a esta clase? Es decir, ¿cómo se puede probar que la respuesta al problema de decisión es no? La única manera es enunciar todas las posibles soluciones y verificar que con ninguna de éstas se puede demostrar que la respuesta al problema de decisión es *sí*. Dado que esta enumeración se realiza en tiempo exponencial no se puede verificar que la respuesta es no en tiempo polinomial, es decir, existen problemas en $CoNP$ que no están en $NP$.

Si un problema $\pi$ es tal que cualquier problema en $NP$ se puede reducir polinomialmente a $\pi$, se dice que $\pi$ es $NP - difícil$. Si además el problema $\pi$ pertenence a la clase $NP$, entonces $\pi$ es $NP - completo$. Por lo tanto, los problemas de clase $NP - completa$ son los más difíciles de la clase $NP$. Los problemas $NP - completos$ son importantes ya que si se encuentra un algoritmo polinomial para resolver alguno de ellos se tendrá un algoritmo polinomial para todos los problemas en $NP$; es decir se habrá mostrado que $P = NP$.

Puede parecer sorprendente que exista un problema para el cual cualquier problema en $NP$ se puede reducir a él. Sin embargo, Cook demostró en 1971 que el problema de satisfacibilidad (SAT) es $NP - completo$, es decir, demostró que la clase $NP - completa$ no es vacía. El *problema de satisfacibilidad* es el siguiente. Sea $S$ una expresión lógica que está formada por el producto de varias sumas. Por ejemplo, $S = (x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_1} + x_2 + x_3) \cdot (\overline{x_1} + \overline{x_2} + \overline{x_3})$, donde las sumas y las multiplicaciones corresponden a las operaciones lógicas *y* y *o* respectivamente y donde cada variable vale 0 (falso) ó 1 (verdadero). El complemento de la variable $x_i$ se denota por

$\overline{x_i}$. Se dice que una expresión lógica se satisface si existe una asignación de ceros y unos de las variables tal que el valor de la expresión sea 1. El problema SAT consiste en determinar si una expresión lógica se satisface. Por ejemplo, la expresión $S$ sí se satisface lo cual puede verificarse con la siguiente asignación: $x_1 = 1, x_2 = 1$ y $x_3 = 0$.

En 1972 Karp demostró que existen otros problemas que pertenecen a la clase $NP - completa$. Una vez que se ha encontrado un problema $NP - completo$ es más fácil demostrar que otros problemas también pertenencen a esta clase usando la siguiente observación: un problema $\pi_1$ es $NP - completo$ si:

1. $\pi_1$ pertenece a $NP$ y
2. $\pi_2$ se reduce polinomialmente a $\pi_1$, para algún problema $\pi_2$ que pertenezca a la clase $NP - completa$.

### 1.3.2 El Problema del Agente Viajero es NP-Completo.

Sabiendo que el SAT es $NP - completo$ se realizarán las transformaciones mostradas en el siguiente esquema para demostrar que el problema del agente viajero es $NP - completo$.

SAT $\rightarrow$ *clique* $\rightarrow$ recubrimiento de vértices $\rightarrow$
circuito hamiltoniano $\rightarrow$ problema del agente viajero

**Transformación del SAT al *Clique*.**

Dada una gráfica no dirigida $G = (V, E)$, un *clique* $C$ en $G$ es una subgráfica de $G$ tal que cada vértice de $C$ está conectado con los demás vértices de $C$. Es decir, un *clique* es una subgráfica completa. El problema de decisión del clique consiste en determinar, dada una gráfica $G$ y un entero $k$, si $G$ tiene un *clique* de tamaño mayor o igual a $k$.

El problema del *clique* pertenece a la clase $NP$ ya que si se adivina un subconjunto de $k$ o más vértices se puede verificar que sea un *clique* en tiempo polinomial.

A continuación se describirá la transformación. Sea $E = E_1 \cdot E_2 \cdots E_m$ una expresión lógica y sea $E_1 = (x + y + z + w)$ una cláusula (se usan 4 variables para ejemplificar el procedimiento). Se asocia una columna de 4 vértices con las variables en $E_i$, aunque alguna variable se repita en otra cláusula. Es decir, la gráfica $G$ tiene un vértice por cada aparición de una variable. Las aristas de $G$ se eligen de acuerdo a las siguientes reglas.

- Los vértices de la misma columna, es decir los vértices asociados con las variables de la misma cláusula, no se conectan.

- Los vértices de diferentes columnas se conectan a menos de que estén asociados con la misma variable en forma complementaria. Es decir, no se conectan dos vértices de cláusulas diferentes si uno está asociado con la variable $x$ y el otro con la variable $\overline{x}$. Por ejemplo, la gráfica asociada con la expresión lógica $E = (x + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z) \cdot (y + \overline{z})$ se muestra en la figura 1.6

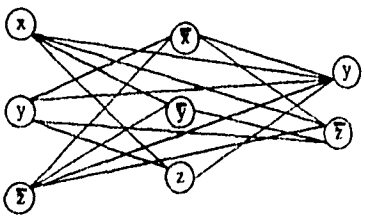



Fig. 1.6 Gráfica $G$

Es claro que esta transformación se puede realizar en tiempo polinomial. Ahora se mostrará que $G$ tiene un *clique* de tamaño mayor o igual a $k$ si y sólo si $E$ se satisface. Sea $k$ igual al número de cláusulas $m$. De hecho la construcción de $G$ garantiza que el *clique* de tamaño máximo no excede de $m$. Supóngase que $E$ se satisface. Entonces existe una asignación verdadera donde cada cláusula tiene al menos una variable cuyo valor es uno. Se elige el vértice que está asociado a esta variable para formar el *clique*. Si hay más de una variable que vale uno en una cláusula, se elige una arbitrariamente. El resultado es un *clique*, ya que dos vértices de diferentes columnas no están conectados solamente cuando uno es el complemento del otro y las variables correspondientes no pueden valer ambas uno en una asignación consistente. Inversamente, supóngase que $G$ contiene un *clique* de tamaño mayor o igual a $m$. El *clique* debe tener exactamente un vértice de cada columna ya que los vértices de una misma columna nunca están conectados. Las variables asociadas a dichos vértices se les asocia el valor de uno. Si alguna variable no ha sido asignada de esta manera entonces se le da un valor arbitrariamente. Dado que los vértices en el *clique* están conectados y los vértices asociados con $x$ y $\overline{x}$ no se conectan, esta asignación verdadera es consistente. Por lo tanto, se ha mostrado que el problema del *clique* es $NP - completo$.

**Transformación del Problema del *Clique* al Problema del Recubrimiento de Vértices.**

Sea $G = (V, E)$ una gráfica no dirigida. Un recubrimiento de vértices de $G$ es un conjunto de vértices tal que cada arista de $G$ es incidente con al menos uno de estos vértices. El problema de decisión es el siguiente. Dada una gráfica no dirigida $G = (V, E)$ y un entero $k$, determinar si $G$ tiene un recubrimiento que consiste de $k$ o menos vértices.

El problema de recubrimiento de vértices es $NP$ ya que si se adivina un recubrimiento de tamaño menor o igual que $k$ se puede verificar en tiempo polinomial que la respuesta al problema de decisión correspondiente es sí.

El problema del ***clique*** se reduce al problema de recubrimiento de vértices mediante el complemento $\overline{G} = (V, \overline{E})$ de la gráfica $G$. $\overline{G}$ tiene los mismos vértices de $G$ y dos vértices de $\overline{G}$ están conectados si y sólo si no lo están en $G$. Es claro que esta transformación se puede realizar en tiempo polinomial.

Se mostrará que si $G$ tiene un *clique* de tamaño $k$ entonces $\overline{G}$ tiene un recubrimiento de vértices de tamaño $n - k$, donde $n = \mid V \mid$. Supóngase que $C = (U, F)$ es un *clique* de $G$. El conjunto de vértices $V - U$ cubre todas las aristas de $\overline{G}$, ya que en $\overline{G}$ no hay aristas que conecten los vértices de $U$. Entonces $V - U$ es un recubrimiento de $\overline{G}$. Si $G$ tiene un *clique* de tamaño $k$ entonces $\overline{G}$ tiene un recubrimiento de tamaño $n - k$. Inversamente, sea $D$ un recubrimiento de vértices de $\overline{G}$. Como $D$ cubre todas las aristas de $\overline{G}$, no puede haber aristas de $\overline{G}$ que conecten vértices de $V - D$. Entonces $V - D$ genera un *clique* en $G$. Por lo tanto, si existe un recubrimiento de tamaño $k$ en $\overline{G}$, existe un *clique* de tamaño $n - k$ en $G$. Entonces, el problema de recubrimiento de vértices es $NP - completo$.

**Transformación del Problema del Recubrimiento de Vértices en el Problema del Circuito Hamiltoniano.**

El problema del circuito hamiltoniano pertenece a la clase $NP$ ya que si se adivina un ordenamiento de los vértices se puede verificar en tiempo polinomial que todas las aristas requeridas para formar el circuito hamiltoniano pertenecen al conjunto de aristas de la gráfica.

Considérese un *ejemplo* del problema de recubrimiento dada por la gráfica $G = (V, E)$ y sea $k$ un entero positivo tal que $k \leq \mid V \mid$. Se debe construir una gráfica $G' = (V', E')$ tal que $G'$ tenga un circuito hamiltoniano si y sólo si $G$ tiene un recubrimiento de vértices de tamaño menor o igual a $k$.

La gráfica $G'$ tiene $k$ vértices selectores $a_1, a_2, ..., a_k$ que se usarán para

seleccionar $k$ vértices del conjunto $V$ de $G$. Además, $G'$ está formada por componentes enlazadas. Para cada arista en $E$, $G'$ contiene una componente que servirá para asegurar que al menos uno de los vértices que forman esta arista está dentro de los $K$ vértices seleccionados. La componente para $e = [u, v]$ de $E$ se ilustra en la figura 1.7.

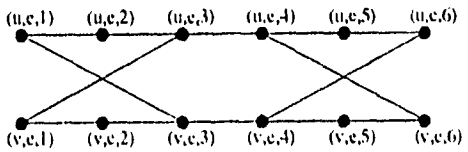



Fig. 1.7 Componente para la arista $e = [u, v]$

Cada componente tiene 12 vértices,

$$V'_e = \{(u,e,i),(v,e,i); 1 \leq i \leq 6\}$$

y 14 aristas,

$$\begin{aligned} E'_e = & \{\{(u,e,i),(u,e,i+1)\},\{(v,e,i),(v,e,i+i)\}; 1 \leq i \leq 5\} \\ & \bigcup \{\{(u,e,3),(v,e,1)\},\{(v,e,3),(u,e,1)\}\} \\ & \bigcup \{\{(u,e,6),(v,e,4)\},\{(v,e,6),(u,e,4)\}\} \end{aligned}$$

Los únicos vértices de las componentes que forman parte de las aristas del resto de la gráfica $G'$ son $(u,e,1),(v,e,1),(u,e,6)$ y $(v,e,6)$. Por lo tanto, cualquier circuito hamiltoniano en $G'$ debe recorrer las aristas de $E'_e$ en alguna de las tres formas mostradas en la figura 1.8:

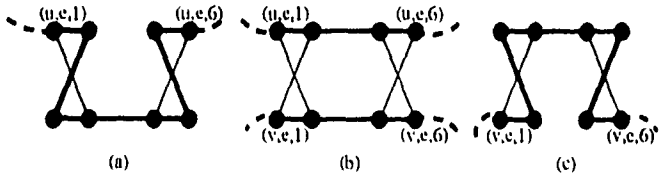



Fig. 1.8 Tres posibles recorridos de las aristas de una componente para $[u,v]$ por el circuito hamiltoniano que corresponden a los siguientes casos: a)$u$ pertenece al recubrimiento pero $v$ no, b) $u$ y $v$ pertenecen al recubrimiento y c) $v$ pertenece al recubrimiento pero $u$ no.

Entonces, si el circuito hamiltoniano entra en esta componente por el vértice $(u, e, 1)$ debe salir por el vértice $(u, e, 6)$ y visitar los 12 vértices del recubrimiento o únicamente los siguientes 6 vértices $(u, e, i)$, $1 \leq i \leq 6$.

Las aristas de $G'$ que no pertenecen a alguna componente sirven para conectar parejas de componentes o para conectar una componente con un vértice selector. Para cada vértice $v$ de $V$ se ordenan arbitrariamente todas las aristas incidentes a $v$: $e_{v[1]}, e_{v[2]}, ..., e_{v[gr(v)]}$, donde $gr(v)$ denota el grado de $v$ en $G$. Todas las componentes asociadas con estas aristas se conectan mediante las siguientes aristas:

$$E'_v = \{\{(v, e_{v[i]}, 6), (v, e_{v[i+1]}, 1)\}; 1 \leq i < gr(v)\}$$

Como se muestra en la figura 1.9, esto genera una única trayectoria en $G'$ que incluye aquellos vértices $(x, y, \cdot)$ con $x = v$.

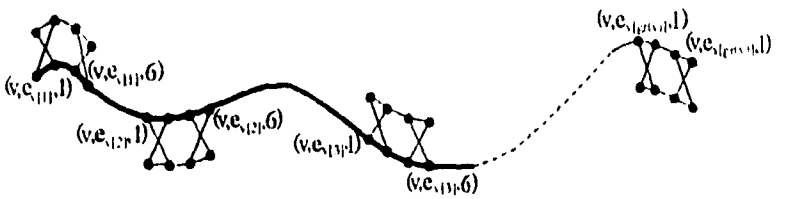



Fig. 1.9

El resto de las aristas de $G'$ unen el primero y el último vértice de estas trayectorias a cada vértice selector $a_1, a_2, ..., a_k$. Estas aristas se especifican de la siguiente manera:

$$E'' = \{\{a_i, (v, e_{v[1]}, 1)\}, \{a_i, (v, e_{v[gr(v)]}, 6)\}, 1 \leq i \leq k, v \in V\}$$

Entonces, la gráfica $G'$ tiene el siguiente conjunto de vértices:

$$V' = \{a_i; 1 \leq i \leq k\} \cup \left(\bigcup_{e \in E} V'_e\right)$$

y el siguiente conjunto de aristas:

$$E' = \left(\bigcup_{e \in E} E'_e\right) \cup \left(\bigcup_{v \in V} E'_v\right) \cup E''$$

Es fácil ver que $G'$ se puede construir a partir de $G$ en tiempo polinomial.

A continuación se mostrará que si $G'$ tiene un circuito hamiltoniano entonces $G$ tiene un recubrimiento de cardinalidad menor o igual que $k$. Supóngase que $< v_1, v_2, ..., v_n >$, donde $n = \mid V \mid$, es un circuito hamiltoniano de $G'$. Considérese una porción de este circuito que comienza en un vértice del conjunto $\{a_1, a_2, ..., a_k\}$, termina en un vértice del mismo conjunto y no contiene estos vértices en el interior de la trayectoria. Debido a la manera en que el circuito hamiltoniano debe pasar a través de una componente (ver figura 1.8), si el circuito entra en una componente por un vértice $v$ de $V$ entonces sale de esa componente por el mismo vértice $v$ y necesariamente entra a otra componente por $v$. Entonces, esta porción del circuito recorre las componentes asociadas a todas aquellas aristas de $E$ que son incidentes a un vértice particular $v$ de $V$. Por tanto, los $k$ vértices del conjunto $\{a_1, ..., a_k\}$ dividen el circuito hamiltoniano en $k$ trayectorias, donde cada trayectoria corresponde a un vértice diferente de $V$.

El circuito hamiltoniano debe incluir todos los vértices de cada una de las componentes, las cuales están asociadas a una arista $e$ de $E$. Cada componente se recorre por una trayectoria asociada a uno de los vértices de $e$ (caso $a$ o $c$ de la figura 1.8) o por dos trayectorias asociadas con los dos vértices de $e$ (caso $b$). Por lo que cada arista de $e$ debe estar formada al menos por uno de los $k$ vértices seleccionados. Por lo tanto, este conjunto de $k$ vértices forma un recubrimiento de $G$.

Ahora se mostrará que si $G$ tiene un recubrimiento menor o igual a $k$ entonces $G'$ tiene un circuito hamiltoniano. Supóngase que $V^* \subseteq V$ es un recubrimiento de $G$ con $\mid V^* \mid \leq k$. Se puede asumir que $\mid V \mid = k$ puesto que si se agregan más vértices de $V$ también se tendrá un recubrimiento. Los elementos de $V^*$ se etiquetan como $v_1, v_2, ..., v_k$. Se eligen las siguientes aristas para formar un circuito hamiltoniano en $G'$ :

1. Para cada componente de $G'$, asociada a una arista $e = [u, v]$ de $E$, se eligen las aristas especificadas en la figura 1.8 dependiendo de si $u$ está en $V^*$, $v$ está en $V^*$ o $u$ y $v$ están en $V^*$. Se debe cumplir alguna de estas tres posibilidades ya que $V^*$ es un recubrimiento de $G$.

2. Se eligen todas las aristas en $E'_{v_i}$ para $1 \leq i \leq k$ . Es decir, las aristas que unen aquellas componentes asociadas a aristas en $E$ que son incidentes a un mismo vértice $v_i$ de $V^*$.

3. Se eligen las aristas

$$\{a_i, (v_i, e_{v_i[1]}, 1)\}, 1 \leq i \leq k$$
$$\{a_{i+1}, (v_i, e_{v_i[gr(v_i)]}, 6)\}, 1 \leq i < k$$
$$\{a_1, (v_k, e_{v_k[gr(v_k)]}, 6)\}$$

las cuales unen las componentes con los vértices selectores.

Para mostrar que es un circuito hamiltoniano se debe verificar que

- todos los vértices de $G'$ estén incluidos exactamente una vez, lo cual es cierto puesto que en (1) se se incluyen todos los vértices de cada componente y en (3) se incluyen los vértices selectores $a_i$ para $1 \leq i \leq k$. Obsérvese que únicamente se repite el vértice $a_1$, el cual puede ser considerado como el vértice inicial y final del circuito hamiltoniano.
- Para el ordenamiento de vértices de $G'$ todas las aristas requeridas para formar el circuito hamiltoniano pertenecen a $E'$. Esto se cumple ya que el conjunto de aristas de (1) es un subconjunto de $E'_{e_1}$ el conjunto de aristas de (2) es un subconjunto de $E'_v$ y las aristas de (3) son un subconjunto de $E''$.

Se ha mostrado que el problema del circuito hamiltoniano es $NP$ – *completo.*

**Ejemplo 1.3.1** *Se ejemplificará el procedimiento de la transformación del problema de recubrimiento de vértices al problema del circuito hamiltoniano. Para ello considérese un* ejemplo *del problema de recubrimiento representada por la gráfica G de la figura 1.10:*

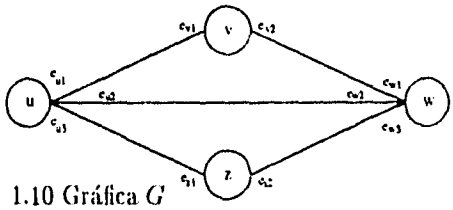



Fig. 1.10 Gráfica $G$

*Se ha etiquetado cada arista de G de dos maneras diferentes, una por cada vértice incidente a ésta. Por ejemplo, la arista [u, z] es etiquetada como $e_{u3}$ y $e_{z1}$ lo cual significa que es la tercera arista incidente al vértice u y la primera incidente a z. Sea $k = 2$.*
*La gráfica $G'$ tiene 5 componentes, puesto que G tiene 5 aristas, y 2 vértices selectores $a_1$ y $a_2$, ya que $k = 2$. Cada vértice está unido por las aristas que se muestran en la figura 1.11.*

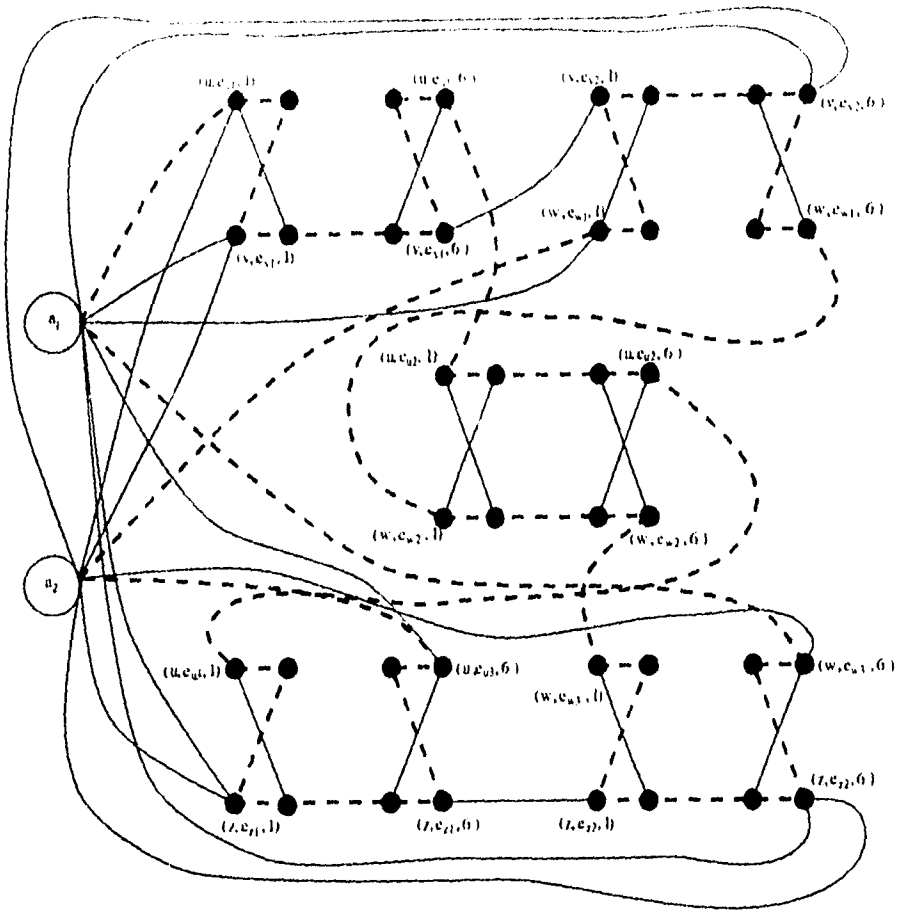


Fig. 1.11 Gráfica $G'$

*La gráfica $G$ tiene un recubrimiento de cardinalidad menor o igual que 2, a saber $\{u, w\}$. Por lo tanto, la gráfica $G'$ tiene un circuito hamiltoniano representada en la figura 1.11 con líneas punteadas.*

**Transformación del Problema Circuito Hamiltoniano en el Problema del Agente Viajero.**

Dado un número positivo $k$ y un tour para un *ejemplo* del problema de decisión, asociado al problema del agente viajero, se puede verificar en tiempo polinomial que la longitud de este tour es menor o igual a $k$. Por lo tanto, el problema de decisión asociado al problema del agente viajero pertenece a la clase $NP$.

Sea $G = (V, E)$, donde $\mid V \mid = n$, un *ejemplo* del problema del circuito hamiltoniano. El correspondiente *ejemplo* para el problema del agente via-

jero $G'$ tiene un conjunto $C$ de ciudades cuyos elementos son los de $V$. La distancia entre cada par de ciudades $v_i, v_j$ en $C$ se define como

$$d(v_i, v_j) = \begin{cases} 1 & \text{si } [v_i, v_j] \in E \\ 2 & \text{en otro caso} \end{cases}$$

Es claro que esta transformación se puede realizar en tiempo polinomial.

Se mostrará que $G$ tiene un circuito hamiltoniano si y sólo si existe un tour que incluye todas las ciudades de $C$ cuya longitud es menor o igual a $k$. Sea $k = n$. Supóngase que $< v_1, v_2, ..., v_n >$ es un circuito hamiltoniano de $G$. Entonces $< v_1, v_2, ..., v_n >$ constituye un tour para $G'$ de longitud $n = k$ ya que la distancia viajada entre cada par de ciudades corresponde a una arista de $G$ y por lo tanto tiene longitud 1. Inversamente, supóngase que $< v_1, v_2, ..., v_n >$ es un tour de $G'$ con longitud menor o igual a $k$. Dado que la distancia entre cada par de ciudades es 1 ó 2 y dado que se suman $n$ distancias para calcular la longitud del tour, el hecho de que $k = n$ implica que la distancia entre cada par de ciudades es 1. Por la manera en como se definió $G'$, las aristas $[v_i, v_{i+1}]$ $1 \leq i < m$ y $[v_m, v_1]$ son aristas de $G$. Entonces $< v_1, v_2, ..., v_n >$ es un circuito hamiltoniano de $G$. Por lo tanto, el problema del agente viajero es un problema $NP - completo$.

## 1.4 Motivación.

Al saber que un problema es $NP-completo$ se pueden tomar varios caminos. Si el problema es pequeño entonces se puede resolver eficientemente con algún algoritmo exacto. Pero si el problema no es pequeño la búsqueda de un algoritmo eficiente y exacto no es prioritario. Es más apropiado concentrarse en metas menos ambiciosas. Por ejemplo, buscar algoritmos eficientes que resuelvan casos particulares del problema general. Buscar algoritmos que aunque no exista garantía de ser eficientes lo sean en la mayoría de los casos. Relajar el problema y buscar algoritmos eficientes para el nuevo problema. Otro enfoque consiste en diseñar algoritmos "heurísticos" eficientes los cuales aunque no garanticen obtener la solución óptima obtienen una cercana a ésta.

# Capítulo 2

# Algoritmos para Problemas Combinatorios.

En este capítulo se describen algunos algoritmos que resuelven cualquier tipo de problemas combinatorios, al cual pertenece el problema del agente viajero. Se discutirán tres tipos de algoritmos. Los primeros obtienen la solución óptima del problema bajo consideración (algoritmos exactos). Con el segundo tipo de algoritmos se obtiene soluciones factibles aproximadas a la óptima (algoritmos heurísticos). Por último, se presentará un algoritmo con el cual se obtienen únicamente cotas inferiores del valor óptimo en el caso de problemas de minimización, es decir este algoritmo no proporciona soluciones factibles.

El capítulo está estructurado de la siguiente manera. En la primera sección se discute la relación que existe entre los algoritmos heurísticos y aquéllos con los cuales únicamente se obtienen cotas inferiores del valor óptimo. Además se explica la importacia de aplicar este último tipo de algoritmo para saber qué tan buena es la solución que se obtiene con los heurísticos. En la segunda sección se presentan los algoritmos exactos y en la última sección se discuten los heurísticos.

## 2.1 Relación entre los Diferentes Tipos de Algoritmos.

Como se vió en el capítulo 1, en ocasiones es conveniente usar algoritmos heurísticos cuando se tienen problemas $NP - Completos$. Sin embargo,

surge el problema de qué tan buena es la solución generada con este tipo de algoritmos. Una manera de evaluar los algoritmos heurísticos consiste en encontrar cotas sobre lo alejada que está la solución generada de la solución óptima en el peor caso o en el caso promedio. Pero este tipo de análisis generalmente es difícil y además no proporciona suficiente información para decidir si la solución generada mediante estos procedimientos es buena para una instancia del problema. Es decir, se desea conocer cómo se comporta un algoritmo heurístico para la instancia del problema en la que se está interesado. La relajación Lagrangeana, que se estudiará más adelante, es un elemento importante para obtener este tipo de información. La relación que existe entre la relajación Lagrangeana y los algoritmos heurísticos se ve claramente con los siguientes diagramas.

Considérese una instancia de un problema de minimización *NP−completo*. Si se dibuja una línea vertical donde se representen los valores de la función objetivo, entonces en algún lugar de esta línea está el valor óptimo de la instancia del problema considerado.

El óptimo divide la línea de valores en dos partes:

- arriba del valor óptimo ( desconocido ) se encuentran las cotas superiores;
- abajo del valor óptimo ( desconocido ) se encuentran las cotas inferiores

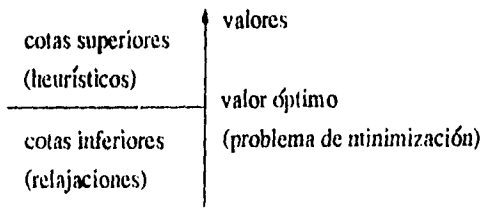



Fig. 2.1

Con el fin de encontrar el valor óptimo, cualquier algoritmo debería calcular cotas superiores e inferiores para este valor. La calidad de estas cotas es importante para el éxito computacional de cualquier algoritmo:

- se desea obtener cotas superiores lo más cercanas posible al valor óptimo, es decir, lo más pequeñas posible;
- se desea obtener cotas inferiores lo más cercanas posible al valor óptimo, es decir, lo más grandes posible.

La relación entre los métodos heurísticos y las relajaciones se puede ver en el diagrama anterior. La cota inferior forma el extremo de un intervalo y la cota superior constituye el otro extremo del intervalo. Si la longitud del intervalo es relativamente pequeña, se puede afirmar que la solución generada con el algoritmo heurístico es una buena solución.

En el contexto de un problema de minimización, la cota superior se genera mediante algún método heurístico. Por otro lado, la técnica más conocida en el caso lineal para generar cotas inferiores es la relajación de *programación lineal* ($PL$). En esta relajación se considera una formulación de programación entera del problema y se relajan las restricciones de integralidad. El resultado es un problema de programación lineal. Al resolver este problema se obtiene una cota inferior para la solución óptima del problema original. Otro método para encontrar cotas inferiores es la *relajación Lagrangeana*, la cual es muy usada. Cuando este tipo de relajaciones se combinan con algoritmos de ramificación y acotamiento (los cuales se verán en la siguiente sección) es posible encontrar la solución óptima del problema.

## 2.2 Algoritmos Exactos.

### 2.2.1 Algoritmos de Ramificación y Acotamiento.

Todo problema acotado de programación entera tiene un número finito de soluciones factibles. Entonces, una manera de resolver este tipo de problemas es evaluar cada una de las soluciones y seleccionar la óptima al comparar cada solución con las demás. A este método se le conoce como *enumeración exhaustiva*. Sin embargo, el número de soluciones factibles generalmente es muy grande. Por esta razón es necesario un procedimiento de enumeración que examine un pequeño número de soluciones factibles. A este procedimiento se le conoce como *enumeración parcial*. Los algoritmos de ramificación y acotamiento son procedimientos de enumeración parcial, ya que eliminan algunas soluciones no prometedoras antes de ser evaluadas.

La idea básica en la que se apoya este método es *divide y vencerás*. Es demasiado complicado resolver directamente el problema original, por esta razón se divide en subproblemas cada vez más pequeños hasta que se puedan resolver. La división ( *ramificación* ) se hace mediante una partición del conjunto completo de soluciones factibles en subconjuntos más pequeños. Se busca el óptimo al *acotar* la mejor solución en cada subconjunto y después se descartan aquellos subconjuntos cuya cota indique que no que contiene la solución óptima para el problema original.

Los pasos de ramificación y acotamiento permiten una gran flexibilidad al diseñar un algoritmo específico para un determinado problema y tienen un efecto importante sobre la eficiencia computacional del algoritmo. Es decir, existen varias estrategias de ramificación y acotamiento dependiendo del problema en consideración. Algunas de estas estrategias se discuten más adelante.

En cada iteración de un algoritmo de ramificación y acotamiento se realizan las siguientes operaciones:

1. Acotamiento
2. Ramificación
3. Búsqueda del óptimo

- **Acotamiento**

Se desea resolver el siguiente problema de programación entera mixta ($PEM$):

$\min Z = cx$
s.a

$$\begin{aligned} Ax &\leq b \\ x_j &\geq 0 \quad \forall j = 1,2,\cdots,n \\ x_j &\in Z \quad \forall j = 1,2,\cdots,l \quad (l \leq n) \end{aligned}$$

El valor mínimo se obtiene al resolver el problema original $PEM$. Pero éste puede ser muy difícil de resolver. Por esta razón se acota inferiormente el valor óptimo. Las cotas se usan para seleccionar subconjuntos prometedores en la búsqueda del óptimo. También se usan para descartar algunos subconjuntos que no pueden contener la solución óptima del problema.

Una buena estrategia de acotamiento es aquélla que:

1. sea fácil de implementar y
2. proporcione una buena cota inferior, es decir, que la cota esté muy cerca del valor óptimo.

Generalmente estos son objetivos contrapuestos y los méritos de cada regla dependen de preferencias.

Una estrategia de acotamiento muy usada consiste en relajar aquellas restricciones que dificulten la solución del problema, es decir, calcular el

valor óptimo sujeto a las restricciones restantes. El problema $PEM$ se puede relajar al eliminar las restricciones de integralidad sobre las variables, dando como resultado el siguiente problema de programación lineal ($PL$):
$\min Z = cx$
s.a

$$\begin{aligned} Ax &\leq b \\ x_j &\geq 0 \quad \forall j = 1, 2, \cdots, n \end{aligned}$$

El valor óptimo del problema relajado $PL$ constituye una cota inferior para el valor óptimo del problema original, puesto que la región factible de este problema está contenida en la región factible del problema relajado. Es decir, siempre se tiene la siguiente desigualdad:

valor óptimo del problema $PL \leq$ valor óptimo del problema $PEM$

Sea $x^*$ la solución óptima del problema relajado $PL$. La igualdad se da cuando $x_j^* \in Z \quad \forall j = 1, 2, ..., I$. En este caso $x^*$ es factible para el problema $PEM$ y por lo tanto este problema está resuelto. Si no se da la igualdad entonces $x_j^* \notin Z$ para alguna $j = 1, 2, ..., I$.

- **Ramificación**

Se aplica la estrategia de ramificación cuando las variables con restricciones enteras no toman un valor entero en la solución óptima del problema relajado $PL$. Es decir, cuando $x_j^* \notin Z$ para alguna $j = 1, 2, ..., I$.

La operación de ramificación consiste en particionar el conjunto de soluciones factibles del problema $PEM$ con el propósito de localizar fácilmente el óptimo. De esta manera se generan dos o más subproblemas.

Sea $S = \{Ax \leq b \mid x_j \geq 0 \quad \forall j = 1, 2, ..., n \quad x_j \in Z \quad \forall j = 1, 2, ..., I \quad (I \leq n)\}$ el conjunto de soluciones factibles del problema $PEM$. Sea una partición de $S$ en subconjuntos $S_1, S_2, ..., S_k$. Cada subconjunto $S_i$ es el conjunto de soluciones factibles que se obtiene agregando restricciones adicionales al problema $PEM$.

Las únicas variables que se toman en cuenta para ramificarlas son las variables con restricciones enteras que tienen un valor no entero en la solución óptima del problema $PL$. La variable que se elige para ramificar se conoce como *variable de ramificación*.

Cuando se usan algoritmos de ramificación y acotamiento el principal objetivo es generar el menor número posible de subproblemas. Entonces, una buena regla de ramificación es aquélla que:

1. genera pocos subproblemas y

2. genera subproblemas con muchas restricciones, es decir, excluye muchas soluciones de cada subproblema.

En este caso los criterios también entran en conflicto y uno tiene que ponderar las ventajas de cada uno.

Se describirá la estrategia de ramificación para problemas binarios. Considere el siguiente problema binario, $PEB$:

$\min Z = cx$
s.a

$$\begin{aligned} Ax &\leq b \\ x_j &= 0,1 \quad \forall j = 1,2,\cdots,n \end{aligned}$$

A este problema se le conoce como subproblema 1. El problema relajado, $PL$, es el siguiente:

$\min Z = cx$
s.a

$$\begin{aligned} Ax &\leq b \\ x_j &\geq 0 \quad \forall j = 1,2,\cdots,n \end{aligned}$$

Supóngase que la solución óptima de este problema no es factible para el problema original $PEB$ , es decir, $x_j^* \notin \{0,1\}$ para alguna $j = 1,2,...,n$. Entonces, se ramifica el subproblema 1 al agregar las siguientes restricciones $x_j = 0$ y $x_j = 1$. De esta manera se generan dos subproblemas:

$$\begin{aligned} \text{subproblema 2} &= \text{subproblema 1} + (x_j = 0) \\ \text{subproblema 3} &= \text{subproblema 1} + (x_j = 1) \end{aligned}$$

Por otro lado, si se tiene un problema $PEM$, la variable de ramificación puede tomar un gran número de valores enteros posibles. Por lo tanto es ineficiente crear y analizar todos los subproblemas que se generan al fijar la variable de ramificación en cada uno de estos valores. Por esta razón se especifican dos intervalos de valores para la variable de ramificación.

Por ejemplo, sea $x_j$ para alguna $j = 1,2,...,I$ la variable de ramificación; y sea $x_j^*$ su valor ( no entero ) en la solución óptima del problema relajado.

Entonces, en cualquier punto de la región factible del problema $PEM$ se tiene que:

$$x_j \leq [x_j^*] \quad \text{o} \quad x_j \geq [x_j^*] + 1$$

ya que en el intervalo $([x_j^*], [x_j^*] + 1)$ no hay ningún punto entero. De esta forma se ramifica sobre la variable $x_j$ y se generan dos subproblemas:

$$\begin{aligned} \text{subproblema } 2 &= \text{subproblema } 1 + (x_j \leq [x_j^*]) \\ \text{subproblema } 3 &= \text{subproblema } 1 + (x_j \geq [x_j^*] + 1) \end{aligned}$$

Un subproblema es *insondable* si no es necesario ramificarlo. Esto puede suceder por las siguientes razones:
1. El subproblema no es factible. En este caso se elimina de consideración puesto que no contiene la solución óptima del problema original.
2. Al acotar un subproblema se obtiene una solución factible para el problema original, la cual es la solución óptima de este subproblema. En este caso se tiene una solución candidata que se compara con otras que se obtengan posteriormente para determinar cuál es la solución óptima.

Supóngase que el valor óptimo de esta solución es $Z_1$. Si $Z^*$ es el valor óptimo del problema original se tiene la siguiente propiedad:

$$Z^* \leq Z_1$$

En cada iteración se debe actualizar esta cota. La solución del subproblema que contenga la menor cota superior es la solución candidata hasta el momento. Al actualizar la cota superior se eliminan todos aquellos subproblemas cuya cota inferior sea mayor que la actual cota superior del problema $PEM$.
3. La cota inferior del subproblema es mayor que la actual cota superior para el valor óptimo del problema original $PEM$. En este caso el nodo no es prometedor y se elimina de consideración.

- **Búsqueda de la Solución Optima**

Los problemas generados se pueden representar mediante un árbol, donde los nodos del árbol son los subproblemas. Por ejemplo:

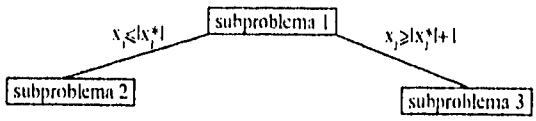



Fig. 2.2

A los nodos que no contienen sucesores se les llama *nodos terminales*, los cuales tienen asociados subproblemas insondables. En el árbol de la figura 2.2, hay dos nodos terminales: los subproblemas 2 y 3. Las restricciones asociadas con cada nodo del árbol son las restricciones del problema $PEM$ más las restricciones asociadas con los arcos que van del subproblema 1 a ese nodo.

La operación de búsqueda indica la secuencia en la que los nodos generados se examinan. Esto es importante porque los árboles generados con las diferentes alternativas pueden variar mucho en su tamaño y por lo tanto en el tiempo requerido para encontrar el óptimo.

Existen varios criterios de búsqueda, como la estrategia de prioridad y la regla de la cota más reciente. En la estrategia de prioridad se acotan todos los subproblemas que se generan al ramificar un nodo. Se examina el nodo ( es decir, se ramifica el nodo ) con la menor cota inferior de todos los subproblemas de la lista en una etapa determinada. Donde la lista es una colección de nodos no terminales. Supóngase que se tienen los subproblemas dos y tres en la lista y sus cotas inferiores son $L_2$ y $L_3$ respectivamente, donde $L_2 < L_3$. Dado que $L_2 < L_3$ es probable que el valor óptimo del subproblema dos sea menor que el de $L_3$. Es decir, el subproblema con la menor cota es el más prometedor en cuanto a contener la solución óptima del problema $PEM$. Si la solución obtenida es factible para el problema original, ésta es la solución óptima del problema $PEM$ y el algoritmo termina aquí. De lo contrario se ramifica el subproblema dos, el cual tiene asociada la menor cota inferior.

Con la regla de la cota más reciente se acota un subproblema solamente si se eligió de la lista. Los subproblemas no insondables de la lista se meten a una pila conforme se van generando con la estrategia de ramificación. Cuando se tenga que seleccionar un subproblema para examinarlo, (es decir, para acotarlo), se elige el último subproblema incluido en la pila. Por esta razón, a este criterio se le conoce como regla de la cota más reciente. Al acotar un nodo se resuelve el subproblema correspondiente con el método

simplex. La ventaja de esta regla es que en lugar de iniciar el método simplex desde el principio, cada vez que se acota un nodo, se puede resolver el subproblema correspondiente por reoptimización utilizando análisis de sensibilidad.

En ambos criterios de búsqueda, el algoritmo termina cuando se tiene que elegir un subproblema de la lista ( o de la pila ) y ésta se encuentra vacía. Si en este momento hay una solución candidata, el problema original *PEM* es factible, de lo contrario se dice que no es factible.

### Algoritmo de Balas

Se describirá un algoritmo de ramificación y acotamiento : el algoritmo de Balas, el cual es un procedimiento de *enumeración implícita.* A los algoritmos de ramificación y acotamiento diseñados específicamente para resolver problemas binarios se les conoce como métodos de enumeración implícita. La idea de estos métodos es considerar solamente algunos puntos de todo el conjunto de soluciones factibles. El resto de las soluciones se eliminan porque no son prometedoras, es decir, se enumeran implícitamente.

Por ejemplo, se desea determinar todas las soluciones factibles de la siguiente desigualdad:

$$3x_1 - 8x_2 + 5x_3 \leq -6 \quad xj = (0,1) \quad \forall j = 1,2,3$$

Por simple inspección se tiene que en cualquier solución factible $x_2$ debe tomar el valor de 1. Esto significa que cualquier combinación binaria que contenga $x_2 = 0$ constituye una solución no factible. Entonces, las cuatro combinaciones: $(0,0,0),(0,0,1),(1,0,0,)$ y $(1,0,1)$ se eliminan automáticamente y se dice que se enumeraron implícitamente.

Dado $x_2 = 1$, se tiene que:

$$3x_1 - 8(1) + 5x_3 \leq -6 \quad \Rightarrow \quad 3x_1 + 5x_3 \leq 2$$

Las variables $x_1$ y $x_3$ pueden tomar el valor de uno si su contribución en el lado izquierdo de la desigualdad no excede a dos. El coeficiente de $x_1$ es tres, por lo que $x_1$ debe tomar el valor cero. Esto significa que $(1,1,0)$ y $(1,1,1)$ se eliminan por no ser prometedoras.

Como $x_1 = 0$ y $x_2 = 1$ , la combinación $(0,1,1)$ se elimina puesto que el coeficciente de $x_3$ es cinco. La combinación que falta, $(0,1,0)$, es la única solución factible de la desigualdad.

Se pueden representar gráficamente todas las combinaciones posibles mediante el siguiente árbol:

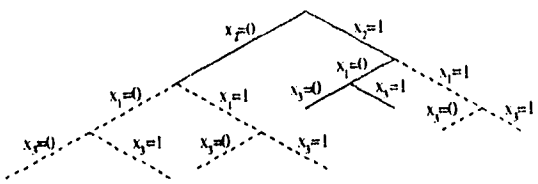



Fig. 2.3

Las líneas punteadas muestran las combinaciones que se enumeraron implícitamente. Obsérvese que en el método de enumeración implícita se empieza con una o más variables que se fijan en un valor binario y gradualmente se construye la solución al aumentar el número de variables con valores fijos.

Se deben considerar los siguientes puntos importantes para implementar el método de enumeración implícita adecuadamente:

1. El método de enumeración que se use debe enumerar todos los puntos en una forma no redundante.

2. Este método debe eliminar el mayor numero posible de soluciones no prometedoras.

Para usar el algoritmo de Balas el problema debe transformarse a la siguiente forma :

$$\min Z = \sum_{j \in N} c_j x_j \qquad c_j \geq 0, \quad j \in N = \{1, 2, ..., n\}$$

s.a

$$\sum_{j \in N} a_{ij} x_j + S_i = b_i \quad i \in M = \{1, 2, \cdots, m\}$$

$$x_j = (0, 1) \quad j \in N$$

$$S_i \geq 0 \quad i \in M$$

si $c_j$ es negativo, entonces se expresa $x_j$ de la siguiente forma:

$$x_j = 1 - y_j \text{ donde } y_j = 0 \text{ o } 1 \quad \forall j \in N$$

- Características del Arbol

Supóngase que un problema de programación entera binario tiene seis variables, $x_j$, y que en alguna etapa del algoritmo se tiene el siguiente árbol:

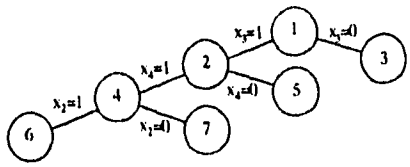



Fig. 2.4

Este árbol tiene las siguientes características:
1. Cada rama del árbol especifica, para alguna variable, $x_j$, que $x_j = 0$ ó 1. Por ejemplo, la siguiente rama establece que $x_3 = 1$:

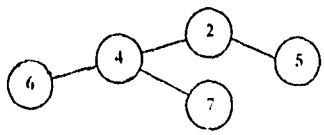



Fig. 2.5

2. En cada nodo se especifican los valores de algunas variables. Por ejemplo, para el nodo 6 se especifican los valores de $x_3$, $x_4$, y $x_2$. A estas variables se les conoce como *variables fijas*. A las demás variables se les llaman *variables libres*. En este caso las variables $x_1, x_5$ y $x_6$ son variables libres. Entonces, se tienen los siguientes conjuntos:

$$U_0 = \textit{conjunto de índices de las variables fijas en cero}$$
$$U_1 = \textit{conjunto de índices de las variables fijas en uno}$$
$$U_L = \textit{conjunto de índices de las variables libres}$$

Obsérvese que $U_L = \{1, 2, ..., n\} - (U_0 \cup U_1)$

Por ejemplo, para el nodo 7 se tiene que: $U_0 = \{2\}$, $U_1 = \{3, 4\}$ y $U_L = \{1, 5, 6\}$

3. Una *terminación de un nodo* es la especificación de los valores de todas las variables libres de ese nodo. Por ejemplo, para el nodo 6, una terminación

es: $x_1 = 1$, $x_5 = 1$ y $x_6 = 0$.

Considérese un nodo determinado del árbol. Dados los valores de las variables fijas en ese nodo, se quiere encontrar su mejor terminación. Para ello se completa el nodo especificando los valores de las variables libres de tal manera que la función objetivo sea más pequeña. Como el vector de costos $c$ es no negativo basta fijar las variables libres en cero. Sin embargo, esta terminación no siempre es factible; es decir, no siempre se satisfacen las restricciones del problema original. Las restricciones se pueden escribir de la siguiente manera:

$$\sum_{j \in U_1} a_{ij}x_j + \sum_{j \in U_0} a_{ij}x_j + \sum_{j \in U_L} a_{ij}x_j + S_i = b_i \quad i \in M,$$

donde $S_i$ es la variable de holgura, pero

$$\sum_{j \in U_0} a_{ij}x_j = \sum_{j \in U_L} a_{ij}x_j = 0 \quad \text{y} \quad \sum_{j \in U_1} a_{ij}x_j = \sum_{j \in U_1} a_{ij}$$

$$\Rightarrow \quad \sum_{j \in U_1} a_{ij} + S_i = b_i$$

$$\Rightarrow \quad S_i = b_i - \sum_{j \in U_1} a_{ij}$$

Si $S_i \geq 0$ para toda $i$ la mejor terminación del nodo es factible. En este caso se tiene una solución candidata. El valor objetivo de esta solución es una cota superior, $Z_{UB}$, para el valor óptimo del problema original. Por otro lado, si la mejor terminación de un nodo no es factible, entonces alguna variable $x_j$ tiene que valer uno. El nodo se ramifica y se generan dos subproblemas, uno con $x_j = 1$ y otro con $x_j = 0$. La variable de ramificación $x_j$ se elige de acuerdo con las pruebas que se verán más adelante. Para ello se introducirá la siguiente notación:

Sea

$S_i^t$ la variable de holgura de la $i - ésima$ restricción en la iteración $t$.

$Z^t$ el valor objetivo de la mejor terminación de un nodo en la iteración $t$.

$N_t$ el conjunto de las variables libres no descartadas por las pruebas 1 y 2 en la iteración $t$.

- **Prueba 1.**

En esta prueba se eliminan aquellas variables que no puedan mejorar la infactibilidad del problema. Alguna variable libre $x_r$ tiene que valer uno. Si $a_{ir} \geq 0$ para toda $i$ tal que $S_i^t < 0$, entonces $x_r$ no puede mejorar la infactibilidad del problema y debe descartarse.

- **Prueba 2.**

El propósito de esta prueba es eliminar aquellas variables libres que al valer uno forman una solución cuyo valor objetivo es mayor que la actual cota superior del problema original. Para cualquier variable libre $x_r$, si

$$c_r + Z^t \geq Z_{UB}$$

entonces esta variable no puede llevar a un valor objetivo mejor y por lo tanto debe descartarse.

- **Prueba 3.**

Esta prueba se realiza para saber si existen terminaciones del nodo que satisfagan aquellas restricciones no satisfechas por la mejor terminación. Considérese la $i$ – *ésima* restricción:

$$a_{ij}x_1 + \ldots + a_{in}x_n + S_i = b_i$$

para la cual

$$S_i^t = b_i - \sum_{j \in U_1^t} a_{ij} < 0$$

Ninguna de las variables de $N_t$ es promisoria si para al menos una $S_i^t < 0$ la condición siguiente se satisface:

$$\sum_{j \in N_t} \min\{0, a_{ij}\} > S_i^t$$

Esto establece que el conjunto $N_t$ no puede llevar a una solución factible y por lo tanto debe descartarse. En este caso el nodo es insondeable y se elimina de consideración.

- **Prueba 4.**

En esta prueba se eligirá una variable del conjunto $N_t$ como variable de ramificación. Si para alguna restricción $i$, al aumentar en uno la variable $x_j$ la variable de holgura es no negativa, entonces

$$\min\{0, S_i^t - a_{ij}\} = 0$$

Entonces, una medida empírica de la infactibilidad total de la variable $x_j$, al fijarla en uno, es

$$v_j^t = \sum_{i=1}^{m} \min\{0, S_i^t - a_{ij}^t\}$$

Si $N_t \neq \emptyset$ se ramifica sobre la variable $x_k$ tal que:

$$v_k^t = \max_{j \in N_t}\{v_j^t\}$$

En caso de empate, se selecciona la variable que tenga el menor coeficiente $c_k$ en la función objetivo.

**Algoritmo.**

En cada iteración $t$ del algoritmo de Balas se realizan los siguientes pasos:

1. Si faltan nodos por analizar se elige un nodo con alguna regla por ejemplo la de la última cota. Ir al paso dos. Por el contrario, si todos los nodos están analizados el algoritmo termina en esta etapa. Si en este momento se tiene una solución candidata, ésta es la solución óptima del problema original. De lo contrario el problema original no es factible.
2. Se encuentra la mejor terminación del nodo, haciendo $x_j = 0 \quad \forall j \in U_L^t$. El valor objetivo de esta de esta terminación es $Z^t$. Ir al paso 3.
3. Se verifica si la mejor terminación es factible. Si es factible, ésta es la nueva solución candidata y su valor objetivo es la actual cota superior para el problema original, es decir $Z_{UB} = Z^t$. Ir al paso 1. Si la mejor terminación no es factible ir al paso 4.
4. Se ramifica el nodo sobre alguna variable libre, $x_j$, generando de esta manera dos subproblemas, uno con $x_j = 1$ y otro con $x_j = 0$. La variable de ramificación se elige con las pruebas descritas anteriormente. Regresar al paso 1.

**Ejemplo 2.2.1** *Considérese el siguiente problema:*
$\min W = 3x_1 + 2x_2 + 5x_3 + 2x_4 + 3x_5 - 8$

*s.a*

$$\begin{aligned} -x_1 - x_2 + x_3 + 2x_4 - x_5 &\leq 1 \\ -7x_1 \quad + 3x_3 - 4x_4 - 3x_5 &\leq -2 \\ 11x_1 - 6x_2 \quad - 3x_4 - 3x_5 &\leq -1 \\ x_j &\in \{0,1\} \quad \forall j \in \{1,\cdots,5\} \end{aligned}$$

ITERACIÓN 0. *Se inicializa la cota superior,* $Z_{UB} = \infty$.
*1. La mejor terminación es* $x_j = 0 \quad \forall j \in \{1,...,5\}$ *, donde* $Z^0 = 0$
*2. Esta terminación no es factible puesto que* $(S_1^0, S_2^0, S_3^0) = (1,-2,-1)$
*3. Se ramifica sobre alguna variable:*
*Prueba1. Se tiene que* $S_i^0 < 0$ *para* $i = 2,3$. *Como* $a_{i3} \geq 0$ *para* $i = 2,3$, *entonces* $x_3$ *empeora la infactibilidad. Por lo tanto se elimina como posible variable de ramificación.*
*Prueba 2. Se tiene que* $c_j + Z^0 < Z_{UB} = \infty \quad \forall j \in \{1,...,5\}$, *por esta razón no se elimina ninguna variable.*
*Prueba 3. Sea* $N_t = \{x_1, x_2, x_4, x_5\}$. *Donde*

$$\sum_{j \in N_t} \min\{0, a_{2j}\} = -7 - 4 - 3 = -14 < -2$$

$$\sum_{j \in N_t} \min\{0, a_{3j}\} = 0 - 6 - 3 - 3 = -12 < -1$$

*Por lo tanto sí existen terminaciones factibles.*
*Prueba 4. Del conjunto* $N_t$, *¿sobre qué variable conviene ramificar ?*

$$\begin{aligned} v_1^0 &= 0 + 0 - 12 = -12 \quad v_4^0 = -1 + 0 + 0 = -1 \\ v_2^0 &= 0 - 2 + 0 = -2 \qquad v_5^0 = 0 + 0 + 0 = 0 \end{aligned}$$

*donde*

$$\max_{j \in N_t}\{v_j^0\} = v_5^0$$

*por lo tanto se ramifica sobre la variable* $x_5$.*(Ver figura 2.6)*
ITERACIÓN 1. *Se elige arbitrariamente un problema, por ejemplo el 3.*
*1. La mejor terminación es :* $x_j = 0 \quad \forall j \in \overline{1,4}$. *Se tienen los siguientes conjuntos:* $U_1 = \{5\}$, $U_0 = \emptyset$ *y* $U_t = \{1,2,3,4\}$. *La cota inferior de este subproblema es* $Z^1 = 3$.
*2. Las variables de holgura tienen los siguientes valores:* $(S_1^1, S_2^1, S_3^1) =$

(2, 1, 2). *Como* $S_i^1 \geq 0 \quad \forall i \in \{1,2,3\}$, *esta terminación es factible, y por lo tanto constituye una solución candidata. La cota superior para el problema original es* $Z_{UB} = 3$.

ITERACIÓN 2. *El siguiente nodo que se debe analizar es el 2.*

*1. La mejor terminación es :* $x_j = 0 \quad \forall j \in \overline{1,5}$, *donde* $U_1 = \emptyset$, $U_0 = \{5\}$ *y* $U_L = \{1,2,3,4\}$. *La cota inferior de este subproblema es* $Z^2 = 0$.

*2. Esta terminación es no es factible puesto que* $(S_1^2, S_2^2, S_3^2) = (1, -2, -1)$

*3. Se debe ramificar este subproblema.*

*Prueba 1. Con esta prueba se excluye a la variable* $x_3$.

*Prueba 2. Se excluyen las variables* $x_1$ *y* $x_3$, *puesto que* $c_1 + Z^2 = 3 = Z_{UB} \quad y \quad c_3 + Z^2 = 5 > Z_{UB}$.

*Prueba 3. Se tiene que* $N_t = \{x_2, x_4\}$. *Por otro lado,* $S_i^2 < 0$ *para* $i = 2, 3$. *Entonces,*

$$\sum_{j \in N_t} \min\{0, a_{2j}\} = -7 - 4 - 3 = -14 < -2$$

$$\sum_{j \in N_t} \min\{0, a_{3j}\} = 0 - 6 - 3 - 3 = -12 < -1$$

*Por lo tanto el nodo 2 sí existen terminaciones factibles.*

*Prueba 4. Del conjunto* $N_t$, *¿sobre qué variable conviene ramificar ?*

$$v_2^2 = 0 - 2 + 0 = -2 \quad v_4^2 = -1 + 0 + 0 = -1$$

*donde*

$$\max_{j \in N_t}\{v_j^2\} = v_4^2$$

*por lo tanto se ramifica sobre la variable* $x_4$.

ITERACIÓN 3. *Arbitrariamente se elige un nodo, por ejemplo el 5.*

*1. La mejor terminación es* $x_j = 0 \quad \forall j \in \{1,2,3\}$, *donde* $U_0 = \{5\}$, $U_1 = \{4\}$ *y* $U_L = \{1,2,3\}$. *La cota inferior para este subproblema es* $Z^3 = 2$.

*2. La mejor terminación no es factible puesto que una variable de holgura es negativa:* $(S_1^3, S_2^3, S_3^3) = (-1, 2, 2)$.

*3. Se debe ramificar sobre alguna vaiable libre.*

*Prueba 1. La variable* $x_3$ *se excluye porque no mejora la infactibilidad del problema.*

*Prueba 2. Con esta prueba se excluyen las variables* $x_1$, $x_2$ *y* $x_3$, *puesto que*

$$\begin{aligned} &si\ x_1 = 1 \ \Rightarrow\ c_1 + Z^3 = 3 + 2 = 5 > Z_{UB} \\ &si\ x_2 = 1 \ \Rightarrow\ c_2 + Z^3 = 2 + 2 = 4 > Z_{UB} \\ &si\ x_3 = 1 \ \Rightarrow\ c_3 + Z^3 = 5 + 2 = 7 > Z_{UB} \end{aligned}$$

*Prueba 3. El conjunto $N_t$ es vacío, por lo tanto el nodo es no prometedor.*
ITERACIÓN 4. *El único nodo que falta por analizar es el nodo 4.*
*1. La mejor terminación es $x_j = 0 \quad \forall j \in \overline{1,3}$, donde $U_0 = \{4,5\}$, $U_1 = \emptyset$ y $U_L = \{1,2,3\}$. La cota inferior para este subproblema es $Z^4 = 0$.*
*2. La mejor terminación no es factible puesto que se tienen variables de holgura negativas: $(S_1^4, S_2^4, S_3^4) = (1, -2, -1)$.*
*3. Se debe ramificar sobre alguna variable libre.*
***Prueba 1.** Con esta prueba se excluye la variable $x_3$.*
***Prueba 2.** Se elimina $x_1$ y $x_3$, ya que*

$$\begin{aligned} &si\ x_1 = 1 \ \Rightarrow\ c + Z^4 = 3 + 0 = 3 = Z_{UB} \\ &si\ x_3 = 1 \ \Rightarrow\ c_3 + Z^4 = 5 + 0 = 5 > Z_{UB} \end{aligned}$$

***Prueba 3.** $N_t = \{x_2\}$. Se tiene que $S_i^4 < 0$ para $i = 2,3$. Como*

$$\min\{0, a_{22}\} = \min\{0,0\} = 0 > -2$$

*este nodo no tiene terminaciones factibles, y por lo tanto se elimina.*
*Todos los nodos han sido examinados. La solución $\overline{x^*} = (0,0,0,0,1)$ es la óptima y el valor óptimo es $Z^* = 3$.*

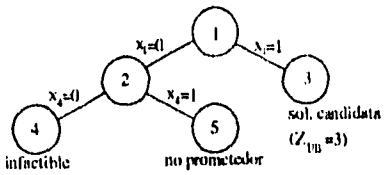


Fig. 2.6

**Ejemplo 2.2.2** *El problema del agente viajero se puede resolver mediante el algoritmo de Balas debido a que es un problema binario. Considérese un problema del agente viajero con la siguiente matriz de costos:*

| $i\backslash j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| – | – | – | – | – | – |
| 1 | M | 1 | 2 | 2 | 3 |
| 2 | 1 | M | 2 | 3 | M |
| 3 | 2 | 2 | M | 2 | 2 |
| 4 | 2 | 3 | 2 | M | M |
| 5 | 3 | M | 2 | M | M |

*donde* $M >> 0$.

*Este problema se puede plantear de la siguiente manera:*

$\min x_{12} + 2x_{13} + 2x_{14} + 3x_{15} + 2x_{23} + 3x_{24} + Mx_{25} + 2x_{34} + 2x_{35} + Mx_{45}$

*s.a*

$$x_{12} + x_{13} + x_{14} + x_{15} + h_1 = 2 \quad (2.1)$$

$$(x_{23} + x_{24} + x_{25}) + (x_{12}) + h_2 = 2 \quad (2.2)$$

$$(x_{34} + x_{35}) + (x_{13} + x_{23}) + h_3 = 2 \quad (2.3)$$

$$(x_{45}) + (x_{14} + x_{24} + x_{34}) + h_4 = 2 \quad (2.4)$$

$$x_{15} + x_{25} + x_{35} + x_{45} + h_5 = 2 \quad (2.5)$$

$$\sum_{i \in S} \sum_{j \in S, i<j} x_{ij} + h_i = |S| - 1, S \subset V = \overline{1,5}, S \neq \emptyset \quad (2.6)$$

$$x_{ij} \in \{0,1\} \quad i,j \in V \quad i < j \quad (2.7)$$

$$h_i = 0 \quad \forall i \in \{1, \cdots, 5\} \quad (2.8)$$

$$h_i \geq 0 \quad i \in \{6, \cdots, 29\} \quad (2.9)$$

*Obsérvese que el conjunto de restricciones (2.6) está formado por 24 igualdades.*

*A continuación se presenta detalladamente la primera iteración del algoritmo.*

*Se inicializa la cota superior en* $Z_{UB} = M$, *donde* $M >> 0$. *En esta etapa* $U_0 = U_1 = \emptyset$.

*1. La mejor terminación para este problema es* $x_{ij} = 0 \quad \forall i,j \in V \quad i < j$, *cuyo valor objetivo es* $Z = 0$.

*2. Claramente esta terminación es no es factible.*

*3. Se realizan las siguientes pruebas para determinar sobre qué variable conviene ramificar.*

*PRUEBA 1. En esta prueba se eliminan aquellas variables libres que no puedan mejorar la infactibilidad del problema.*

*Se tiene una solución que no es factible cuando* $h_i$ *es diferente de cero para alguna* $i \in \{1,...,5\}$, *o cuando* $h_i$ *toma un valor negativo para alguna* $i \in \{6,...,29\}$. *Entonces, para cualquier variable libre* $x_{ir}$ *con* $i \in \{1,...,5\}$, *si* $a_{ir} \leq 0$ *para toda* $i$ *que corresponde a* $h_i > 0$, $x_{ir}$ *no puede mejorar la infactibilidad del problema. Por otro lado, para cualquier variable libre* $x_{ir}$ *con* $i \in \{1,...,29\}$ *si* $a_{ir} \geq 0$ *para toda* $i$ *que corresponde a* $h_i < 0$, *entonces* $x_{ir}$ *tampoco puede mejorar la infactibilidad del problema.*

*En este caso* $h_i = 2$ *para* $i = \overline{1,5}$. *Como* $a_{ij} = 1$ *para toda* $i = \overline{1,5}$, *entonces no se elimina ninguna variable libre.*

*PRUEBA 2. Esta prueba elimina aquellas variables $x_{rs}$ tales que $c_{rs} + Z > Z_{UB}$.*
*Se tiene que $Z_{UB} = \mathrm{M}$ , por otro lado $c_{25} = \mathrm{M}$ y $c_{45} = \mathrm{M}$. Por lo tanto, las variables $x_{25}$ y $x_{45}$ se eliminan ya que si valen uno no producen una solución mejor que la actual.*
*PRUEBA 3. El propósito de esta prueba es saber si el problema tiene terminaciones factibles.*
*Sea $N = U_L - \{(2,5),(4,5)\}$. Claramente, se pueden dar valores adecuados a las variables del conjunto $N$ para que se satisfagan las restricciones del problema. Por lo tanto, el problema sí tiene terminaciones factibles.*
*PRUEBA 4. Si la variable de holgura $h'_k = h_k - a_{ij}$ donde $k \in \{1,\ldots,5\}$ es cero al fijar $x_{ij}$ en uno, entonces*

$$\max\{0, | h_k - a_{ij} |\} = 0$$

*Por otro lado, si al fijar en uno alguna variable libre $x_{ij}$, la variable de holgura toma un valor no negativo, entonces*

$$\min\{0, h_k - a_{ij}\} = 0 \quad con \quad k \in \{6,\ldots,29\}$$

*Entonces, una medida de la infactibilidad total del problema al fijar la variable $x_{ij}$ en uno es*

$$v_{ij} = \sum_{k=6}^{29} \min\{o, h_k - a_{ij}\} - \sum_{k=1}^{5} \max\{0, | h_k - a_{ij}\}$$

*Por lo tanto, si $N \neq \emptyset$ la variable de ramificación $x_{rs}$ se elige como aquélla en donde*

$$v_{rs} = \max_{(i,j)\in N} \{v_{ij}\}$$

*Para este problema se tiene que $v_{ij} = -8 \quad \forall i,j \in V \quad i < j$. Se elige la variable $x_{12}$ como variable de ramificación, puesto que $c_{12} = \min_{i<j}\{c_{ij}\}$. Este procedimiento se sigue para cada nodo no terminal. En la figura 2.7 se muestra el árbol de ramificación y acotamiento generado mediante el criterio de la última cota. En este árbol se puede observar que la solución óptima del problema es $1 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ , y el valor óptimo es $Z^* = 11$.*

*Obsérvese que a pesar de que se resolvió un problema simétrico con pocas ciudades, se generaron muchos nodos en el árbol. Por otro lado, el número de nodos generados con este procedimiento es a lo más $2^n$ , donde $n$ es el*

*número de variables de decisión. Por esta razón, entre más variables se tengan el árbol puede crecer considerablemente. Entonces, el algoritmo de Balas no es un buen método para resolver el problema del agente viajero. En el siguiente capítulo se verán algoritmos más eficientes en donde se aprovecha la estructura del problema.*

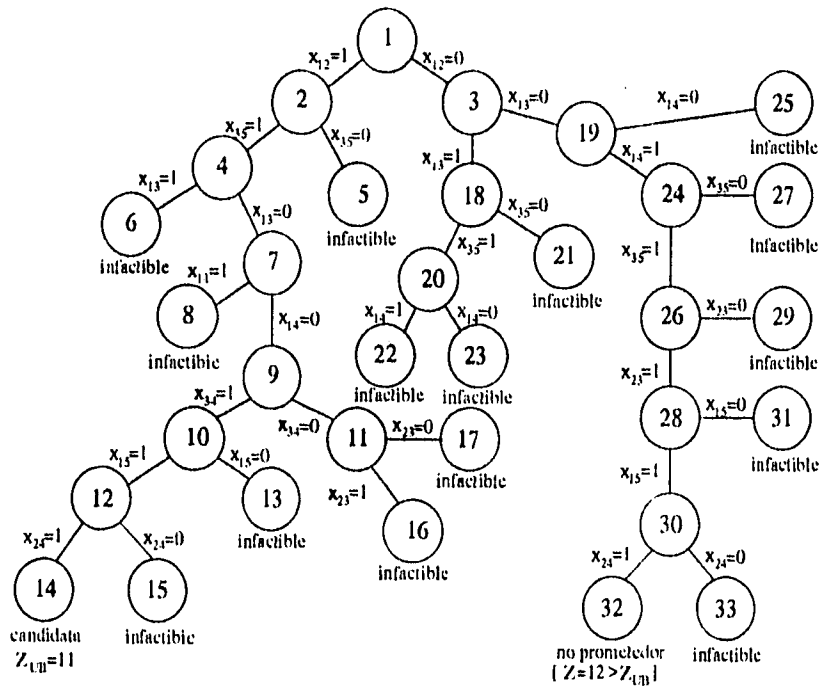



Fig. 2.7

### 2.2.2 Relajación Lagrangeana.

#### Introducción.

Otra técnica para encontrar cotas inferiores es la *relajación Lagrangeana*. Esta técnica se desarrolló a principios de los 70's con el trabajo de Held y Karp en el problema del agente viajero. Actualmente es una técnica indispensable para generar cotas inferiores usadas en algoritmos que resuelven problemas de optimización combinatoria. En la relajación Lagrangeana se realizan los siguientes pasos:

1. Se considera una formulación de programación entera del problema.

2. Se asocian multiplicadores de Lagrange a algunas restricciones las cuales se incluyen en la función objetivo y se eliminan de las restricciones.

3. Se encuentra la solución exacta del problema de programación entera obtenido en el punto 2.

La solución que se obtiene en el punto 3 constituye una cota inferior para la solución óptima del problema original.

A primera vista podría parecer que la relajación Lagrangeana no es un buen método, ya que en el paso 1 se tiene una formulación de programación entera del problema y se propone generar una cota inferior para éste al resolver otro problema de programación entera ( paso 3 ). Sin embargo, hay dos razones principales para usar la relajación Lagrangeana:

1. Muchos problemas de optimización combinatoria están constituidos por un problema fácil ( en el sentido $NP-completo$, es decir se pueden resolver mediante un algoritmo polinomialmente acotado ) que se complica al agregarle otras restricciones. Si se absorben estas restricciones en la función objetivo ( paso 2 ) entonces se tendrá un problema fácil de resolver. De esta manera sólo se deberán elegir los valores numéricos apropiados para los multiplicadores de Lagrange, ya que éstos determinan la calidad de la cota inferior generada ( se desean obtener cotas inferiores cercanas al óptimo ). Un método para encontrar estos valores es la *optimización subgradiente.*

2. La experiencia práctica con la relajación Lagrangeana indica que se obtienen cotas muy buenas a un costo computacional razonable.

**Definición de la relajación Lagrangeana.**

Considere el siguiente problema binario ( escrito en forma matricial ) que se denotará como problema $(P)$ :

$$\begin{aligned} \min \quad & cx \\ s.a \quad & \\ & Ax \geq b \\ & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{aligned}$$

Se define la relajación Lagrangeana del problema ( $P$ ) con respecto al conjunto de restricciones $Ax \geq b$ como :

$$\begin{array}{ll} \min & cx + \lambda(b - Ax) \\ s.a & \\ & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{array}$$

donde $\lambda$ es un vector no negativo de multiplicadores de Lagrange. Este problema se denotará como $PCIL$ ( problema de la cota inferior Lagrangeana ).

Lo que se ha hecho es:

- elegir un conjunto de restricciones en el problema para relajarlo; y
- asociar multiplicadores de Lagrange a estas restricciones para incluirlas en la función objetivo.

El problema generado con la relajación Lagrangeana, para cualquier $\lambda \geq 0$, proporciona cotas inferiores para la solución óptima del problema original ($P$). Esto se puede ver como sigue. El valor óptimo del problema $P$ :

$$\begin{array}{ll} \min & cx \\ s.a & \\ & Ax \geq b \\ & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{array}$$

es mayor o igual que el valor óptimo del siguiente problema:

$$\begin{array}{ll} \min & cx + \lambda(b - Ax) \\ s.a & \\ & Ax \geq b \\ & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{array}$$

ya que $\lambda \geq 0$ y $(b - Ax) \leq 0$ , es decir se agrega un término que es $\leq 0$ a la función objetivo. Obsérvese que si se relajan restricciones que corresponden a igualdades ($Ax = b$), no es necesario restringir en signo a $\lambda$. El valor óptimo de este problema es a su vez mayor o igual que el valor óptimo del problema $PCIL$ :

$$\begin{aligned} \min \quad & cx + \lambda(b - Ax) \\ s.a \quad & \\ & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{aligned}$$

puesto que al eliminar un conjunto de restricciones de un problema de minimización se reduce el valor óptimo de la función objetivo. Entonces el valor óptimo del problema $P$ es mayor o igual que el del problema $PCIL$. Por lo tanto para cualquier $\lambda \geq 0$, el problema $PCIL$ proporciona cotas inferiores para el problema $P$.

Obsérvese que si la solución óptima del problema $PCIL$ es factible para el problema original entonces no necesariamente es también solución óptima de éste. Considérese el problema $PCIL$. Supóngase que los multiplicadores $\lambda$ son tales que la solución $x^*$ del problema $PCIL$ es factible para el problema original. Es decir, $x^*$ satisface las siguientes restricciones:

$$\begin{aligned} Ax^* &\geq b \\ Bx^* &\geq d \\ x_j^* \in \{0,1\} \quad \forall j &= 1,2,\ldots,n \end{aligned}$$

El valor de esta solución factible es $cx^*$, mientras que el valor de la cota inferior obtenida con el problema $PCIL$ es $cx^* + \lambda(b - Ax^*)$. Si estos dos valores coinciden, es decir, si la cota superior $cx^*$ es igual a la cota inferior $cx^* + \lambda(b - Ax^*)$, entonces $x^*$ es la solución óptima del problema original. Ya que como se puede ver en la figura 2.1, el único lugar en el que la cota inferior y la cota superior pueden coincidir en la línea de valores es en el punto óptimo.
Entonces, la solución $x^*$ del problema $PCIL$ es óptima para el problema original si:

1. $x^*$ es factible para el problema original, y
2. $cx^* = cx^* + \lambda(b - Ax^*)$, es decir $\lambda(b - Ax^*) = 0$.

Si se relajan restricciones de igualdad ( $Ax = b$ ) cualquier solución del problema $PCIL$ que sea factible para el problema original satisface las dos condiciones anteriores y por lo tanto es solución óptima del problema $P$.

Se está interesado en encontrar los valores para los multiplicadores de Lagrange que proporcionen la máxima cota inferior, es decir la cota inferior que se acerque lo más posible al valor óptimo del problema original. Esto es equivalente a encontrar aquellos multiplicadores $\lambda \geq 0$ tales que:

$$\max_{\lambda \geq 0} \left\{ \begin{array}{cc} \min & cx + \lambda(b - Ax) \\ s.a & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{array} \right\}$$

A este problema se le conoce como *problema dual Lagrangeano.*

Este problema se puede formular de otra manera, la cual es más conveniente para resolverlo. Considérese el problema *PCIL*:

$$\begin{array}{cc} \min & cx + \lambda \cdot (b - Ax) \\ \text{s. a} & Bx \geq d \\ & x_j \in \{0,1\} \quad \forall j = 1,2,\ldots,n \end{array}$$

Cada solución factible $x^k$ de este problema tiene un valor objetivo al que se denotará como $c_k + \lambda \cdot v_k$ , donde $c_k = cx^k$ y $v_k = (v_{k1},\ldots,v_{km})$ con $v_{ki} = b_i - \sum_{j=1}^{n} a_{ij}x_j^k \quad \forall i = 1,\ldots,m$. Supóngase que este problema tiene $K$ soluciones factibles; se desea encontrar la mejor de éstas:

$$\min\{c_k + \lambda \cdot v_k; k = 1,\ldots,K\}$$

Entonces el problema dual Lagrangeano se puede reformular como:

$$\max_{\lambda \geq 0}\{w(\lambda)\}$$

donde

$$w(\lambda) = \min\{c_k + \lambda \cdot v_k; k = 1,\ldots,K\}$$

Un procedimiento muy eficaz que resuelve este problema es el método de *optimización subgradiente.* Este es un procedimiento iterativo en el cual, a partir de un conjunto inicial de multiplicadores de Lagrange se genera otro conjunto de multiplicadores de una manera sistemática. Se puede ver como un procedimiento que pretende maximizar el valor de la cota inferior obtenida al resolver el problema *PCIL* mediante una elección adecuada de multiplicadores. La primera vez que se trabajó en este procedimiento fue en el trabajo de Held y Karp [4] para resolver el problema del agente viajero. El procedimiento de Held y Karp es una aplicación del método estudiado por Motzkin y Schoenberg [9] para resolver sistemas de desigualdades lineales, el cual se discutirá a continuación.

**Método de Relajación para Resolver Sistemas de Desigualdades Lineales.**

Sea $A$ un conjunto cerrado de puntos en un espacio euclideano n-dimensional $E^n$. Si $p$ y $p_1$ son puntos de $E^n$ tales que

$$| p - a | > | p_1 - a | \quad \forall a \in A \tag{2.10}$$

entonces se dice que $p_1$ está *puntualmente más cerca* que $p$ del conjunto $A$. Si $p$ es tal que no existe ningún punto $p_1$ que esté puntualmente más cerca que $p$ de $A$, entonces $p$ es el *punto más cercano* del conjunto $A$.

Considérese un sistema consistente de $m$ desigualdades lineales:

$$\sum_{j=1}^{n} a_{ij}x_j + b_i \geq 0 \quad i = 1, ..., m \tag{2.11}$$

donde $a_{ij}$ y $b_i$ son coeficientes conocidos y donde $rango(A) = m$ con $A = \{a_{ij}\}$. Se desea encontrar un procedimiento que obtenga una solución $(x_1, ..., x_n$ la cual satisface el sistema (2.11). Si el sistema es homogéneo, es decir $b_i = 0 \quad \forall i$, entonces se debe encontrar una solución diferente de la trivial $(0, ..., 0)$.

Cada desigualdad en (2.11) define un semiespacio cerrado

$$H_i = \sum_{j=1}^{n} a_{ij}x_j + b_i \geq 0$$

entonces, el conjunto de puntos que satisfacen el sistema (2.11) coincide con el poliedro convexo

$$A = \bigcap_{i=1}^{m} H_i$$

Supóngase que $A$ no es vacío.

Sea $p \notin A$. La siguiente construcción encuentra un punto $p_1$ que está más cerca que $p$ de $A$. Claramente $\exists k$ tal que $p \notin H_k$. Sea $p'$ el simétrico de $p$ con respecto a la frontera de $H_k$. Si $p_1$ está sobre el segmento que une los puntos $p$ y $p'$ y además $p_1 \neq p, p'$ entonces

$$| p - a | > | p_1 - a | \quad \forall a \in H_k$$

Dado que $A \subset H_k$, se tiene que

$$| p - a | > | p_1 - a | \quad \forall a \in A$$

por lo tanto $p_1$ está puntualmente más cerca que $p$ de $A$. Véase la figura 2.8.

La construcción numérica de $p_1$ se realiza de la siguiente manera:

$$p_1 = p + \pi(q - p)$$

donde $q$ es la proyección de $p$ en el hiperplano $\sum_{j=1}^{n} a_{kj}x_j + b_k = 0$ y $\pi$ es un escalar tal que $0 < \pi < 2$.

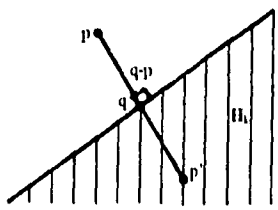



Fig. 2.8

Obsérvese que al pasar de $p$ a $p_1$, $p_1$ se acerca más rápidamente a $A$ si entre todos los semiespacios $H_i$ que no contienen a $p$ se elige el que está más alejado de $p$. Si $\pi = 2$ entonces $p_1 = p'$ y la desigualdad (2.10) se cumple con excepción de aquéllos puntos de $A$ que están sobre la frontera de $H_k$, si éstos existen, en donde se da la igualdad.

La observación anterior sugiere la siguiente búsqueda sistemática de un punto de $A$. Elígase un punto $p$. Si $p \in A$, es decir sus coordenadas satisfacen (2.11), la búsqueda termina en esta etapa. Si $p \notin A$, sea $H_k$ tal que

$$dist(p, H_k) = \max_i \{dist(p, H_i)\}$$

donde *dist* denota distancia euclideana. Sea $q$ tal que

$$q \in H_k, \mid p - q \mid = dist(p, H_k)$$

Sea $\pi$ una constante tal que $0 < \pi \leq 2$ y defínase

$$p_1 = p + \pi(q - p)$$

Por conveniencia se abreviará esta construcción como:

$$p_1 = F_\pi(p)$$

Si $p_1 \in A$ el proceso termina aquí, de lo contrario se realiza otra iteración:

$$p_2 = F_\pi(p_1)$$

Se continúa de esta manera generando una sucesión de puntos $p = p_0, p_1, p_2, \ldots$ los cuales están afuera del conjunto $A$ y se relacionan de la siguiente manera:

$$p_{n+1} = F_\pi(p_n) \quad n = 0, 1, 2, \ldots$$

Existen dos posibilidades:

1. El proceso termina después de $N$ iteraciones con un punto $p_N \in A$.
2. El proceso continúa indefinidamente produciendo una sucesión de puntos $\{p_n\}$. Se puede demostrar que si $0 < \pi < 2$ entonces $p_n$ converge cuando $n \to \infty$ a un punto en la frontera de $A$.

**Optimización subgradiente.**

Se aplicará el método anterior para resolver el problema

$$\max_{\lambda \geq 0} \{w(\lambda)\}$$

donde $$w(\lambda) = \min\{c_k + \lambda \cdot v_k; \quad k = 1, \ldots, K\}$$

Obsérvese que si $w(\lambda) = \min\{c_k + \lambda \cdot v_k; \quad k = 1, \ldots, K\}$ entonces $w(\lambda) \leq c_k + \lambda \cdot v_k; \forall k = 1, \ldots, K$. Por lo tanto el problema se puede reescribir como

$$\max_{\lambda \geq 0} \{w(\lambda)\}$$

donde $$w(\lambda) \leq c_k + \lambda \cdot v_k; \quad \forall k = 1, \ldots, K$$

Más aún, dado un valor fijo $\overline{w} < \max_{\lambda \geq 0}\{w(\lambda)\}$ se desea encontrar un punto $\overline{\lambda}$ tal que $w(\overline{\lambda}) \geq \overline{w}$. Es decir, se desea mejorar la cota inferior $\overline{w}$. Esto es equivalente a resolver el siguiente sistema de desigualdades:

$$\overline{w} \leq c_k + \lambda \cdot v_k \quad \forall k = 1, \ldots, K \text{ y } \lambda \geq 0 \tag{2.12}$$

A continuación se describirá geométricamente como se obtiene una sucesión de puntos $\{\lambda^m\}$ que se van acercando al poliedro $P_{\overline{w}}$ de soluciones factibles del sistema (2.12). De acuerdo con el método anterior, dado un punto $\lambda$, se debe seleccionar aquella desigualdad no satisfecha por $\lambda$ que maximice $\overline{w} - (c_k + \lambda \cdot v_k)$; es decir que minimice $c_k + \lambda \cdot v_k$. Obsérvese que:

1. Si $w_1 < w_2 \Rightarrow P_{w_1} \supset P_{w_2}$

2. Si $w(\lambda) < \overline{w}$, es decir $\lambda$ no satisface (2.12), entonces $P_{\overline{w}}$ está contenido en el semiespacio determinado por el hiperplano que pasa por $\lambda$ y cuya normal es $v_{k(\lambda)}$.

3. El rayo $\{\lambda + t \cdot v_{k(\lambda)} \mid t > 0\}$ es perpendicular a la cara de $P_{\overline{w}}$ cuya ecuación es $\overline{w} = c_{k(\lambda)} + \overline{\lambda} \cdot v_{k(\lambda)}$. El rayo intersecta esta cara en el punto

$$t = \frac{\overline{w} - w(\lambda)}{\| v_{k(\lambda)} \|^2} \tag{2.13}$$

Véase la siguiente figura:

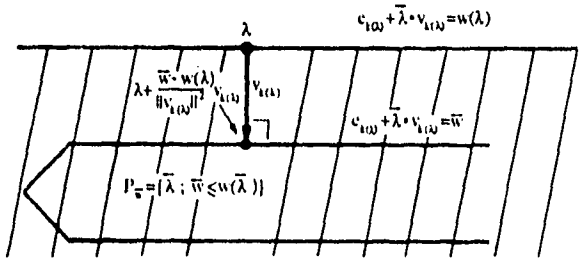



Fig. 2.9

Entonces, sea $\overline{w} < \max w(\lambda)$. La sucesión $\{\lambda^m\}$ se obtiene de la siguiente manera:

$$\lambda^{m+1} = \lambda^m + \pi_m \left[ \frac{\overline{w} - w(\lambda)}{\| v_{k(\lambda)} \|^2} \right] v_{k(\lambda)}$$

donde $0 < \pi_m \leq 2$.

En la práctica, el valor $\overline{w}$ es reemplazado por alguna cota superior de $w^* = \max w(\lambda)$, $Z_{UB}$, ya que ésta es más fácil de calcular. Además se ha observado que los resultados que se obtienen no dependen críticamente del valor exacto de $Z_{UB}$ [5].

**Algoritmo de Optimización Subgradiente:**

1. Sea $\pi$ un parámetro definido por el usuario, tal que $0 < \pi \leq 2$. Se inicializa la cota superior, $Z_{UB}$, para el valor óptimo del problema original con alguna heurística. Se elige un conjunto inicial $(\lambda_i)$ de multiplicadores.

2. Se resuelve el problema $PCIL$ con el conjunto actual de multiplicadores $(\lambda_i)$. Sea $(x^{k^*}{}_j)$ la solución óptima de este problema y $Z_{LB} = w(\lambda)$ su correspondiente valor óptimo, es decir, se elige $(x^{k^*}{}_j)$ tal que $1 \leq k^* \leq K$ y $c_{k^*} + \lambda \cdot v_{k^*}$ sea el mínimo de $c_k + \lambda \cdot v_k \forall k = 1, ..., K$.

3. Se definen los *subgradientes* $v_{k^*i}$ para las restricciones relajadas evaluadas en la solución $(x^{k^*}{}_j)$ de la siguiente manera:

$$v_{k^*i} = b_i - \sum_{j=1}^{n} a_{ij} x^{k^*}{}_j \text{ para } i = 1, ..., m$$

4. Se define la magnitud de la iteración $t$ como:

$$t = \frac{\pi(Z_{UB} - Z_{LB})}{\sum_{i=1}^{m} v_{k^*i}^2}$$

5. Se actualiza $\lambda_i$ :

$$\lambda_i = \max(0, \lambda_i + t v_{k^*i}) \text{ para } i = 1, ..., m$$

y se regresa al paso 2 para resolver el problema $PCIL$ con este nuevo conjunto de multiplicadores.

Hay varias reglas para terminar el procedimiento. Una de ellas está basada en el número de iteraciones que se está dispuesto a realizar. Otra regla depende del valor de $\pi$, donde $\pi$ se reduce periodicamente. En este caso el procedimiento termina cuando $\pi$ toma un valor pequeño.

Con el procedimiento de optimización subgradiente no se debe esperar una mejoría continua en las cotas inferiores que se obtienen en cada iteración. Es decir, en cada iteración no necesariamente se alcanza una cota inferior mayor que la calculada en la iteración anterior. De hecho, la cota inferior puede resultar negativa.

Sea $Z_{\max}$ la máxima cota inferior que se ha encontrado de todas las iteraciones realizadas. Al inicio $Z_{\max} = -\infty$, y este valor se actualiza en cada iteración como sigue:

$$Z_{\max} = \max(Z_{\max}, Z_{LB})$$

Lo que se ha observado en la práctica es que el valor $Z_{max}$ se incrementa rápidamente en las primeras iteraciones, pero después de realizar muchas iteraciones este aumento se hace cada vez más lento. Generalmente el valor de $Z_{max}$ se aproxima mucho ( o incluso alcanza ) a la máxima cota inferior, es decir, al valor óptimo del problema dual Lagrangeano. En la siguiente figura se ilustra la situación a medida que se realizan iteraciones. Sobre la línea de valores se marcan las cotas inferiores encontradas en cada iteración. La mejor de estas cotas es $Z_{max}$, es decir la más cercana al valor óptimo.

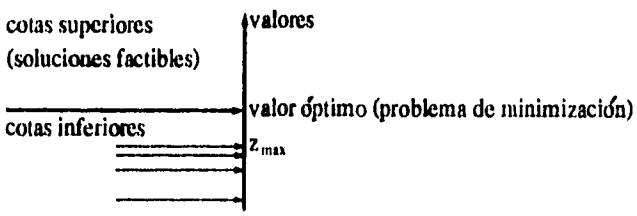



Fig. 2.10

John E. Beasly es un investigador que ha trabajado mucho con la relajación Lagrangeana [13]. De acuerdo con su experiencia se pueden hacer las siguientes observaciones:

1. Es claro que el procedimiento de optimización subgradiente se puede terminar cuando $Z_{max} = Z_{UB}$. En este caso el valor de la máxima cota inferior $Z_{max}$ coincide con el valor de una solución factible $Z_{UB}$, y por lo tanto éste debe ser el óptimo. Ya se había observado que el único lugar en el que una cota inferior y una cota superior pueden ser iguales en la línea de valores es en el punto óptimo.

2. Inicialmente $\pi = 2$. Si $Z_{max}$ no se ha incrementado en las últimas $N$ iteraciones con el valor actual de $\pi$, entonces $\pi$ se reduce a la mitad. Un valor para $N$ de 30 parece ser razonable. Sin embargo, es conveniente experimentar con diferentes valores de $N$ con la intención de ver si se producen resultados significativamente mejores para un problema particular.

3. Si no se ha cumplido la condición de la observación 1 para que se termine el procedimiento, entonces éste puede terminarse cuando $\pi$ sea pequeño, por ejemplo $\leq 0.005$. Generalmente esto significa que se tienen que realizar cientos de iteraciones, pero es inevitable si se desea obtener una cota de buena calidad.

4. De acuerdo con la experiencia de John E. Beasly, los valores iniciales de los multiplicadores de Lagrange no son importantes para el procedimiento. Qué tan rápido se termine el procedimiento es relativamente insensible a la elección inicial de multiplicadores, así como la calidad de la cota inferior $Z_{max}$ que se obtenga.

5. A veces es conveniente reemplazar el valor $Z_{UB}$ en la ecuación:

$$t = \frac{\pi(Z_{UB} - Z_{LB})}{\sum_{i=1}^{m} v_{k*i}^2}$$

por $1.05Z_{UB}$. La razón de esto es la siguiente: si los valores $Z_{UB}$ y $Z_{LB}$ coinciden entonces se ha encontrado la solución óptima . Pero a medida que $Z_{LB}$ se aproxima a $Z_{UB}$ el valor de $t$ se hace más pequeño, y por lo tanto deberán realizarse muchas iteraciones. Si se modifica el valor de $Z_{UB}$ por el de $1.05Z_{UB}$ ( donde $1.05$ se eligió arbitrariamente ) se evita este problema.

**Ejemplo 2.2.3** *Considérese el siguiente problema de recubrimiento* $(P)$:

$$\min\ 2x_1 + 3x_2 + 4x_3 + 5x_4$$

$$s.a \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} x_2 + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} x_3 + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} x_4 \geq \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$x_j = 0 \text{ ó } 1; j = 1, ..., 4$$

*Obsérvese que la solución óptima es* $x_1 = x_2 = 1$ *y* $x_3 = x_4 = 0$. *El valor óptimo es* 5.

*Para obtener el correspondiente problema* $(PCIL)$ *se asocia un multiplicador* $\lambda_i$ *a cada restricción:*

$$\begin{aligned} min \quad & 2x_1 + 3x_2 + 4x_3 + 5x_4 \ + \ \lambda_1(1 - x_1 - x_3) \\ & + \ \lambda_2(1 - x_1 - x_4) \\ & + \ \lambda_3(1 - x_2 - x_3 - x_4) \\ s.a \quad & x_j = 0, 1; j = 1, ..., 4 \end{aligned}$$

*Este problema es equivalente al siguiente problema:*

$$\min c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + \lambda_1 + \lambda_2 + \lambda_3$$

$$s.a \quad x_j = 0 \text{ o } 1; j = 1, , 4$$

*donde*

$c_1 = (2 - \lambda_1 - \lambda_2)$, $c_2 = (3 - \lambda_3)$, $c_3 = (4 - \lambda_1 - \lambda_3)$, $c_4 = (5 - \lambda_2 - \lambda_3)$

*Con el fin de obtener una cota inferior del valor óptimo* $(P)$ *se eligen los siguientes valores para los multiplicadores de Lagrange:* $\lambda_1 = 1.5, \lambda_2 = 1.6$ *y* $\lambda_3 = 2.2$, *entonces* $c_1 = -1.1, c_2 = 0.8, c_3 = 0.3$ *y* $c_4 = 1.2$. *La solución óptima de este problema es* $x_1^* = 1, x_2^* = x_3^* = x_4^* = 0$ *y el valor óptimo es* $Z_{UB} = 4.2$, *el cual es una cota inferior para el valor óptimo de* $(P)$.
*A continuación se ejemplificará una iteración del procedimiento de optimización subgradiente:*
1. *Sea* $\pi = 2$. *Sea* $Z_{UB} = 6$ *(supóngase que se ha aplicado alguna heurística al problema* $(P)$ *y se ha encontrado la solución factible* $x_1 = x_3 = 1, x_2 = x_4 = 0$*). Sea* $\lambda_1 = 1.5, \lambda_2 = 1.6$ *y* $\lambda_3 = 2.2$.
2. *Ya se calculó la solución óptima del correspondiente problema* $(PCIL)$*:* $x_1^* = 1, x_2^* = x_3^* = x_4^* = 0$ *con* $Z_{UB} = 4.2$.
3. *Las ecuaciones para los subgradientes son :* $v_1 = (1 - x_1^* - x_3^*) = 0$, $v_2 = (1 - x_1^* - x_4^*) = 0$ *y* $v_3 = (1 - x_2^* - x_3^* - x_4^*) = 1$.
4. *La magnitud del paso* $t$ *está dada por:* $t = \frac{2(6-4.2)}{0^2+0^2+1^2} = 3.6$
5. *Los nuevos valores de* $\lambda_i$ *son:* $\lambda_1 = \max\{0, 1.5 + 3.6(0)\} = 1.5$, $\lambda_2 = \max\{0, 1.6 + 3.6(0)\} = 1.6$ *y* $\lambda_3 = \max\{0, 2.2 + 3.6(1)\} = 5.8$
*Al resolver el problema* $(PCIL)$ *con este conjunto de multiplicadores se obtiene* $x_1^* = x_2^* = x_3^* = x_4^* = 1$ *y* $Z_{UB} = -0.7$. *Obsérvese que en este caso la cota inferior no mejoró.*

## 2.3 Heurísticos.

**Definición 2.3.1** *Un heurístico es una técnica que busca buenas soluciones (es decir, soluciones cercanas a la óptima ) a un costo computacional razonable, sin garantizar optimalidad o factibilidad; y en muchos casos sin poder afirmar qué tan cerca está del óptimo una solución particular.*

Los algoritmos heurísticos se usan por las siguientes razones. Cuando un problema es $NP-completo$ , es improbable que cualquier algoritmo eficiente garantice encontrar soluciones óptimas cuando el número de variables es grande. En estos casos se pueden tener algoritmos que corran rápido, o que encuentren soluciones óptimas, pero no se pueden tener ambas cosas simultáneamente. Por esta razón se diseñan algoritmos heurísticos eficientes que, aunque no garanticen encontrar el óptimo, generan soluciones cercanas a éste.

Existe otra razón para usar heurísticos. Lo que se optimiza es un modelo de un problema real. No se puede garantizar que la mejor solución del modelo sea también la mejor solución para el problema real. En otras palabras, es preferible tener una solución aproximada de un modelo exacto, que tener una solución exacta de un modelo aproximado. Es difícil tener un modelo exacto, sin embargo, los algoritmos heurísticos generalmente son más flexibles para resolver problemas donde la función objetivo y las restricciones son más complicadas. Es decir, este tipo de algoritmos resuelve modelos más realistas.

Los algoritmos heurísticos se pueden clasificar en procedimientos que construyen una solución y procedimientos que tratan de mejorar sistemáticamente una solución inicial. Dentro de la primera categoría se encuentran, por ejemplo, los algoritmos glotones. Los procedimientos de mejoras sucesivas pertenecen a la segunda categoría.

### 2.3.1 Algoritmos Glotones.

Los algoritmos glotones son aquellos en los que el tomador de decisión selecciona, en una etapa del proceso, la mejor alternativa entre todas las alternativas factibles; sin considerar el impacto que esta elección pueda tener en las subsecuentes decisiones. Generalmente, la mejor alternativa se refiere a la más favorable con respecto a la función objetivo. A continuación se dará un ejemplo de un algoritmo glotón, el algoritmo de Kruskal, el cual resuelve correctamente el problema del árbol expandido de peso mínimo en una gráfica conexa.

**Ejemplo 2.3.1** *Considérese el problema del árbol expandido de peso mínimo. Existen varios algoritmos para resolver este problema, pero el más conocido es el de Kruskal. La idea del algoritmo de Kruskal es seleccionar en cada etapa la arista de menor peso que no forme ciclos, hasta generar un árbol expandido.*

ALGORITMO DE KRUSKAL

*1. Inicializar el conjunto $S$, el cual contiene las aristas del árbol : $S \leftarrow \emptyset$.*

*2. Se incrementa el conjunto $S$. Sea $e$ una de las aristas de menor peso, tal que $e \notin S$ y las aristas $S \cup \{e\}$ no formen ciclos. Se realiza la asignación $S \leftarrow S \cup \{e\}$.*

*3. Este paso determina si se ha construido un árbol expandido. Si $\mid S \mid = \mid V \mid -1$, donde $V$ es el conjunto de vértices de la gráfica, el algoritmo termina en esta etapa. De otra manera regresar al paso 2.*

*Considérese la gráfica $G$:*

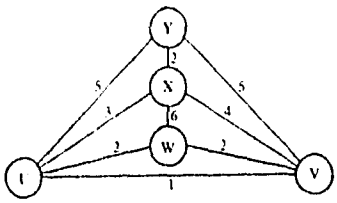



Fig. 2.11

*La siguiente figura muestra cómo se construye el árbol expandido de peso mínimo con el algoritmo deKruskal:*

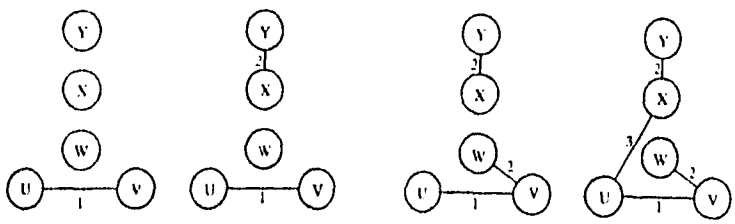



Fig. 2.12

*El algoritmo de Kruskal es un algoritmo glotón ya que sucesivamente selecciona una de las aristas de menor peso, de aquellas aristas restantes que no forman ciclos. Es decir, en cada etapa se selecciona la mejor alternativa entre todas las alternativas factibles.*

### 2.3.2 Mejoras Sucesivas.

Encontrar el óptimo global de una instancia de un problema puede ser muy difícil, sin embargo a menudo es posible hallar una solución $x$ la cual es la mejor en el sentido de que no hay una solución mejor a ésta en una vecindad alrededor de $x$, $N(x,\sigma)$. Donde una vecindad $N(x,\sigma)$ de una solución $x$ es un conjunto de soluciones que pueden ser generadas a partir de $x$ mediante una operación $\sigma$. Esta operación puede consistir en eliminar o agregar o intercambiar objetos de una solución. Si una solución $x$ es mejor que cualquier otra solución en la vecindad $N(x,\sigma)$, entonces $x$ es un óptimo local con respecto a esta vecindad.

Considérese el problema general de optimización discreta:

$$\min\{cx \mid x \in F\} \tag{2.14}$$

donde el conjunto de soluciones factibles $F$ está definido como

$$F = \{x \geq 0 \mid Ax \geq b, x_j = 0 \text{ ó } 1 \quad \forall j \in I\}$$

El algoritmo de mejoras sucesivas, también conocido como mejoras locales, empieza con una solución $x^k \in F$ y trata de mejorar $x^k$ al buscar una solución superior en una vecindad apropiada $N(x^k, \sigma)$ alrededor de $x^k$.

ALGORITMO

PASO 0. Inicialización. Se elige un punto inicial $x^1 \in F$, y se inicializa el contador en 1 : $k \leftarrow 1$.

PASO 1. Mejora. Considérese el conjunto de mejoras locales

$$\{x \in F \cap N(x^k, \sigma); cx < cx^k\} \tag{2.15}$$

Si este conjunto está vacío, el algoritmo termina en esta etapa y $x^k$ es un mínimo local de (2.14) con respecto a la vecindad $N(x, \sigma)$. De otra manera, se elige un elemento del conjunto, $x^{k+1}$, y se incrementa el contador : $k \leftarrow k + 1$. Se repite el paso 1.

Este es un algoritmo muy general y por lo tanto requiere más detalle en al menos dos aspectos. Primero, el procedimiento comienza en el paso 0 con una solución factible $x^1 \in F$. ¿Cómo se obtiene una buena solución inicial cuando no se tiene ninguna disponible? Un camino consiste en utilizar otro heurístico, por ejemplo, un algoritmo glotón.

Cuando no se conoce ninguna solución factible, se puede usar la versión de dos fases del procedimiento de mejoras sucesivas. En esta versión, se comienza con una solución que no es factible, y en la fase I del método de mejoras locales se minimiza la infactibilidad de la solución. Si en esta fase se logra una infactibilidad de cero, entonces la fase II del procedimiento empieza con el punto final de la fase I. La única modificación del procedimiento de mejoras sucesivas durante la fase I consiste en reemplazar el conjunto (2.15) por

$$\{x \in N(x^k, \sigma) \mid d(x) < d(x^k)\}$$

donde la función $d(x)$ mide la infactibilidad de la solución $x$.

Otro aspecto que es necesario especificar es la definición de la vecindad $N(x, \sigma)$. En problemas de optimización continuos, la vecindad se toma como una bola abierta alrededor de $x$. Sin embargo, en modelos discretos es posible que estas bolas no contengan otros puntos de $F$ debido a la discontinuidad

del espacio de soluciones. Por otro lado, si se incluyen todos los puntos en $\mathbb{R}^n$ cuyas componentes difieren de las de $x$ en cantidades de 0 ó 1, se tendrán que evaluar $2^n$ soluciones vecinas. Claramente, una vecindad apropiada debe ser suficientemente grande para incluir algunas soluciones factibles y suficientemente pequeña para que no se complique el análisis. A continuación se presentan varios ejemplos de vecindades.

- VECINDAD UNITARIA. Dada una solución $x^k$, la vecindad unitaria alrededor de $x^k$ es aquélla en la que se complementan las componentes de $x^k$ una a la vez. Es decir,

$$N_1(x^k) = \{x \text{ binaria} ; \sum_j \mid x_j - x_j^k \mid = 1\}$$

- VECINDAD DE T-CAMBIOS. Esta vecindad generaliza la vecindad unitaria al permitir hasta $t$ complementos en las componentes de la solución. Específicamente,

$$N_t(x^k) = \{x \text{ binaria} ; \sum_j \mid x_j - x_j^k \mid \leq t\}$$

- INTERCAMBIO POR PAREJA. En esta vecindad se intercambian 2 componentes binarias a la vez, pero de una manera complementaria.

$$N_{2p}(x^k) = \{x \text{ binaria} ; \sum_j \mid x_j - x_j^k \mid = 2, \sum_j (x_j - x_j^k) = 0\}$$

- VECINDAD DE T-INTERCAMBIOS. En esta vecindad se cambian hasta $t$ valores de una solución de una manera complementaria.

$$N_{tp}(x^k) = \{x \text{ binaria} ; \sum_j \mid x_j - x_j^k \mid \leq t, \sum_j (x_j - x_j^k) = 0\}$$

**Ejemplo 2.3.2** *Para ilustrar el procedimiento de mejoras locales de dos fases considérese el siguiente problema de recubrimiento.*

$$\min \quad 20x_1 + 5x_2 + 4x_3 + 6x_4 + 2x_5$$

*s. a*

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} x_2 + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} x_3 + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} x_4 + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} x_5 \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$x \text{ binaria}$$

*Es posible comenzar el procedimiento con la solución factible* $x = (1,1,1,1,1)$, *sin embargo para mostrar la versión de dos fases se empezará con una solución no factible :* $x^1 = (0,0,0,0,0,)$. *En la fase I se aplicará la vecindad unitaria. Además,* $d(x)$ *se define como el número de renglones que no se satisfacen.*

*La vecindad unitaria* $N_1(0)$ *alrededor de* $x^1$ *está formada por las siguientes soluciones:* $(1,0,0,0,0)$, $(0,1,0,0,0)$, $(0,0,1,0,0,)$, $(0,0,0,1,0)$ *y* $(0,0,0,0,1)$. *Estas soluciones reducen* $d(x^1)$ *en 3, 2, 2, 2 y 2 unidades respectivamente. La solución con menor infactibilidad es* $x^2 = (1,0,0,0,0)$.

*Esta solución aún no es factible, entonces se debe aplicar otra vez la fase I. La vecindad unitaria consiste de las soluciones:* $(0,0,0,0,0,)$, $(1,1,0,0,0)$, $(1,0,1,0,0)$, $(1,0,0,1,0)$ *y* $(1,0,0,0,1)$. *Unicamente la segunda y la cuarta solución disminuyen la infactibilidad ( en una unidad cada una ). Si se selecciona la segunda, la fase I termina con una solución factible* $x^3 = (1,1,0,0,0)$ *y un costo de 25 unidades.*

*La fase II comienza a partir de* $x^3$. *En esta fase se considerará una vecindad de intercambios por parejas. La vecindad alrededor de* $x^3$ *está formada por las siguientes soluciones:* $(0,1,1,0,0)$, $(0,1,0,1,0)$, $(0,1,0,0,1)$, $(1,0,1,0,0)$, $(1,0,0,1,0)$, $(1,0,0,0,1)$. *Ninguna de estas soluciones son factibles, excepto* $(0,1,1,0,0)$ *y* $(1,0,0,1,0)$. *Esta última tiene un costo mayor que* $x^3$, *por lo tanto se elige* $x^4 = (0,1,1,0,0,)$ *con un costo de 9 unidades.*

*Si se aplica la fase II sobre esta nueva solución, se verá que* $x^4$ *no se puede mejorar localmente. Entonces,el algoritmo termina en esta etapa. Por otro lado, es fácil comprobar que el óptimo global se alcanza en* $x^* = (0,0,0,1,1)$, *con un costo de 8 unidades.*

Una modificación de este algoritmo consiste en empezar otra vez todo el procedimiento con un punto inicial diferente, con la esperanza de que converga a un óptimo local diferente. Se elige la mejor de estas dos soluciones generadas. Otra modificación al método de mejoras locales es aquella en donde se permite reiniciar el procedimiento con una solución peor que la anterior. Debe haber algunas restricciones que permitan continuar con una solución peor, de otra manera el procedimiento se incrementaría al buscar en todo el espacio de soluciones. Al usar esta idea existe la posibilidad de que el proceso pueda salir de un óptimo local y encontrar una mejor solución. Un ejemplo de esta versión del procedimiento de mejoras locales es el *recocido simulado*, el cual se estudiará más adelante.

# Capítulo 3

# Algoritmos Específicos para el Problema del Agente Viajero.

Al resolver un *ejemplo* del problema del agente viajero con uno de los algoritmos presentados en el capítulo anterior se pudo observar que a pesar de ser un problema pequeño se tuvieron que hacer muchas iteraciones antes de encontrar la solución óptima. En este capítulo se discutirán algoritmos más eficientes para resolver el problema del agente viajero en donde se aprovecha la estructura de éste, a diferencia de los algoritmos presentados previamente. Se resolverá, con cada algoritmo , el mismo *ejemplo* del problema del capítulo anterior para mostrar que son más eficientes.

En la primera sección de este capítulo se presentan los algoritmos exactos y en la última se describen los heurísticos.

## 3.1 Algoritmos Exactos.

En esta sección se estudiarán dos algoritmos exactos que usan el método de ramificación y acotamiento para obtener la solución óptima. La diferencia principal entre estos algoritmos es la estrategia de acotamiento que usan, en uno se emplea la relajación Lagrangeana y en otro el problema de asignación.

### 3.1.1 El Problema de Asignación con la Función de Costos del Problema del Agente Viajero.

En el capítulo 1 se vió que la formulación del problema del agente viajero es:

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

s.a

$$\begin{aligned} \sum_{j \in V} x_{ij} &= 1 \quad \forall i \in V \\ \sum_{i \in V} x_{ij} &= 1 \quad \forall j \in V \\ x_{ij} &= 0,1 \quad \forall i,j \in V \\ x &= (x_{ij}) \text{ es la asignación de un } \textit{tour} \end{aligned}$$

La relajación más sencilla para el problema del agente viajero y también la primera en ser usada es la que se obtiene de la formulación anterior al eliminar la restricción de que $x = (x_{ij})$ debe ser la asignación de un *tour*. Es decir, el problema de asignación con la misma función de costos que la del problema del agente viajero, el cual se resuelve eficientemente mediante el *método húngaro*. Esta es la estrategia de acotamiento que se usará.

**Estrategia de Acotamiento.**

Considérese un *ejemplo* del problema del agente viajero con matriz de costos $C$. Se resuelve el problema relajado, es decir el problema de asignación con matriz de costos $C$, mediante el *método húngaro*. Sea $\overline{C}$ la matriz de costos reducida que se obtiene y sea $r_0$ el valor óptimo de este problema. Recuérdese que la matriz que se obtiene de la matriz original de costos al restarle una constante a todas las entradas de un rengón o una columna se le conoce como matriz de costos reducida. Entonces, para cualquier asignación de un *tour* $x$ se tiene que:

$$Z_C(x) = r_0 + Z_{\overline{C}}(x) \tag{3.1}$$

Ahora, se sabe que la matriz $\overline{C}$ , es no negativa. Entonces,

$$r_0 \leq Z_C(x) \quad \text{para toda } x \text{ asignación de un } \textit{tour} \tag{3.2}$$

por lo tanto $r_0$ es una cota inferior para el problema original del agente viajero.

Supóngase que la asignación óptima $x^*$ es la asignación de un *tour*, por lo tanto el valor de $x^*$ en la matriz de costos reducida $\overline{C}$ es cero. Entonces

$$\begin{aligned} & Z_C(x^*) = r_0 + Z_{\overline{C}}(x^*) \\ \Rightarrow \quad & Z_C(x^*) = r_0 + 0 \\ \Rightarrow \quad & Z_C(x^*) = r_0 \end{aligned}$$

de aquí que la igualdad en (3.2) se da cuando la asignación óptima es la asignación de un *tour*. En este caso, ésta es la solución óptima para el problema del agente viajero.

**Estrategias de ramificación.**

El problema del agente viajero es un problema binario, por esta razón se usará un procedimiento de enumeración implícita para resolverlo. Recuérdese que en este tipo de procedimientos se empieza con una o más variables que se fijan en un valor binario y gradualmente se construye la solución al aumentar el número de variables fijas. Entonces, la regla de ramificación selecciona alguna variable libre $x_{rs}$ y crea dos subproblemas cuando se fija $x_{rs}$ en uno y en cero. Al ramificar el problema original (subproblema 1) se generan los siguientes subproblemas:

$$\begin{aligned} \text{subproblema 2} &= \text{subproblema 1} + (x_{rs} = 0) \\ \text{subproblema 3} &= \text{subproblema1} + (x_{rs} = 1) \end{aligned}$$

Si se elige el subproblema 2 para acotarlo inferiormente se debe resolver el problema de asignación cuya matriz de costos es $C_1$. Esta matriz se obtiene a partir de la matriz $\overline{C}$ al modificar la entrada $\overline{C}(r,s)$ por $M$, donde $M >> 0$, ya que si $x_{rs} = 0$ no se puede viajar de la ciudad $r$ a la ciudad $s$. Por otro lado, sea $\underline{C}$ la matriz que se obtiene a partir de la matriz original $C$ al modificar la entrada $c(r,s)$ por $M$. Entonces, para cualquier asignación $x$ de un *tour* se tiene que

$$Z_{\underline{C}}(x) = r_0 + Z_{C_1}(x)$$

Supóngase que el valor óptimo del problema de asignación asociado con el subproblema 2 es $r_1$ y sea $\overline{C_1}$ la matriz de costos reducida. Entonces, para cualquier asignación de un *tour* $x$ se tiene que:

$$Z_{C_1}(x) = r_1 + Z_{\overline{C_1}}(x)$$

pero

$$Z_{\underline{C}}(x) = r_0 + Z_{\underline{C}_1}(x) \qquad \text{para toda } x \text{ asignación de un } \textit{tour}$$
$$\Rightarrow Z_{\underline{C}}(x) = r_0 + r_1 + Z_{\overline{C}_1}(x) \qquad \text{para toda } x \text{ asignación de un } \textit{tour}$$

Además, se sabe que $\overline{C}_1 \geq 0$, por lo que

$$r_0 + r_1 \leq Z_{\underline{C}}(x) \qquad \text{para toda } x \text{ asignación de un } \textit{tour}$$

entonces $r_0 + r_1$ es una cota inferior para el valor objetivo de cualquier asignación de un *tour* con matriz de costos $\underline{C}$. Es decir, $r_0 + r_1$ es una cota inferior para el subproblema 2.

Se sabe que una cota inferior para el valor óptimo del problema original es $r_0$. También se calculó una cota inferior para el valor óptimo del subproblema 2, $r_0 + r_1$. Entonces, $r_1$ es la cantidad por la cual la cota inferior del subproblema 2 es mayor que la cota inferior para el problema original. Se desea seleccionar la variable de ramificación de tal manera que la cantidad $r_1$ sea lo más grande posible. Si se selecciona de esta manera la variable de ramificación, entonces la cota inferior para al menos uno de los subproblemas generados será lo más grande posible.

Sin embargo, para determinar la cantidad $r_1$ de cada variable es necesario resolver un problema de asignación. Esto requiere de muchos cálculos y por lo tanto es ineficiente. Entonces, es necesario desarrollar un criterio para seleccionar la variable de ramificación que no involucre muchos cálculos y que haga una elección razonable.

Para ello se define la *evaluación de una variable* con respecto a la matriz de costos reducida $\overline{C}$. Para cada variable $x_{ij}$ se calcula:

$$P_{ij} = \min\{\overline{c}_{ih};\quad h \neq j\} + \min\{\overline{c}_{hj};\quad h \neq i\}$$

$P_{ij}$ es la mínima cantidad por la cual el valor óptimo del subproblema se incrementa si la variable $x_{ij}$ se fija en cero. La cantidad $r_1$ es mayor o igual que la evaluación de cada variable. Entonces, una buena regla para seleccionar la variable de ramificación es elegir aquélla cuya evaluación sea la más alta. Por lo tanto, se elige la variable de ramificación $x_{rs}$ tal que

$$P_{rs} = \max\{P_{ij}\}$$

Obsérvese que para aquellas variables $x_{ij}$ donde $\overline{c}_{ij} \neq 0$ la evaluación de la variable $P_{ij}$ es cero ya que en la matriz reducida $\overline{C}$ al menos hay un cero en

cada renglón y en cada columna. Entonces, basta calcular la evaluación de aquellas variables $x_{ij}$ donde $\bar{c}_{ij} = 0$.

Por último, cabe hacer notar que si se fija la variable $x_{rs}$ en uno, para evitar un *subtour* entre las ciudades $r$ y $s$, la variable $x_{rs}$ debe fijarse en cero.

### Algoritmo.

En cada iteración $t$ de este algoritmo se realizan los siguientes pasos:

1. Se inicializa la cota superior $Z_{UB}$: $Z_{UB} = \infty$.

2. Si faltan nodos del árbol por analizar se elige alguno con la regla de la cota más reciente o con la estrategia de prioridad. Por el contrario, si todos los nodos están analizados el algoritmo termina en esta etapa. Si en este momento se tiene una solución candidata, ésta es la solución óptima del problema original. De lo contrario el problema es infactible.

3. Se acota inferiormente el subproblema seleccionado al resolver el problema de asignación asociado con este subproblema. Sea $Z^t$ el valor óptimo del problema de asignación. Si la asignación óptima corresponde a la asignación de un *tour* entonces ésta constituye una solución candidata. En este caso, se actualiza el valor de la cota superior, $Z_{UB} = Z^t$. Regresar al paso 2. Si la asignación óptima no es factible para el problema original entonces ir al paso 4.

4. Se ramifica el subproblema sobre alguna variable libre $x_{rs}$ generando de esta manera dos subproblemas, uno con $x_{rs} = 0$ y otro con $x_{rs} = 1$ y $x_{sr} = 0$. La variable de ramificación es aquélla cuya evaluación sea la más alta. Regresar al paso 2.

**Ejemplo 3.1.1** *Considérese el* ejemplo *del problema del agente viajero del ejemplo 2.2.2.*
ITERACIÓN 1.*Se inicializa la cota superior $Z_{UB}$ en $\infty$. Se resuelve el problema relajado de asignación correspondiente al problema original. La ma-*

*triz de costos reducida $\bar{C}^1$ que se obtiene es la siguiente:*

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | $M$ | 0 | 2 | 0 | 1 |
| 2 | 0 | $M$ | 1 | 0 | $M$ |
| 3 | 2 | 1 | $M$ | 0 | 0 |
| 4 | 0 | 0 | 0 | $M$ | $M$ |
| 5 | 1 | $M$ | 0 | $M$ | $M$ |

*La asignación óptima es $1-2-4-1, 3-5-3$ donde $Z^1 = 10$. Dado que esta solución tiene subtours se debe seleccionar una variable para ramificar el problema. Para ello se calcula la evaluación de aquellas variables $x_{ij}$ tales que $\bar{c}_{ij} = 0$.*

| *variable $x_{ij}$* | *evaluación $P_{ij}$* |
|---|---|
| $x_{12}, x_{14}, x_{21}, x_{24}, x_{34}, x_{41}, x_{42}, x_{43}$ | 0 |
| $x_{35}, x_{53}$ | 1 |

*donde $P_{35}$ y $P_{53} = \max\{P_{ij}\}$. Debido a que hay un empate se elige arbitrariamente alguna de estas dos variables, por ejemplo $x_{35}$. Al fijar $x_{35}$ en uno y en cero se generan los siguientes subproblemas:*

$$\begin{aligned} \textit{subproblema 2} &= \textit{subproblema 1} + (x_{35} = 0) \\ \textit{subproblema 3} &= \textit{subproblema 1} + (x_{35} = 1 \text{ y } x_{53} = 0) \end{aligned}$$

*Estos subproblemas se incluyen en la lista de subproblemas no insondeables.* ITERACIÓN 2. *La lista de nodos no insondeables está formada por los subproblemas 2 y 3. En este ejemplo se usará la regla de la cota más reciente para seleccionar un subproblema. Los dos subproblemas de la lista se generáron al mismo tiempo entonces se elige uno arbitrariamente, por ejemplo el subproblema 2. La matriz de costos de este subproblema es*

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | $M$ | 0 | 2 | 0 | 1 |
| 2 | 0 | $M$ | 1 | 0 | $M$ |
| 3 | 2 | 1 | $M$ | 0 | $M$ |
| 4 | 0 | 0 | 0 | $M$ | $M$ |
| 5 | 1 | $M$ | 0 | $M$ | $M$ |

*La matriz de costos reducida que se obtiene al resolver el problema de asignación es*

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | *M* | *0* | *2* | *0* | *0* |
| 2 | *0* | *M* | *1* | *0* | *M* |
| 3 | *2* | *1* | *M* | *0* | *M* |
| 4 | *0* | *0* | *0* | *M* | *M* |
| 5 | *1* | *M* | *0* | *M* | *M* |

*La asignación óptima es 1-5-3-4-2-1 donde* $Z^2 = 11$. *Esta asignación es la asignación de un "tour", por lo tanto ésta constituye una solución candidata. La cota superior para el valor óptimo del problema original es* $Z_{UB} = 11$.

ITERACIÓN 3. *El único nodo no insondeable de la lista corresponde al subproblema 3. La matriz de costos de este subproblema es la siguiente:*

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | *M* | *0* | *2* | *0* |
| 2 | *0* | *M* | *1* | *0* |
| 4 | *0* | *0* | *0* | *M* |
| 5 | *1* | *M* | *0* | *M* |

*Obsérvese que se ha eliminado el tercer renglón y la quinta columna debido a que la variable* $x_{35}$ *está fija en uno. La matriz de costos reducida para este subproblema es*

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | *M* | *0* | *2* | *0* |
| 2 | *0* | *M* | *1* | *0* |
| 4 | *0* | *0* | *0* | *M* |
| 5 | *0* | *M* | *0* | *M* |

*La asignación óptima es 1-2-4-3-5-1 donde* $Z^3 = 11$. *Esta asignación corresponde a la asignación de un "tour" y de hecho coincide con la solución del subproblema 2 debido a que se trata de un problema simétrico. Por lo tanto ésta es la solución óptima para el problema original y el valor óptimo es* $Z^* = 11$. *El árbol de ramificación y acotamiento generado es el siguiente*

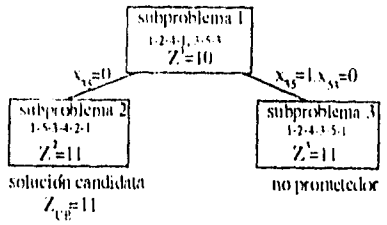



Fig. 3.1

### 3.1.2 El Problema del 1-Arbol con Función Objetivo Lagrangeana.

En esta sección se describirá la relajación del 1-árbol con función objetivo Lagrangeana. Con esta relajación se obtendrán cotas inferiores bastante cercanas al valor óptimo del problema simétrico del agente viajero. La relajación del 1-árbol con función objetivo Lagrangeana también puede usarse para resolver el problema asimétrico del agente viajero. Sin embargo, la experiencia computacional indica que para problemas asimétricos esta relajación es menos eficiente que la del problema de asignación. Además, se estudiará este algoritmo porque para el problema simétrico del agente viajero los métodos de búsqueda que usan la relajación del 1-árbol son más eficientes que aquéllos que usan la relajación de asignación, debido a que se obtienen cotas más cercanas al óptimo.

Sea $G = (V, E)$ una gráfica no dirigida. El problema del *árbol expandido de peso mínimo* consiste en encontrar el árbol de peso mínimo en el conjunto de vértices $V = \{1, 2, ..., n\}$. En este algoritmo se usará una variante de este problema: el *1-árbol de peso mínimo*. Un 1-árbol consiste de un árbol en el conjunto de vértices $\{2, ..., n\}$ junto con dos aristas distintas incidentes al vértice 1. En la siguiente figura se muestra un 1-árbol:

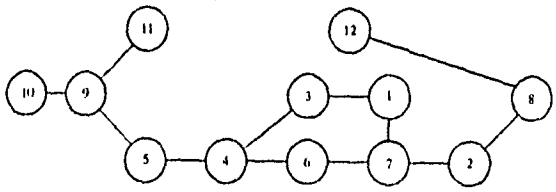



Fig. 3.2 Un 1-árbol

Obsérvese que un 1-árbol tiene sólo un ciclo, en el cual está incluido el vértice 1. Además este vértice siempre tiene grado 2.

Nótese que:

1. Un *tour* es un 1-árbol en el cual cada vértice tiene grado 2. Entonces, todo *tour* es un 1-árbol y un 1-árbol es un *tour* sólo si cada uno de sus vértices tiene grado 2. Por lo tanto, el problema del 1-árbol de peso mínimo constituye una relajación para el problema del agente viajero.

2. El 1-árbol de peso mínimo es fácil de calcular. Este puede encontrarse al construir el árbol de peso mínimo en el conjunto de vértices $\{2, ..., n\}$ y después agregar las dos aristas de menor peso incidentes al vértice 1.

3. La transformación de las distancias $c_{ij} \rightarrow c_{ij} + \lambda_i + \lambda_j$ deja el problema del agente viajero invariante pero modifica el 1-árbol de peso mínimo. Esto se aclara con el siguiente lema:

**Lema 3.1.1** *Sea $\lambda = (\lambda_1, \lambda_2, ..., \lambda_n)$ un vector con componentes reales. Si $x^*$ es el "tour" de menor costo con respecto a las distancias $c_{ij}$, entonces también es el "tour" de menor costo con respecto a los pesos $c_{ij} + \lambda_i + \lambda_j$.*

DEMOSTRACIÓN. El peso de un *tour* $x$, con respecto a las distancias $c_{ij}$ es :

$$\sum_{(i,j)\in x} c_{ij} \qquad (*)$$

y con respecto a las distancias $c_{ij} + \lambda_i + \lambda_j$ es :

$$\sum_{(i,j)\in x} (c_{ij} + \lambda_i + \lambda_j) \qquad (**)$$

Si se resta $(**) - (*)$ se tiene que:

$$\begin{aligned}
\sum_{(i,j)\in x} (C_{ij} + \lambda_i + \lambda_j) - \sum_{(i,j)\in x} C_{ij} &= \sum_{(i,j)\in x} C_{ij} + \sum_{(i,j)\in x} \lambda_i + \sum_{(i,j)\in x} \lambda_j - \sum_{(i,j)\in x} C_{ij} \\
&= \sum_{(i,j)\in x} \lambda_i + \sum_{(i,j)\in x} \lambda_j \\
&= \sum_{i=1}^{n} \lambda_i + \sum_{i=1}^{n} \lambda_i \\
&= 2\sum_{i=1}^{n} \lambda_i
\end{aligned}$$

Entonces, al cambiar un conjunto de pesos por otro el costo de todos los *tours* se altera en la misma cantidad y por lo tanto no se modifica la comparación entre los *tours*. ■

Si se considera la observación *3*, una posible estrategia para resolver el problema del agente viajero consiste en encontrar un vector $\lambda$ tal que el 1-árbol de peso mínimo con respecto a las distancias $c_{ij} + \lambda_i + \lambda_j$ sea un *tour*.

### Estrategia de Acotamiento : Formulación del Problema PCIL.

Se sabe que todo *tour* es un 1-árbol en el cual cada vértice tiene grado 2. En base a esta observación se puede formular el problema del agente viajero como el problema del 1-árbol de peso mínimo con la restricción adicional de que cada vértice debe tener grado 2. Estas restricciones se pueden escribir como:

$$d_{ik} = 2 \; \forall i \in \{1, 2, ..., n\}$$

donde $d_{ik}$ es el grado del vértice $i$ en el $k - ésimo$ 1-árbol.

Para formular el problema de la cota inferior Lagrangeana ($PCIL$) se elige este conjunto de restricciones con el fin de relajar el problema original. A cada una de estas restricciones se le asocia un multiplicador de Lagrange $\lambda_i$ y se incluye en la función objetivo. El problema $PCIL$ generado es el siguiente:

$$\min_k [c_k + \sum_{i=1}^{n} \lambda_i (d_{ik} - 2)]$$

donde $c_k$ es el costo del $k - ésimo$ 1-árbol con respecto a los pesos $c_{ij}$. Este problema también se puede escribir de la siguiente manera:

$$w\,(\lambda) = \min_k [c_k + \lambda V_k]$$

donde $V_k$ es un n-vector cuya $i - ésima$ componente es $d_{ik} - 2$.

En el capítulo 2 se mostró que para cualquier $\lambda$, $w\,(\lambda)$ es una cota inferior para el valor óptimo del problema original, ya que las restricciones que se relajaron corresponden a igualdades. Esto también se puede comprobar de la siguiente manera. Claramente, el costo del *tour* óptimo es mayor o igual al costo del 1-árbol de peso mínimo con respecto a las distancias $c_{ij} + \lambda_i + \lambda_j$. El costo del *tour* óptimo con respecto a estas distancias es :

$$c^* + 2\sum_{i=1}^{n} \lambda_i$$

Por otro lado, el costo del 1-árbol de peso mínimo con respecto a las distancias $c_{ij} + \lambda_i + \lambda_j$ es :

$$\min_k [c_k + \sum_{i=1}^{n} \lambda_i d_{ik}]$$

Entonces,

$$\begin{aligned}
c^* + 2\sum_{i=1}^{n} \lambda_i &\geq \min_k \left[ c_k + \sum_{i=1}^{n} \lambda_i d_{ik} \right] \\
\Rightarrow \quad c^* &\geq \min_k \left[ c_k + \sum_{i=1}^{n} \lambda_i d_{ik} \right] - 2\sum_{i=1}^{n} \lambda_i \\
\Rightarrow \quad c^* &\geq \min_k \left[ c_k + \sum_{i=1}^{n} \lambda_i (d_{ik} - 2) \right] \\
\Rightarrow \quad c^* &\geq \min_k [c_k + \lambda V_k] \\
\Rightarrow \quad c^* &\geq w(\lambda)
\end{aligned}$$

De esta manera se obtiene una familia infinita de cotas inferiores para el costo del *tour* óptimo. La mejor de estas cotas es :

$$\max_{\lambda} \{w(\lambda)\}$$

Este es el problema dual Lagrangeano. Se puede usar el método iterativo de optimización subgradiente para calcular o aproximar el valor $\max_\lambda \{w(\lambda)\}$. Con las iteraciones se calcula una sucesión de n-vectores $\{\lambda^m\}$ de acuerdo a la siguiente fórmula:

$$\lambda^{m+1} = \lambda^m + t_m V_{k(\lambda^m)}$$

donde $k(\lambda)$ es el índice que identifica al 1-árbol de peso mínimo con respecto a las distancias $c_{ij} + \lambda_i + \lambda_j$ y $\{t_m\}$ es una sucesión de escalares que se calculan de la siguiente manera:

$$t_m = \pi_m \left( \frac{Z_{UB} - w(\lambda^m)}{\| V_{k(\lambda^m)} \|^2} \right) \text{ con } 0 < \pi_m \leq 2;$$

$w(\lambda^m)$ constituye una cota inferior , a la cual se le llamó $Z_{IB}$ en el capítulo anterior.

Los algoritmos de ramificación y acotamiento particionan sucesivamente el conjunto de soluciones factibles de un problema y calculan una cota inferior para el valor óptimo de cada subproblema generado. En el caso del problema del agente viajero esta partición generalmente se realiza al incluir y excluir un conjunto de aristas. Los subproblemas generados al incluir el conjunto de aristas $X$ y excluir el conjunto de aristas $Y$ son de la misma forma que el problema original. Por esta razón el método que se acaba de describir para acotar inferiormente un problema también se aplica a los subproblemas generados durante el proceso de ramificación. Sea $T(X,Y)$ el conjunto de todos los 1-árboles que incluyen las aristas en $X$ y excluyen las aristas en $Y$ y sea

$$w_{X,Y}(\lambda) = \min_{k \in T(X,Y)} \{c_k + \lambda V_k\}$$

Entonces $w_{X,Y}(\lambda)$ es una cota inferior para el valor óptimo del subproblema generado.

**Regla de ramificación.**

Considérese el problema $PCIL$:

$$\begin{array}{lcc} \min & & cx + \lambda(b - Ax) \\ & = & (c - \lambda A)x + \lambda b \\ s.a & & Bx \geq d \\ & & x \in \{0,1\} \end{array}$$

En un algoritmo de ramificación y acotamiento basado en la relajación Lagrangeana, una regla de ramificación razonable es la siguiente:

1. Se denota por $(\lambda^*, x^*)$ a los multiplicadores y a los valores de las variables asociadas con la mejor cota inferior del nodo que se va a ramificar. Considérese el vector de valores $\lambda^*(b - Ax^*)$. Se elige la restricción con el mayor valor (absoluto) en este vector, a la cual se hará referencia como restricción $i$. Si $x$ es una solución factible y satisface que $\lambda(b - Ax) = 0$, entonces ésta es la solución óptima para el problema original.Se elige la restricción (es decir la entrada) con el máximo valor absoluto en el vector $\lambda^*(b - Ax^*)$ para que al ramificar se intente hacer cero este vector.

2. Defínase

$$S = \{j \mid x_j \text{ aparezca en la } i - \acute{e}sima \text{ restricción y } x_j^* = 1\}$$

si $S = \phi$ (es decir ninguna de las variables de la $i - \acute{e}sima$ restricción aparece en la solución $x^*$ del problema $PCIL$) entonces defínase

$$S = \{j \mid x_j \text{ aparezca en la } i - \acute{e}sima \text{ restricción y } x_j^* = 0\}$$

La variable del conjunto $S$ que se elige para fijarla en uno es aquélla que tenga el menor coeficiente $(c - \lambda^* A)$ en la función objetivo del problema $PCIL$. Es decir, se elige la variable que más convenga que tome el valor de 1 en la solución óptima. Cuando se elige la variable de ramificación no se consideran aquéllas que están fijas en 1 ó 0.

Con el fin de obtener la variable de ramificación en el problema del agente viajero, se usará otra notación para definir el problema $PCIL$. La restricción que establece que cada vértice debe tener grado 2 ($d_{ik} = 2, \forall i \in V$) se puede escribir de otra manera:

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

Entonces, el problema $PCIL$ generado a partir del problema del agente viajero se puede reformular como:

$$\min_{x \in X(k)} \left[ \sum_{i \in V} \sum_{j>i} c_{ij} x_{ij} + \sum_{i \in V} \lambda_i \left( 2 - \sum_{j<i} x_{ji} - \sum_{j>i} x_{ij} \right) \right]$$

$$= \min_{x \in X(k)} \left[ \sum_{i \in V} \sum_{j>i} (c_{ij} + \lambda_i + \lambda_j) x_{ij} \right] - 2 \sum_{i \in V} \lambda_i$$

donde $X(k)$ es el conjunto de todos los 1-árboles en la gráfica. La regla de ramificación es la siguiente:

1. Sea $\lambda^*$ el vector de multiplicadores y $x^*$ el vector de valores de las variables asociadas con la mejor cota inferior del nodo que se va a ramificar. Considérense los valores $\lambda_i^*(2 - \sum_{j<i} x_{ji}^* - \sum_{j>i} x_{ij}^*)$ para $i \in V$. Se elige la restricción asociada con el mayor valor (absoluto) de estos valores, a la cual se hará referencia como restricción $k$.

2. Se define

$$S = \{x_{ij} \mid x_{ij} \text{ aparezca en la } k\text{-ésima restricción y } x_{ij}^* = 1\}$$

si $S = \emptyset$, entonces se define

$$S = \{x_{ij} \mid x_{ij} \text{ aparezca en la } k\text{-ésima restricción y } x_{ij}^* = 0\}$$

La variable del conjunto $S$ que se elige para fijarla en uno es aquélla que tenga el menor coeficiente $(c_{ij} + \lambda_i + \lambda_j)$ en la función objetivo del problema $PCIL$.

**Algoritmo.**

Cualquier etapa del procedimiento de ramificación y acotamiento se especifica mediante una lista de subproblemas generados con sus correspondientes cotas inferiores. Las entradas son de la forma :

$$(\ X,\ Y,\ \lambda,\ w_{X,Y}\ (\lambda)\ )$$

Al inicio, la lista consiste de una sola entrada: ( $\phi$, $\phi$, 0, $w(0)$ ) donde 0 es un $n-$vector. El algoritmo de ramificación y acotamiento es el siguiente:

1. Se inicializa la cota superior $Z_{UB}$ con alguna heurística. Se inicializan los multiplicadores de Lagrange, $\lambda = 0$.

2. Si faltan nodos por analizar se selecciona una entrada $(X, Y, \lambda, w_{X,Y}(\lambda))$ con la regla de la última cota o con la estrategia de prioridad y se actualiza el vector $\lambda$ : $\lambda^{m+1} = \lambda^m + t_m V_{k(\lambda^m)}$. Ir al paso 3.
   Si todos los subproblemas son insondeables el algoritmo termina en esta etapa. La solución óptima es la solución candidata que se tenga en este momento. Si no hay ninguna solución candidata el problema es infactible.

3. Se resuelve el problema $PCIL$ con el vector de multiplicadores $\lambda^{m+1}$. La cota inferior que se obtenga se puede mejorar con el método de optimización subgradiente. Al resolver este problema se obtiene alguno de los siguientes resultados:
   a. La cota inferior $w_{X,Y}(\lambda)$ del subproblema correspondiente es mayor que la cota superior $Z_{UB}$. En este caso se elimina el nodo de consideración y se regresa al paso 2.

b. La solución del problema $PCTL$ es factible para el problema original y por lo tanto se tiene una solución candidata. Regresar al paso 2.
c. La cota inferior no cumple con las observaciones de los dos puntos anteriores. Ir al paso 4.

4. Ramificar el subproblema con la regla descrita anteriormente y regresar al paso 2.

**Ejemplo 3.1.2** *Considérese el ejemplo 2.2.2. La gráfica asociada a este problema es la siguiente:*

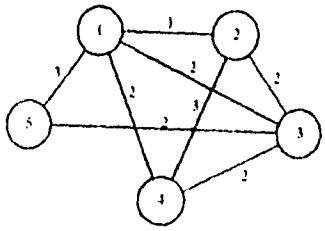



Fig. 3.3

ITERACIÓN 1. *Supóngase que se ha aplicado alguna heurística y se ha encontrado la siguiente solución* $1-4-2-3-5-1$, *donde* $Z_{UB}=12$. *Se inicializan los multiplicadores en* $\lambda=0$. *Se resuelve el problema PCTL con estos multiplicadores. Como el vector* $\lambda$ *es el vector cero, entonces esto equivale a encontrar el 1-árbol de peso mínimo de la gráfica anterior, el cual se muestra a continuación:*

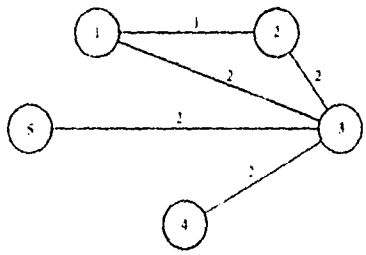



Fig. 3.4

*El valor óptimo del 1-árbol es* $w(0)=9$.
*Con el fin de mejorar esta cota inferior se aplicará el método de optimización subgradiente. Para ello se actualiza el vector* $\lambda$ :

$$\lambda^t=\lambda+tv_k$$

*donde* $v_k = d_k - 2 = (2-2, 2-2, 4-2, 1-2, 1-2) = (0, 0, 2, -1, -1)$
*y*

$$t = \pi \left( \frac{Z_{UB} - w(0)}{\|v_{k(\nu)}\|^2} \right) = 2 \left( \frac{12 - 9}{0^2 + 0^2 + 2^2 + (-1)^2 + (-1)^2} \right) = 1$$

$\Rightarrow \lambda^1 = (0,0,0,0,0) + (0,0,2,-1,-1) = (0,0,2,-1,-1)$
*Los pesos* $c_{ij} + \lambda_i^1 + \lambda_j^1$ *se muestran en la siguiente figura:*

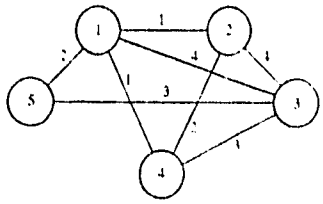



Fig. 3.5

*Se resuelve el problema PCTL con los nuevos costos. El 1-árbol de peso mínimo es*

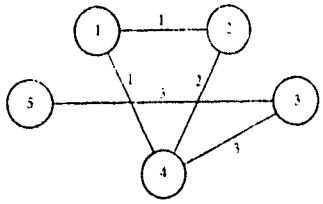



Fig. 3.6

*El valor óptimo es* $w(\lambda^1) = 10$.
*Debido a que el 1-árbol no corresponde a un "tour" se debe ramificar el problema. (paso 1).Para ello se calcula el vector* $\lambda_i^1(2 - \sum_{j<i} x_{ji}^* - \sum_{j>i} x_{ij}^*)$ *para* $i = 1, \ldots, 5$ *donde*

$$x_{ij}^* = \begin{cases} 1 & si \\ 0 & en\ otro\ caso \end{cases} \qquad (i,j) = \{(1,2),(1,4),(2,4),(3,4),(3,5)\}$$

$$\lambda_1^1 \left( 2 - \sum_{j<1} x_{j1}^* - \sum_{j>1} x_{1j}^* \right) = 0\,[2 - (x_{12}^* + x_{13}^* + x_{14}^* + x_{15}^*)] = 0$$

$$\lambda_2^1\left(2-\sum_{j<2}x_{j2}^*-\sum_{j>2}x_{2j}^*\right) = 0\,[2-x_{12}^*-(x_{23}^*+x_{24}^*+x_{25}^*)]=0$$

$$\lambda_3^1\left(2-\sum_{j<3}x_{j3}^*-\sum_{j>3}x_{3j}^*\right) = 2\,[2-(x_{13}^*+x_{23}^*)-(x_{34}^*+x_{35}^*)]=0$$

$$\lambda_1^1\left(2-\sum_{j<1}x_{j1}^*-\sum_{j>1}x_{1j}^*\right) = -1\,[2-(x_{14}^*+x_{24}^*+x_{34}^*)-x_{45}^*]=1$$

$$\lambda_5^1\left(2-\sum_{j<5}x_{j5}^*-\sum_{j>5}x_{5j}^*\right) = -1\,[2-(x_{15}^*+x_{25}^*+x_{35}^*+x_{45}^*)]=-1$$

*Por otro lado*

$$\max\{\ \ |\lambda_i^*(2-\sum_{j<i}x_{ji}^*-\sum_{j>i}x_{ij}^*)|\quad \textit{para } i=1,\ldots,5\ \ \}=1$$

*este valor corresponde a la cuarta restricción.*
*(paso 2). Sea*

$S=\{x_{ij}\mid x_{ij}$ *aparezca en la cuarta restricción y* $x_{ij}^*=1\}=\{x_{14},x_{24},x_{34}\}$

*Estas variables tienen los siguientes costos en la función objetivo del problema PCTL:*

$$\begin{array}{lrcl} \textit{para } x_{14}: & c_{14}+\lambda_1^1+\lambda_4^1 & = & 2+0-1 \;=\; 1\\ \textit{para } x_{24}: & c_{24}+\lambda_2^1+\lambda_4^1 & = & 3+0-1 \;=\; 2\\ \textit{para } x_{34}: & c+\lambda_3^1+\lambda_4^1 & = & 2+2-1 \;=\; 3 \end{array}$$

*El mínimo de estos valores es 1, por esta razón se elige la variable $x_{14}$ para ramificarla. Se generan los siguientes subproblemas:*

$$\begin{array}{lcl} \textit{subproblema } 2 & = & \textit{subproblema } 1+(x_{14}=0)\\ \textit{subproblema } 3 & = & \textit{subproblema } 1+(x_{14}=1) \end{array}$$

ITERACIÓN 2. *Se elige la regla de la última cota para seleccionar un subproblema. Puesto que el subproblema 2 y el 3 se generaron al mismo tiempo, se elige uno arbitrariamente por ejemplo el 2.*
*Se actualiza el vector $\lambda$:*

$$\lambda^2=\lambda^1+t_1v_{k(\lambda^1)}=(0,0,2,1,-3)$$

*Los pesos $c_{ij} + \lambda_i^2 + \lambda_j^2$ se muestran en la siguiente figura:*

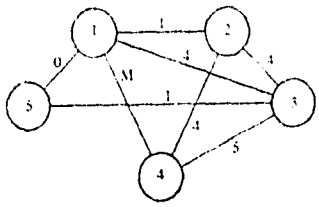



Fig. 3.7

*Al resolver el problema PCTL con los nuevos costos, se obtiene la siguiente solución óptima:*

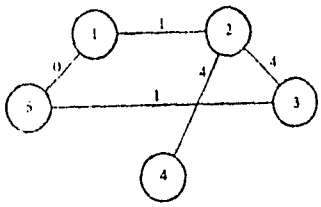



Fig. 3.8

*El valor óptimo es* $w(\lambda^2) = 10$.

*A continuación se aplicará el método de optimización subgradiente para mejorar esta cota. Se actualiza el vector* $\lambda^2$ :

$$\lambda^3 = \lambda^2 + t_2 v_{k(\lambda^2)} = (0, 2, 2, -1, -3)$$

*En la siguiente figura se muestra la gráfica con los costos* $c_{ij} + \lambda_i^3 + \lambda_j^3$.

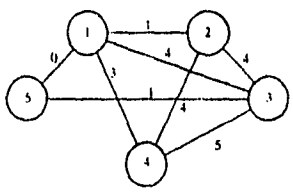



Fig. 3.9

*El 1-árbol óptimo es el siguiente:*

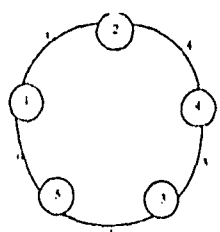



Fig. 3.10

*donde* $w(\lambda^3) = 11$. *Esta solución corresponde a un "tour" x y por lo tanto es una solución candidata. El valor óptimo de este "tour" x con respecto a los costos originales es* $\sum_{(i,j)\in x}(c_{ij} + \lambda_i^3 + \lambda_j^3) - 2\sum_{i=1}^{5}\lambda_i^3 = 11$.

ITERACIÓN 3. *El único subproblema que falta por analizar es el subproblema 3. El vector* $\lambda^2$ *se calculó en la iteración anterior. A continuación se muestran los pesos* $c_{ij} + \lambda_i^2 + \lambda_j^2$ *de la gráfica correspondiente a este subproblema:*

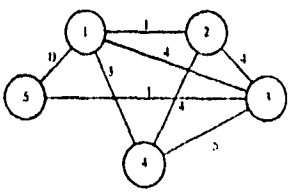


Fig. 3.11

*Para construir el 1-árbol de peso mínimo se toma en cuenta que la arista* $(1,4)$ *está fija en uno:*

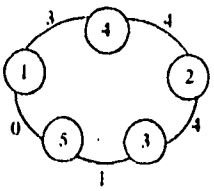


Fig. 3.12

*El valor óptimo es 12. Esta solución corresponde a un "tour". El valor óptimo de este "tour" x con respecto a los costos originales es* $\sum_{(i,j)\in x}(c_{ij} + \lambda_i^2 + \lambda_j^2) - 2\sum_{i=1}^{5}\lambda_i^3 = 12$.

*El algoritmo termina en esta etapa puesto que ya no faltan nodos por analizar. El "tour" óptimo es* $1-2-4-3-5-1$ *y el valor óptimo es 11. El árbol de ramificación y acotamiento se muestra en la figura 1.13:*

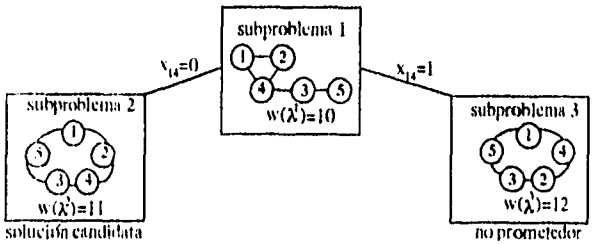



Fig. 3.13

## 3.2 Algoritmos Heurísticos.

### 3.2.1 Algoritmo Glotón.

Cualquier algoritmo glotón para el problema del agente viajero realiza los siguientes pasos:

1. Encuentra un *subtour* inicial que contiene 2 o más vértices.
2. Con alguna regla de selección elige un vértice para agregarlo al *subtour*.
3. Con algún criterio de inserción agrega el vértice seleccionado en el *subtour*.

Los pasos 2 y 3 se repiten hasta que todos los vértices se hayan insertado; es decir, hasta que se haya encontrado una solución para el problema del agente viajero.

A continuación se describirá un algoritmo que se basa en la inserción más barata. El *subtour* inicial $T$, en este caso, está formado por 2 ciudades $i_1$ e $i_2$ donde

$$d_{i_1 i_2} + d_{i_2 i_1} = \min_{i,j \in V, i \neq j} (d_{ij} + d_{ji})$$

Es decir, se elige el *subtour* de menor longitud que esté formado por 2 ciudades.

La regla para seleccionar un vértice y el criterio de inserción se combinan en este algoritmo de la siguiente manera. Primero se encuentra el costo más barato $c_k$ en el que se incurre al insertar cada vértice $k$ en el *subtour*. Obsérvese que si el vértice $k$ se inserta entre los vértices $i$ y $j$ del *subtour* $T$, el costo del nuevo *subtour* va a aumentar en $d_{ik} + d_{kj} - d_{ij}$. Entonces $c_k$ se calcula como

$$c_k = \min_{(i,j) \in T} \{d_{ik} + d_{kj} - d_{ij}\} \quad \forall k \in V$$

Se elige el vértice $k^*$ para insertarlo, tal que

$$c_{k^*} = \min_{k \in V} \{c_k\}$$

Es decir, se elige el vértice que tenga asociado el menor costo de inserción $c_k$ y se inserta entre los vértices correspondientes al cálculo de $c_{k^*}$.

**Ejemplo 3.2.1** *Considérese el problema simétrico del ejemplo 3.2.2.*
PASO 1. *Cada entrada de la siguiente matriz indica las distancias $d_{ij}+d_{ji}$ del "subtour" formado por los vértices i y j. Dado que esta matriz es simétrica sólo se muestra la parte triangular superior.*

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | $M$ | 2 | 4 | 4 | 6 |
| 2 | | $M$ | 4 | 6 | $M$ |
| 3 | | | $M$ | 4 | 4 |
| 4 | | | | $M$ | $M$ |
| 5 | | | | | $M$ |

*El "subtour" de menor costo es $1-2-1$, entonces éste se elige como "subtour" inicial.*
ITERACIÓN 1. *Se calcula el costo de inserción para los vértices que no están en el "subtour", esto es*

$$c_k = \min\{d_{1k} + d_{k2} - d_{12}, d_{2k} + d_{k1} - d_{21}\} \quad k = 3,4,5$$

*Sin embargo, $d_{ij} = d_{ji} \quad \forall i,j \in V = \{1,2,3,4,5\}$ puesto que se trata de un problema simétrico. Entonces*

$$c_k = d_{1k} + d_{k2} - d_{12} \quad k = 3,4,5$$

*La siguiente tabla muestra estos costos.*

| $k$ | $d_{1k} + d_{k2} - d_{12}$ | = | $c_k$ | |
|---|---|---|---|---|
| 3 | $2+2-1$ | = | 3 | |
| 4 | $2+3-1$ | = | 4 | |
| 5 | $3+M-1$ | = | $N$ | *donde* $N >> 0$ |

*Se elige el vértice 3 para insertarlo en el "subtour" puesto tiene asociado el menor costo $c_k$. El nuevo "subtour" es $1-3-2-1$.*
ITERACIÓN 2. *Se calculan los costos de inserción para los vértices 4 y 5, donde*

$$c_k = \min\{d_{1k} + d_{k3} - d_{13}, d_{3k} + d_{k2} - d_{32}, d_{2k} + d_{k1} - d_{21}\} \quad k = 4,5$$

*Estos se muestran a continuación.*

| $k$ | $d_{1k} + d_{k3} - d_{13}$ | $d_{3k} + d_{k2} - d_{32}$ | $d_{2k} + d_{k1} - d_{21}$ | $c_k$ |
|---|---|---|---|---|
| 4 | $2+2-2=2$ | $2+3-2=3$ | $3+2-1=4$ | 2 |
| 5 | $3+2-2=3$ | $2+M-2=M >> 0$ | $M+3-1=N >> 0$ | 3 |

*El vértice 4 tiene el menor costo de inserción. El nuevo "subtour" es* $1-4-3-2-1$.

ITERACIÓN 3. *El único vértice que falta por insertar es el 5. Para saber dónde conviene insertarlo se calcula* $c_5$:

$$\begin{aligned} c_5 &= \min\{d_{15}+d_{54}-d_{14}, d_{45}+d_{53}-d_{43}, d_{35}+d_{52}-d_{32}, d_{25}+d_{51}-d_{21}\} \\ \Rightarrow c_5 &= \min\{3+M-2, M+2-2, 2+M-2, M+3-1\} \end{aligned}$$

*Obsérvese que al insertar el vértice 5 en cualquier parte del "subtour" se incurre en un costo muy grande. Entonces, arbitrariamente se inserta entre los vértices 3 y 2. El "tour" que se obtiene es* $1-4-3-5-2-1$. *El costo de esta solución es* $M$, *donde* $M >> 0$.

Obsérvese que el valor de la solución que se obtiene está bastante alejado del valor óptimo $Z^* = 11$. En general los algoritmos glotones son muy sencillos y la solución que se obtiene no es muy buena. Existen algoritmos mejores que estos los cuales no solamente generan una solución factible sino que también toman en cuenta otros factores, como la estructura del problema, para obtener una solución más cercana a la óptima. Algunos algoritmos de este tipo requieren de una cota superior del valor óptimo la cual se puede obtener mediante algún algoritmo glotón.

### 3.2.2 Mejoras Sucesivas.

Recuérdese que en un algoritmo de mejoras sucesivas se busca un óptimo local. Es decir, se busca una solución factible $x$ que sea mejor que cualquier otra solución en la vecindad $N(x,\sigma)$. En el problema del agente viajero la vecindad $N(x,\sigma)$ de un *tour* $x$ es un conjunto de soluciones que se generan a partir de $x$ al realizar una serie de intercambios de aristas en la que se eliminan $k (k \geq 2)$ aristas de $x$ y se agregan otras $k$ aristas diferentes a las anteriores para formar un nuevo *tour*. El algoritmo de mejoras sucesivas más simple es aquél en el que se intercambian 2 aristas.

Este algoritmo funciona mejor para el problema simétrico del agente viajero, (*Sulkin, 1989*), por lo que sólo se considerará este caso.

**Algoritmo.**

1. Se encuenta un *tour* inicial $x^1$ y se inicializa el contador en 1: $k \leftarrow 1$. Este *tour* se puede obtener mediante un algoritmo glotón.

2. Se evalúan todos los posibles intercambios de 2 aristas como se muestra en la figura 3.14.

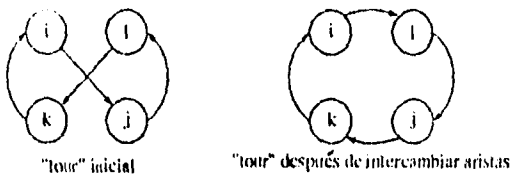



Fig. 3.14

La evaluación del intercambio de aristas de la figura anterior es $(d_{ij} + d_{lk}) - (d_{il} + d_{jk})$, lo cual representa el ahorro en la longitud del *tour* $x^k$ que se obtiene al realizar el intercambio. Si el intercambio que produce el mayor ahorro reduce la longitud total del *tour* $x^k$ entonces se realiza el intercambio correspondiente para así generar otro *tour* $x^{k+1}$ y se incrementa el contador: $k \leftarrow k + 1$. Se vuelve a aplicar este paso. De lo contrario el algoritmo termina aquí, puesto que ya no se puede mejorar la solución $x^k$ en la vecindad $N(x^k, \sigma)$ y por lo tanto constituye un óptimo local.

**Ejemplo 3.2.2** *Considérese el ejemplo 2.2.2. En la sección anterior se obtuvo, mediante un algoritmo glotón, la siguiente solución factible para este problema:* $1-4-3-5-2-1$, *cuyo costo es de* $M$ *unidades.*
ITERACIÓN 1. *La siguiente tabla muestra la diferencia de costos al intercambiar 2 aristas en el "tour".*

| $i$ | $j$ | $l$ | $k$ | $(d_{ij} + d_{lk}) - (d_{il} + d_{jk})$ | = | *ahorro* |
|---|---|---|---|---|---|---|
| 1 | 4 | 3 | 5 | $(2+2) - (2+M)$ | = | $-N, N >> 0$ |
| 1 | 4 | 5 | 2 | $(2+M) - (3+3)$ | = | $N$ |
| 4 | 3 | 5 | 2 | $(2+M) - (M+2)$ | = | 0 |
| 4 | 3 | 2 | 1 | $(2+1) - (3+2)$ | = | $-2$ |
| 3 | 5 | 2 | 1 | $(2+1) - (2+3)$ | = | $-2$ |

*El mayor ahorro es* $N$, *donde* $N >> 0$ *y se obtiene al intercambiar las aristas* $(1,4)$ *y* $(5,2)$ *por* $(1,5)$ *y* $(4,2)$. *El nuevo "tour" es* $1-5-3-4-2-1$, *el cual tiene un valor de* 11.

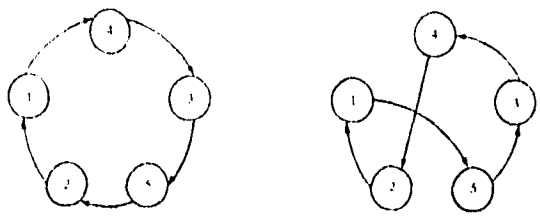


Fig. 3.15 Intercambio de aristas para formar otro "tour".

ITERACIÓN 2. *A continuación se muestra la diferencia de costos al intercambiar 2 aristas del nuevo "tour".*

| $i$ | $j$ | $l$ | $k$ | $(d_{ij}+d_{lk})-(d_{il}+d_{jk})$ | = | *ahorro* |
|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 3 | $(1+2)-(2+2)$ | = | $-1$ |
| 1 | 2 | 3 | 5 | $(1+2)-(2+M)$ | = | $-N, N>>0$ |
| 2 | 4 | 3 | 5 | $(3+2)-(2+M)$ | = | $-N$ |
| 2 | 4 | 5 | 1 | $(3+3)-(M+2)$ | = | $-N$ |
| 4 | 3 | 5 | 1 | $(2+3)-(M+2)$ | = | $-N$ |

*Obsérvese que no se obtiene ningún ahorro al intercambiar 2 aristas. Por lo tanto el algoritmo termina en esta etapa. Para este ejemplo la solución que se obtiene al aplicar el algoritmo,* $1-5-3-4-2-1$, *es la solución óptima del problema.*

Algunos autores [8] han sugerido evaluar 3 o más intercambios de aristas para obtener mejores soluciones. Sin embargo, esto requiere de un mayor trabajo computacional.

### 3.2.3 Recocido Simulado.

En este capítulo se han presentado 2 algoritmos heurísticos para resolver el problema del agente viajero: uno glotón y otro de mejoras locales. Sin embargo, se mostró que el algoritmo glotón generalmente no genera soluciones cercanas a la óptima puesto que es un algoritmo muy simple. Por otro lado, se mostrará que el algoritmo de mejoras locales tampoco genera soluciones satisfactorias puesto que éstas dependen de la solución inicial. Entonces, es necesario considerar otros algoritmos heurísticos más sofisticados para poder

obtener mejores soluciones. La heurística que se presenta a continuación se conoce como recocido simulado. Esta se empezó a usar como una técnica de optimización discreta a principios de los 80's y además ha tenido mucha aceptación porque es fácil de implementar y tiene muchas aplicaciones.

El recocido simulado se puede ver como una variante de la técnica heurística conocida como búsqueda local, en la cual se examina un subconjunto de soluciones factibles al moverse de una solución a otra vecina. La búsqueda se realiza de tal manera que se pase de una solución a otra mejor. Sin embargo, con esta estrategia generalmente se converge a un óptimo local en lugal de a uno global. En el capítulo 2 se mencionaron varias modificaciones que se le pueden hacer a este algoritmo con el fin de evitar este problema. Una de ellas consiste en repetir el proceso utilizando diferentes soluciones iniciales. Otra modificación es incrementar la complejidad de las soluciones vecinas. Pero ninguna de estas modificaciones ha sido completamente satisfactoria, puesto que la solución que se obtiene depende totalmente de la solución inicial. En la siguiente figura se muestra un ejemplo en el cual cada solución tiene 2 vecindades que corresponden a los puntos que están a la izquierda y a la derecha. En una estrategia de este tipo, el movimiento siempre será hacia abajo del valle donde se encuentra el punto inicial. Por ejemplo, el punto inicial $P$ siempre conducirá a la solución $Q$. Las únicas excepciones son los máximos locales ( por ejemplo el punto $R$ ) que puede condicir a 2 soluciones ( $Q$ y $S$ ).

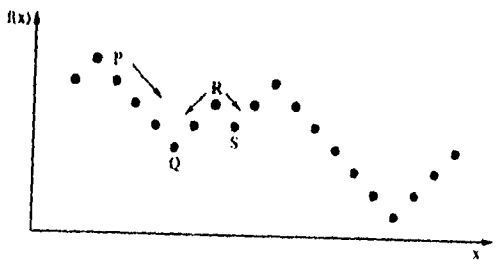



Fig. 3.16

De acuerdo a lo anterior, una buena heurística es aquélla que no depende del punto inicial. Entonces, la heurística debe permitir movimientos hacia soluciones peores de una manera controlada. En el recocido simulado se permiten este tipo de movimientos donde su frecuencia está controlada con una función de probabilidad. Por lo tanto, la frecuencia cambia a medida que avanza el algoritmo.

Fenómeno físico sobre el cual se basa el algoritmo.

Las ideas sobre las cuales se basa el recocido simulado fueron publicadas por primera vez por Metrópolis en 1953 donde se presenta un algoritmo que simula el enfriamiento de un material que ha sido calentado previamente. A este proceso se le conoce como recocido. Si un material sólido se calienta más allá de su punto de fusión y después se enfría hasta que alcance su estado sólido, las propiedades de éste pueden depender de la tasa de enfriamiento. Por ejemplo, los cristales grandes pueden crecer al enfriar el material lentamente; pero si se enfrian rápidamente, éstos pueden tener imperfecciones. El proceso de recocido puede ser simulado al considerar el material como un sistema de partículas. Entonces, el algoritmo de Metrópolis simula el cambio de energía del sistema cuando está sujeto a un proceso de enfriamiento, hasta que converge a un estado de congelamiento estable.

Las leyes de la termodinámica establecen que a una determinada temperatura $t$, la probabilidad de un incremento de energía de magnitud $\delta E$ está dado por:

$$\mathbb{P}(\delta E) = \exp(\frac{-\delta E}{kt}) \tag{3.3}$$

donde $k$ es una constante física conocida como constante de Boltzmann. La simulación de Metrópolis genera una perturbación y calcula el cambio de energía producido. Si la energía disminuye, el sistema se mueve a este nuevo estado. Si la energía se incrementa, se acepta el nuevo estado de acuerdo a la probabilidad dada por (3.3). El proceso se repite un número predeterminado de iteraciones para cada temperatura, después del cual la temperatura se disminuye hasta que el sistema alcanza un estado de congelación estable.

Treinta años más tarde, Kirkpatrick sugirió que este tipo de simulación se puede usar para buscar soluciones factibles de un problema de optimización que convergan a una solución óptima. Esto se logra al realizar un mapeo del los elementos del proceso físico de enfriamiento con los elementos del problema de optimización combinatoria como se muestra a continuación:

| SIMULACION TERMODINAMICA | OPTIMIZACION COMBINATORIA |
|---|---|
| Estados del sistema | Soluciones factibles |
| Energía | Costo |
| Cambio del estado | Soluciones vecinas |
| temperatura | Control del parámetro |
| Estado de congelación | Solución heurística |

Entonces, cualquier algoritmo de optimización local se puede transformar a un algoritmo de recocido simulado al :

- tomar una solución vecina aleatoriamente y
- permitir una solución inferior (es decir, una solución peor que la actual) con probabilidad $\mathbb{P}(\delta E)$.

La decisión de permitir una solución peor depende del incremento en la función de costos (incremento en la energía) y del tiempo de búsqueda en una etapa determinada (temperatura actual).

**Algoritmo.**

El recocido simulado no se considera como un solo algoritmo, sino como una familia de algoritmos o una estrategia heurística. Entonces, en la descripción general de éste se deben tomar varias decisiones para implementar el método a un problema particular. Los pasos generales de este método son los siguientes:

Seleccionar una solución inicial $s_0$.
Seleccionar una temperatura inicial $t_0 > 0$.
Seleccionar una función $\alpha$ para reducir la temperatura.
Repetir
  Repetir
    Seleccionar aleatoriamente $s \in N(s_0, \sigma)$
    $\delta = f(s) - f(s_0)$
    Si $\delta < 0$
      entonces $s_0 = s$
    sino
      generar una *v.a.* uniforme $x$ en el intervalo $(0, 1)$
      si $x < \exp(\frac{-\delta}{t})$ entonces $s_0 = s$
  Hasta que el *contador de iteraciones* = *número de repeticiones*
  Hacer $t = \alpha(t)$
Hasta que la condición para terminar el algoritmo sea verdadera. $s_0$ es la aproximación al óptimo.

Obsérvese que $t$ aparece simplemente como un parámetro de control y no tiene ninguna interpretación física, entonces la constante de Boltzmann ha sido eliminada de la función de probabilidad. Sin embargo, generalmente

a $t$ se le sigue llamando temperatura y a la tasa en que se reduce $t$, modo de enfriamiento. El algoritmo descrito previamente sigue la simulación original de Metróplos por lo que mantiene la analogía con el proceso físico de enfriamiento. Entonces, el uso de la distribución (3.3) se debe únicamente a las leyes de la termodinámica y no hay razón para suponer que alguna otra distribución no pueda funcionar mejor para ciertos problemas. Sin embargo, ésta es la función de distribución que generalmente se usa. Esto se debe a que ésta tiene las características que son intuitivamente correctas para el criterio de aceptación. Las soluciones se aceptan de acuerdo a la magnitud del incremento en el costo, para incrementos grandes la probabilidad de aceptación es pequeña mientras que los incrementos pequeños regularmente se aceptan.

**Implementación del algoritmo.**

El algoritmo presentado es muy general y por lo tanto se deben tomar algunas decisiones para implementarlo a un problema particular. Las decisiones se pueden dividir en 2 categorías:

1. Las que tienen que ver con el algoritmo (decisiones genéricas) como la temperatura inicial, el modo de enfriamiento (que está determinado por el número de repeticiones y la función $\alpha$ para reducir la temperatura) y la condición para terminar el algoritmo.

2. Las que tienen que ver con el problema como la elección del espacio de soluciones factibles, la función de costos y la vecindad.

Los 2 tipos de decisiones deben realizarse con cuidado porque afectan la rapidez del algoritmo y la calidad de las soluciones obtenidas.

**Decisiones genéricas.**

**1.Temperatura inicial.** Se desea que la solución final sea independiente de la solución inicial, entonces la temperatura inicial debe ser lo suficientemente alta para permitir un movimiento a una solución peor. En algunos casos hay suficiente información en el problema para estimar esto. Por ejemplo, si se conoce la máxima diferencia del costo entre soluciones vecinas, $t$ puede calcularse adecuadamente ya que se desea que $\exp(\frac{-\delta}{t})$ sea lo más grande posible para que $x < \exp(\frac{-\delta}{t})$ y así poder aceptar un

movimiento a una solución peor. Sin embargo, generalmente esta información es difícil de obtener. En estos casos, se decide de antemano la proporción de movimientos hacia soluciones peores que se van a aceptar y el sistema se calienta rápidamente (es decir se incrementa $t$) hasta que esta proporción alcance el valor deseado. En este punto se empieza a disminuir la temperatura. Obsérvese que este método corresponde a la analogía física en el cual una substancia se calienta rápidamente hasta alcanzar su estado líquido y después se enfría lentamente de acuerdo con el proceso de recocido.

**2. Modo de enfriamiento.** El modo de enfriamiento está determinado por el número de repeticiones y la función $\alpha$ para reducir la temperatura. Se sugiere que el número de iteraciones que se realizan para cada temperatura sea tal que se alcance un equilibrio, hasta que $\delta$ esté cercano a cero. Además la temperatura debe converger gradualmente a cero. Sin embargo, para lograr esto se requiere un número de iteraciones exponencial al tamaño del problema. Entonces, de debe reducir el número de iteraciones de alguna manera. Esto se logra al realizar un número grande de iteraciones con pocas temperaturas o al realizar pocas iteraciones con muchas temperaturas. A continuación se presenta uno de los métodos más usados.

En este método se usa una función geométrica para reducir la temperatura, $\alpha(t) = at$ donde $a < 1$. Generalmente los valores de $a$ que se usan son los que están entre 0.8 y 0.99. Por otro lado, el número de iteraciones para cada temperatura está relacionado con el tamaño de las vecindades y varía de temperatura a temperatura. Por ejemplo, es importante realizar más iteraciones a bajas temperaturas para asegurarse de que el óptimo local ha sido explorado completamente. Entonces, es conveniente incrementar el número de repeticiones geométricamente (multiplicandolo por un factor mayor que 1) o aritméticamente (sumandole una constante) en cada temperatura.

Otra manera de determinar el número de repeticiones es la siguiente. Para cada temperatura se debe fijar el número de movimientos hacia soluciones peores que se van a aceptar. Esto implica un periodo corto de tiempo para temperaturas altas. En cambio, para temperaturas bajas el número de movimientos hacia soluciones peores que se aceptan es pequeño y por lo tanto se requiere un número muy grande de iteraciones. Entonces, es necesario imponer un número máximo de restricciones para cada temperatura.

**3. Condición para terminar el algoritmo.** En teoría, la temperatura debe llegar a cero antes de que la condición para terminar el algoritmo se

satisfaga. Sin embargo, no hay necesidad de disminuir tanto la temperatura. A medida que $t$ se aproxima a cero, la probabilidad de aceptar un movimiento a una solución peor se vuelve muy pequeña. Entonces, antes de llegar a una temperatura de cero, la probabilidad de salirse del actual óptimo local es muy pequeña. Por lo tanto, el criterio para detenerse puede expresarse en términos un valor mínimo del parámetro de temperatura. Otra regla consiste en especificar el número de temperaturas que deben pasar sin que se produzca un movimiento a una solución peor.

En general, no es fácil especificar los valores de los parámetros. En muchas de las aplicaciones de este algoritmo que han dado buenos resultados, los parámetros se determinan después de realizar muchas pruebas.

**Decisiones específicas del problema.** Una vez que se ha elegido el problema para el cual se desea encontrar una solución factible mediante el recocido simulado, se debe decidir qué vecindad se va a utilizar. En esta sección únicamente se discutirá la vecindad para el problema del agente viajero.

La vecindad que generalmente se usa es la que se definió en la sección anterior. Sea $x_0$ un tour, entonces la vecindad $N(x_0, \sigma)$ está formada por todas las soluciones que se generan a partir se $x_0$ al intercambiar 2 aristas. Si se eliminan las aristas $(i,j)$ y $(l,k)$ la única manea para formar un nuevo *tour* es conectar $i$ con $l$ y $j$ con $k$. Entonces, una solución vecina se puede obtener al generar 2 variables aleatorias uniformes $i$ y $l$ donde $1 \leq i, j \leq n$ ($n$ es el número de ciudades). De esta manera sólo hay que calcular el sucesor de $i$ y de $l$ en la solución $x_0$ y realizar el intercambio de aristas, como se describe arriba, para generar una solución vecina.

La función de costos, es la longitud del *tour*. Sin embargo, para calcular el incremento en el costo $\delta$, no es necesario calcular la longitud del nuevo *tour*. En el capítulo 3 se hizo notar que $\delta$ se puede determinar de la siguiente manera:

$$\delta = (d_{il} + d_{jk}) - (d_{ij} + d_{jk})$$

**Ejemplo 3.2.3** *Considérese el problema del agente viajero definido en el ejemplo 2.2.2. La matriz de costos de este problema es :*

| $i \setminus j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | $M$ | 1 | 2 | 2 | 3 |
| 2 | 1 | $M$ | 2 | 3 | $M$ |
| 3 | 2 | 2 | $M$ | 2 | 2 |
| 4 | 2 | 3 | 2 | $M$ | $M$ |
| 5 | 3 | $M$ | 2 | $M$ | $M$ |

*Para aplicar el recocido simulado a este ejemplo se deben tomar varias decisiones.*

*1. Solución inicial $s_0$. La solución inicial que se usará es $s_0 = 1-4-3-5-2-1$, la cual se obtuvo en la sección 3.2.1 al aplicar un algoritmo glotón. El costo de esta solución es 107, si se fija $M = 100$.*

*2. Vecindad. Se usará la vecindad $N(s_0, \sigma)$ definida como el conjunto de soluciones que se generan a partir de $s_0$ al intercambiar dos aristas.*

*3. Temperatuta inicial $t_0$. En este caso se pueden determinar todas las soluciones vecinas de $s_0$ y la diferencia de costos con respecto a la solución inicial.*

| | $(i,j)$ | $(l,k)$ | $(i,l)$ | $(j,k)$ | $(d_{il}+d_{jk})-(d_{ij}+d_{lk})$ | $=\delta$ |
|---|---|---|---|---|---|---|
| $s_1$ | (1,4) | (3,5) | (1,3) | (4,5) | $(2+100)-(2+2)$ | $=98$ |
| $s_2$ | (1,4) | (5,2) | (1,5) | (4,2) | $(3+3)-(2+100)$ | $=-96$ |
| $s_3$ | (4,3) | (5,2) | (4,5) | (3,2) | $(100+2)-(2+100)$ | $=0$ |
| $s_4$ | (4,3) | (2,1) | (4,2) | (3,1) | $(3+2)-(2+1)$ | $=2$ |
| $s_5$ | (3,5) | (2,1) | (3,2) | (5,1) | $(2+3)-(2+1)$ | $=2$ |

*Las soluciones vecinas de $s_0$ que tienen un costo más grande que la solución inicial son $s_1$, $s_3$, $s_4$ y $s_5$. Se desea elegir la temperatura inicial de tal manera que se acepten las soluciones $s_3$, $s_4$, y $s_5$. Para ello se fija un valor de $\exp(\frac{-\delta}{t})$ en un número cercano a uno, para $\delta = 2$, con el fin de que sea más probable aceptar estas soluciones. Si $\exp(\frac{-2}{t}) = 0.9$ entonces $t = 18.98$. Por otro lado, $\exp(\frac{-2}{18.98}) = 0.0057$. Esto significa que es muy poco probable que se acepte la solución $s_1$.*

*4. Modo de enfriamiento. La función $\alpha$ para reducir la temperatura se determina como $\alpha = at$ con $a = 0.5$. Por lo tanto la temperatura se reduce a la mitad. El número de repeticiones para cada temperatura es de 3 y la condición para terminar el algoritmo es cuando $t < 2.5$.*

*A continuación se muestran las iteraciones que se hicieron para resolver*

*este problema. Los números $x_1$ y $x_2$ son variables aleatorias uniformes que pueden valer 1, 2, 3, 4 o 5. Es decir, están asociadas con las ciudades del problema y sirven para generar una solución aleatoria. El número $x$ es una variable aleatoria uniforme que está en el intervalo $(0, 1)$.*

ITERACIÓN 1.

*1.* $x_1 = 5, x_2 = 4 \Rightarrow s = 1 - 4 - 5 - 3 - 2 - 1$

*2.* $\delta = (d_{54} + d_{23}) - (d_{52} + d_{43}) = 0$

*3.* $x = 0.803$

*4.* $\exp(\frac{0}{18.98}) = 1$

*5. Como* $x < \exp(\frac{0}{18.98}) \Rightarrow s_0 = 1 - 4 - 2 - 3 - 5 - 1$

*6.* $numit = 1$

ITERACIÓN 2.

*1.* $x_1 = 2, x_2 = 4 \Rightarrow s = 1 - 4 - 2 - 3 - 5 - 1$

*2.* $\delta = (d_{21} + d_{15}) - (d_{21} + d_{45}) = -95$

*3. Como* $\delta < 0 \Rightarrow s_0 = 1 - 4 - 2 - 3 - 5 - 1$

*4.* $numit = 2$

ITERACIÓN 3.

*1.* $x_1 = 4, x_2 = 3 \Rightarrow s = 1 - 4 - 3 - 2 - 5 - 1$

*2.* $\delta = (d_{43} + d_{25}) - (d_{42} + d_{35}) = 97$

*3.* $x = 0.907$

*4.* $\exp(\frac{-97}{18.98}) = 0.006$

*5. Como* $x > \exp(\frac{-97}{18.98})$ *no se acepta* $s$.

*6.* $numit = 3$

ITERACIÓN 4.

*Como* $numit = 3 \Rightarrow t = 0.5(18.98) = 9.49$ *y se inicializa la variable numit en cero.*

*1.* $x_1 = 3, x_2 = 1 \Rightarrow s = 1 - 3 - 2 - 4 - 5 - 1$

*2.* $\delta = (d_{31} + d_{54}) - (d_{35} + d_{14}) = 98$

*3.* $x = 0.920$

*4.* $\exp(\frac{-98}{9.49}) = 0.000033$

*5. Como* $x > \exp(\frac{-98}{9.49})$ *entonces no se acepta* $s$.

*6.* $numit = 1$

ITERACIÓN 5.

*1.* $x_1 = 2, x_2 = 1 \Rightarrow s = 1 - 2 - 4 - 3 - 5 - 1$

*2.* $\delta = (d_{21} + d_{34}) - (d_{23} + d_{14}) = -1$

*3. Como* $\delta < 0$ *entonces* $s_0 = 1 - 2 - 4 - 3 - 5 - 1$

*4.* $numit = 2$

ITERACIÓN 6.

*1.* $x_1 = 3, x_2 = 1 \Rightarrow s = 1 - 3 - 4 - 2 - 5 - 1$

2. $\delta = (d_{31} + d_{52}) - (d_{35} + d_{12}) = 99$
3. $x = 0.113$
4. $\exp(\frac{-99}{9.49}) = 0.000029$
5. *Como* $x > \exp(\frac{-99}{9.49})$ *no se acepta s*
6. $numit = 3$

ITERACIÓN 7.

*Como* $numit = 3 \Rightarrow t = 0.5(9.49) = 4.745$ *y se inicializa la variable* $numit$ *en cero.*

1. $x_1 = 4, x_2 = 5 \Rightarrow s = 1 - 2 - 4 - 5 - 3 - 1$
2. $\delta = (d_{45} + d_{31}) - (d_{43} + d_{51}) = 97$
3. $x = 0.897$
4. $\exp(\frac{-97}{4.745}) = 0.1 \times 10^{-8}$
5. *Como* $x > \exp(\frac{-97}{4.745})$ *entonces no se acepta s.*
6. $numit = 1$

ITERACIÓN 8.

1. $x_1 = 3, x_2 = 2 \Rightarrow s = 1 - 2 - 3 - 4 - 5 - 1$
2. $\delta = (d_{32} + d_{54}) - (d_{35} + d_{24}) = 97$
3. $x = 0.968$
4. $\exp(\frac{-97}{4.745}) = 0.1 \times 10^{-8}$
5. *Como* $x > \exp(\frac{-97}{4.745})$ *no se acepta s*
6. $numit = 2$

ITERACIÓN 9.

1. $x_1 = 1, x_2 = 3 \Rightarrow s = 1 - 3 - 4 - 2 - 5 - 1$
2. $\delta = (d_{13} + d_{25}) - (d_{12} + d_{35}) = 99$
3. $x = 0.315$
4. $\exp(\frac{-99}{4.745}) = 8.7 \times 10^{-10}$
5. *Como* $x > \exp(\frac{-99}{4.745})$ *no se acepta s*
6. $numit = 3$

ITERACIÓN 10.

*Como* $numit = 3 \Rightarrow t = 0.5(4.745) = 2.3725$. *Dado que* $t < 2.5$ *en este momento se termina el algoritmo. La solución generada con este algoritmo es* $s = 1 - 2 - 4 - 3 - 5 - 1$ *con costo* $Z = 11$. *Obsérvese que esta solución coincide con la solución óptima del problema original.*

# Capítulo 4

# Paquete de Cómputo.

Se realizó un paquete de cómputo para resolver el problema del agente viajero con el algoritmo de ramificación y acotamiento que utiliza como relajación el problema de asignación. Además, este paquete obtiene cotas inferiores del valor óptimo mediante el método de relajación Lagrangeana descrito en el capítulo 3. En este capítulo se describe cómo se implementaron los principales procedimientos de cada algoritmo y qué estructuras de datos se usaron. Además se discute qué tan eficiente es la implementación que se usó, ya que el tiempo en el que se obtenga la solución no depende únicamente del algoritmo.

## 4.1 Algoritmo de Ramificación y Acotamiento.

### 4.1.1 Introducción.

Al desarrollar un algoritmo en la computadora para un problema que es $NP - completo$, como el problema del agente viajero, es importante implementarlo de tal manera que sea lo más eficiente posible. Es decir, se está interesado en que el tiempo de ejecución del algoritmo sea lo más corto posible. En general, las implementaciones estáticas son más eficientes que las dinámicas. Por la razón anterior la matriz de costos del problema se implementó con una estructura estática: una matriz de incidencia $D$ de dimensión $k \times k$, donde $d_{ij}$ representa la distancia que hay del la ciudad $i$ a la ciudad $j$. El valor de $k$ es fijo y es de 8. Es decir, este paquete sólo resuelve problemas donde el número de ciudades sea menor o igual que 8. Otra razón

por la que se usó una implementación estática es que la matriz de costos del problema no se modifica en el transcurso del algoritmo.

### 4.1.2 Método Húngaro.

En esta sección se describe cómo se programó el método Húngaro, el cual se utiliza para resolver el problema de asignación. A continuación se presenta un diagrama para ilustrar el algoritmo y los principales procedimientos que se utilizan.

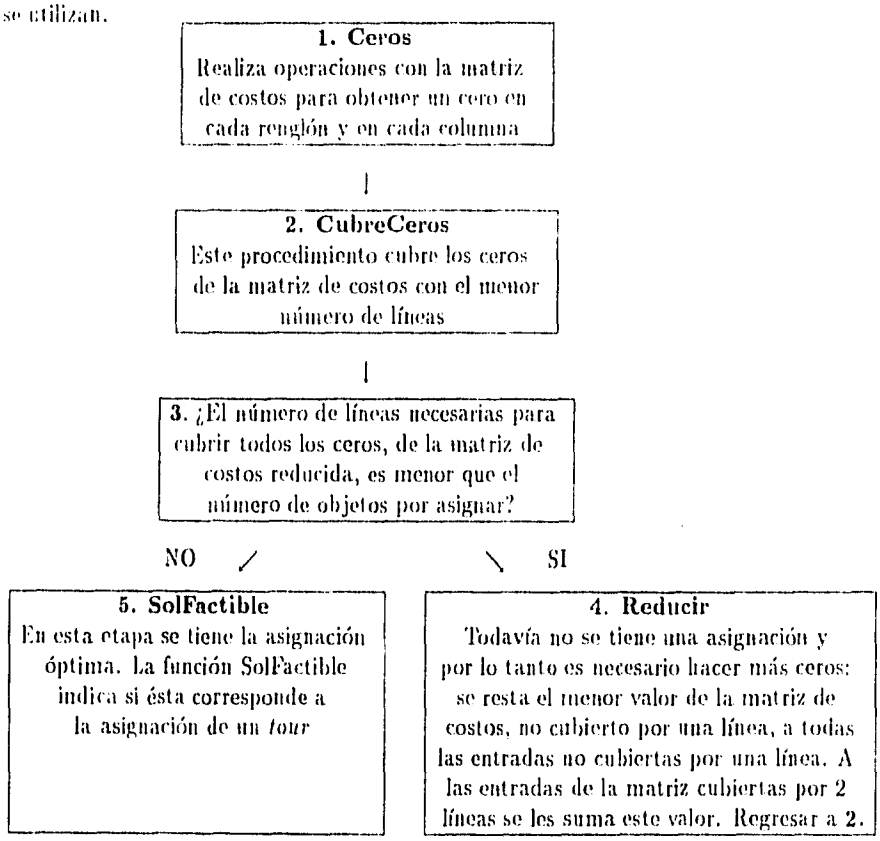



Fig. 4.1

A continuación se describen los algoritmos utilizados en el procedimiento **CubreCeros** y **SolFactible**.

1. CubreCeros. Antes de describir el algoritmo se definirán algunos conceptos.

**Definición 4.1.1** *Un conjunto de entradas de una matriz de costos es independiente si éstas se encuentran en renglones y columnas diferentes.*

Cuando se resuelve el problema de asignación se desea encontrar $n$ entradas independientes, de la matriz de costos, cuya suma sea mínima. Esto es equivalente a obtener $n$ ceros independientes en una matriz de costos reducida.

**Definición 4.1.2** *Un conjunto independiente de ceros con el mayor número de elementos se le conoce como conjunto máximo independiente de ceros.*

Por ejemplo, en la siguiente matriz se muestra un conjunto independiente de ceros de 3 elementos (estas entradas están marcadas con un asterisco). Dado que no existe un conjunto independiente de ceros con 4 o 5 elementos, éste es el conjunto máximo independiente.

| | | | | |
|---|---|---|---|---|
| 3 | 9 | 0* | 8 | 9 |
| 3 | 0* | 7 | 1 | 7 |
| 0* | 2 | 1 | 0 | 0 |
| 4 | 7 | 0 | 11 | 6 |
| 4 | 0 | 2 | 2 | 2 |

Fig. 4.2

En esta matriz es fácil verificar que no existe un conjunto independiente de ceros con más de 3 elementos. Sin embargo, en una matriz más grande es difícil determinar el conjunto máximo. Más adelante se dará un criterio para encontrar el conjunto máximo independiente de ceros en una matriz de costos reducida.

**Definición 4.1.3** *Una cubierta para un conjunto de ceros en una matriz de costos es un conjunto de líneas verticales y horizontales que cubren todos los ceros de la matriz.*

En la matriz de la figura 4.2 se necesitan 3 líneas para cubrir todos los ceros. Obsérvese que el número de líneas en una cubierta no puede ser menor que la cardinalidad del conjunto independiente ya que cada línea puede cubrir a lo más un cero del conjunto independiente. En este ejemplo, la cardinalidad del conjunto máximo independiente es igual al número de líneas de la cubierta más pequeña. Esto sucede en general y esta relación se enuncia en el teorema de König.

**Teorema 4.1.1** *Para cada matriz que contenga ceros en algunas entradas, la cardinalidad del conjunto máximo independiente de ceros es igual al número de líneas de la mínima cubierta de ceros.*

A continuación se describe un algoritmo para encontrar el máximo conjunto independiente de ceros y la mínima cubierta en una matriz de costos reducida. La matriz de costos para el problema de asignación es cuadrada, sin embargo este algoritmo se puede aplicar a cualquier matriz. Considérese la siguiente matriz, en la cual se han eliminado las entradas diferentes de cero.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | | 0 | | 0* | | | | | |
| 2 | 0* | | | | | 0 | | 0 | |
| 3 | | | 0* | | 0 | | 0 | | |
| 4 | | 0* | | | 0 | | | | |
| 5 | 0 | | | 0 | | | | | 0* |
| 6 | | | 0 | | 0* | | | | |
| 7 | | | | | 0 | | 0* | | |
| 8 | | | | | | | 0 | | |

Fig. 4.3

El primer paso del algoritmo consiste en encontrar un conjunto independiente. En la matriz de la figura 4.3, este conjunto está formado por los ceros que tienen asterisco. Después se dibuja una línea horizontal que cubra cada cero marcado. Dado que estos ceros forman un conjunto independiente, cada línea cubre exactamente uno de ellos. Los renglones no cubiertos se etiquetan con cero (véase la figura 4.4).

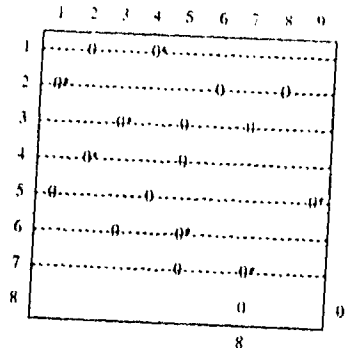


Fig. 4.4

Empezando por arriba, se localiza una entrada que tenga un cero no cubierto. En la figura enterior, esta entrada está en el renglón 8. Se etiqueta la columna donde se encuentra este cero con el índice del renglón; 8 en este caso. Después, se localiza un cero con asterisco en la columna etiquetada. Si existe, se elimina la línea horizontal que lo cubre y se dibuja una vertical para cubrirlo. Se etiqueta el renglón descubierto con el índice de la columna que contiene al cero con asterisco; 7 en este caso. A continuación se muestra este paso.

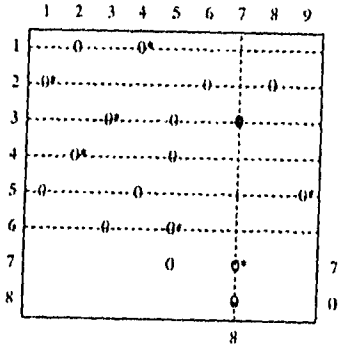


Fig. 4.5

El procedimiento descrito anteriormente se repite otra vez para la matriz de la figura 4.5. Es decir, se busca un cero no cubierto. En este caso caso, se encuentra en el renglón 7. Se etiqueta la columna, donde se está este cero, con 7 (el índice del renglón) y se localiza un cero con asterisco en esta columna. Este se encuentra en el renglón 6. Se elimina la línea horiziontal que cubre al cero con asterisco y se dibuja una vertical que lo cubra. El

renglón donde se encuentra este cero se etiqueta con el índice de la columna; 5 en este caso.

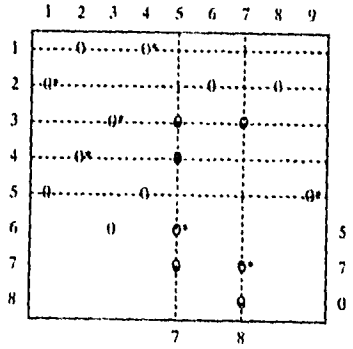



Fig. 4.6

Si se repite el procedimiento otra vez, se obtiene la siguiente cubierta.

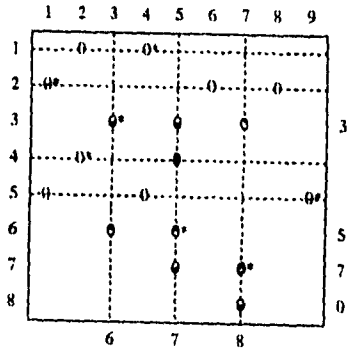



Fig. 4.7

El algoritmo termina en esta etapa porque todos los ceros están ubiertos por una línea. Obsérvese que se inició con una cubierta de cardinalidad igual al número de ceros con asterisco y además ésta se preservó en cada iteración. Por lo tanto al terminar el algoritmo, el número de ceros del conjunto máximo independiente es igual al número de líneas de la mínima cubierta de ceros. En este ejemplo se ilustró el algoritmo cuando se tiene un conjunto máximo independiente de ceros. Ahora, se mostgrará qué hacer cuando se empieza con un conjunto menor. Considérese la siguiente matriz:

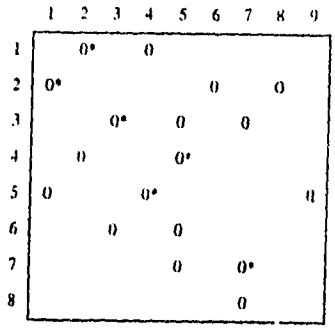



Fig. 4.8

El patrón de ceros es el mismo del ejemplo anterior. Por esta razón se sabe que hay un conjunto independiente de 7 ceros. Sin embargo, en la figura anterior sólo se tienen 6. Obsérvese que este conjunto de ceros no se puede expander a un conjunto independiente mayor. A este conjunto se le conoce como conjunto independiente *completo*. Si se aplica el algoritmo descrito anteriormente a la matriz de la figura 4.8, se obtiene la siguiente tabla

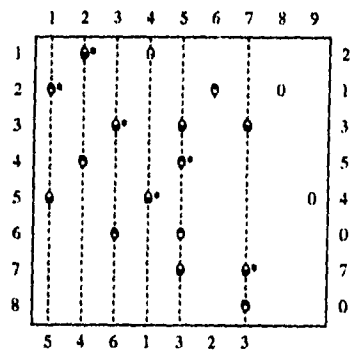



Fig. 4.9

Obsérvese que en esta matriz se etiquetó la columna 6 con el renglón que contiene un cero no cubierto, a saber el renglón 2. Sin embargo, en esta columna no hay ningún cero con asterisco. A este fenómeno se le conoce como *ruptura*. En este momento se puede determinar un conjunto independiente de cardinalidad mayor que el inicial de la siguiente manera. Las etiquetas determinan un camino que empieza con un cero no cubierto. La columna etiquetada con 2 indica que el segundo renglón contiene este cero. Este renglón está etiquetado con 1 y significa que la columna 1, sobre el

mismo renglón, contiene un cero con asterisco. La etiqueta 5 de esta columna indica que en el quinto renglón, sobre esta columna, se encuentra el siguiente cero, etc. Este camino se muestra en la siguiente figura.

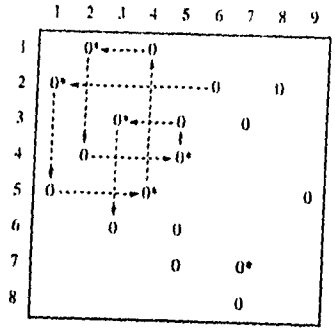



Fig. 4.10

En el camino que se obtiene, se alternan ceros con asterisco y sin asterisco. Si se borra el asterisco de aquéllos ceros con asterisco y se agrega un asterisco a los que no tienen, la cardinalidad del conjunto independiente se incrementa en uno. Obsérvese que el nuevo conjunto de ceros con asterisco es independiente y completo. Después de este paso se deben borrar las etiquetas y las líneas de la cubierta para repetir el proceso otra vez. En cada iteración se obtiene un conjunto independiente de ceros más grande o una cubierta que cubre todos los ceros de la matriz.

**Algoritmo de König.**

1. Se empieza con un conjunto completo e independiente de ceros. Este puede obtenerse de la siguiente manera. Se agrega un asterisco al primer cero del renglón 1. En el siguiente renglón se elige el primer cero que no se encuentre en la misma columna que otro cero con asterisco. El proceso se repite hasta que ya no se puedan meter más ceros en el conjunto independiente. Cada cero con asterisco se cubre con una línea horizontal. Todos los renglones no cubiertos se etiquetan con cero.

2. Se encuentra un renglón que contenga un cero no cubierto. Si todos los ceros están cubiertos, terminar. En este momento se tiene una cubierta donde el número de líneas es igual al número de ceros del conjunto independiente. Si existe un cero no cubierto, se etiqueta la columna con el índice del renglón donde se encuentra éste. Ir a 3.

3. Se encuentra un cero con asterisco en la columna etiquetada en el paso 2. Si no hay ningún cero con asterisco entonces se tiene una *ruptura*. Ir al paso 4. Si existe un cero con asterisco en la columna, éste está cubierto por una línea horizontal. Se etiqueta el renglón donde se encuentra el cero con el índice de esta columna. Se elimina la línea horizontal que lo cubre y se dibuja una vertical para cubrirlo. Regresar al paso 2.

4. La *ruptura* proporciona un camino donde se alternan los ceros con asterisco y los que no tienen asterisco. Sobre este camino se puede incrementar el número de ceros del conjunto independiente. El camino se construye al ir al renglón que indica la etiqueta en cada columna y al ir a la columna indicada por la etiqueta de los renglones. El camino se termina cuando se tenga una etiqueta con cero.

El camino empieza y termina con un cero sin asterisco. Es decir, el número de ceros sin asterisco es uno más que el número de ceros con asterisco. Se coloca un asterisco a los ceros que no tienen y se elimina el asterisco de aquéllos ceros que sí tienen. Esto genera un conjunto independiente de ceros con un elemento más. Se eliminan las etiquetas y las líneas de la cubierta. Regresar al paso 2.

**2. SolFactible**. Se define un vector $S$ de dimensión $n$ cuyas componentes son enteros. En este vector se guardará la asignación óptima de la siguiente manera: si la persona $i$ es asignada al trabajo $j$ entonces $S[i] = j$, donde $1 \leq i, j \leq n$. En el caso del problema del agente viajero significa que de la ciudad $i$ se viaja a la ciudad $j$.

Para saber si la asignación óptima es la asignación de un *tour* se realiza lo siguiente. En la variable *cadena* que es de tipo *string* se guardan las ciudades en el orden en el que se van visitando, hasta que se vuelve a repetir alguna ciudad. En este momento se cuentan las ciudades incluidas en la variable *cadena*. Si este número es menor que $n$ entonces se forman *subtours*, de lo contrario la asignación óptima es factible:

```
FUNCTION SolFactible : Boolean;
VAR
    NAux, i, j, k : integer;
    CAux, cadena : string;
begin
    i := 0; {núm. de cds. incluidas en la variable cadena}
    j := 1; {entrada del vector S}
    cadena :=' 1';
    Repeat
```

```
            NAux := S[j];
            Str(NAux, CAux);
            k := Pos(CAux, cadena);
            IF (k = 0) then cadena := concat(cadena, CAux);
            j := NAux;
            i := i + 1
        Until (k <> 0);
        SolFactible := (i = numV)
    end;
```

### 4.1.3 Arbol de Ramificación y Acotamiento y Algoritmo de Búsqueda.

Para construir el árbol de ramificación y acotamiento se utiliza una *unidad de árboles*, la cual realiza operaciones con árboles binarios. Un árbol binario ordenado se define como un conjunto finito de elementos (nodos) que es vacío o bien consiste de una raíz (nodo) con 2 árboles binarios disjuntos llamados subárbol izquierdo y derecho de la raíz.

Durante el desarrollo de los algoritmos de ramificación y acotamiento, el árbol crece contínuamente. Por esta razón, la representación del árbol con estructuras dinámicas, es decir elementos enlazados con apuntadores, es la más adecuada. Entonces, en el paquete el árbol se definió de la siguiente manera:

```
TYPE
    Elemento = String;
    Arbol = ^Nodo;
    Nodo = Record
                Subp : Elemento;
                Izq, Der : Arbol
           end;
```

Cada nodo del árbol guarda como información el número del subproblema correspondiente (*Arbol*^.*Subp*).

La estrategia de búsqueda que se utiliza en el paquete es la de la última cota. Es decir, en cada iteración del algoritmo se elige un subproblema no insondeable, del árbol de ramificación y acotamiento, para analizarlo. El subproblema que se elige es el último subproblema generado. Por esta razón se utiliza una *pila*. Una pila es una lista, o sucesión de elementos, en donde éstos se insertan o se eliminan sólo en un extremo de la lista. Entonces, cada vez que se genera un subproblema éste se mete a la pila. En cada iteración

del algoritmo se analiza el último subproblema incluido en la pila y después se elimina de la pila.

Durante la ejecución del programa, el número de subproblemas no insondeables crece y decrece constantemente. Por esta razón se utiliza una implementación dinámica para la pila. Es decir, se usan apuntadores para enlazar los elementos de la pila:

```
TYPE
    Elemento = String;
    Pila = ^Nodo;
    Nodo = Record
               Subp : Elemento;
               sig : Pila
           end;
```

Cada elemento de la pila contiene el número del subproblema correspondiente (*Pila^.Subp*).

Una vez que se ha analizado un subproblema de la pila, éste se busca en el árbol de ramificación y acotamiento ya sea para agregar 2 nodos al árbol (cuando es necesario ramificar) o para indicar que el subproblema es insondeable. La unidad de árboles contiene una función que encuentra un nodo específico del árbol. El algoritmo que usa esta función es recursivo y se basa en la siguiente idea. Busca el elemento o subproblema $x$ en la raíz del árbol. Si la raíz contiene este elemento, ya se encontró el nodo buscado (es la raíz). Obsérvese que este nodo se puede considerar como un árbol, de acuerdo a la definición de árbol binario. Por esta razón la función *EncuentrA* devuelve un árbol. Si el subproblema no está en la raíz, busca en el subárbol izquierdo y en el subárbol derecho.

```
FUNCTION EncuentraA(A : Arbol; x : Elemento) : Arbol;
begin
    IF EsVacio(A) then
        EncuentraA := Nil
        {si el árbol A está vacío, devuelve un árbol vacío}
    else
        IF (A^.Subp = x) then
            EncuentraA := A {el nodo buscado es la raiz}
        else {busca en el subárbol izquierdo y derecho}
            IF (EncuentraA(A^.izq, x) <> Nil) AND
              (EncuentraA(A^.der, x) <> Nil) then
                  EncuentraA := Nil
            else
```

```
        IF EncuentraA(A^.izq, x) <> Nil) then
          EncuentraA := EncuentraA(A^.izq, x)
        else
          EncuentraA := EncuentraA(A^.der, x)
    end;
```

Al terminar el algoritmo, se habrá generado un árbol de ramificación y acotamiento, el cual se manda a imprimir. El procedimiento que imprime el árbol está definido en la unidad de árboles. Este procedimiento dibuja el subárbol izquierdo abajo de la raíz y el subárbol derecho arriba de la raíz. La raíz es el elemento que está hasta la izquierda. El algoritmo que se utiliza es recursivo y realiza los siguientes pasos. Imprime el subárbol derecho con $h + 1$ espacios de sangría, imprime la raíz con $h$ espacios de sangría y por último imprime el subárbol izquierdo con $h + 1$ espacios de sangría.

```
PROCEDURE Imprime_Arbol(A : Arbol; h : integer);
VAR
  i : integer;
begin
  IF NOT EsVacio(A) then
  begin
    Imprime_Arbol(A^.der, h + 1);
    For i := 1 to h Do
      Write(' ');
    WriteLn(A^.subp);
    Imprime_Arbol(A^.izq, h + 1)
  end
end;
```

## 4.2 Problema Dual Lagrangeano.

Recuérdese que el problema dual Langrangeano para el problema del agente viajero es

$$\max_{\lambda}\{w(\lambda)\}$$

donde $w(\lambda) = \min_{\lambda}[C_k + \lambda v_k]$, $C_k$ es el costo del $k - ésimo$ 1-árbol con respecto a los pesos $c_{ij}$. El problema dual Lagrangeano se resuelve con el método de optimización subgradiente. En cada iteración se actualiza el vector $\lambda$ y se encuentra el 1-árbol óptimo con respecto a este valor.

El 1-árbol óptimo está representado en el paquete mediante un arreglo de $n$ entradas donde $n$ es el número de vértices. La $i - ésima$ entrada del

arreglo contiene una lista de los vértices adyacentes al vértice $i$ en el 1-árbol óptimo.

$$1\_Arbol = array[1..n]\ of\ Lista$$

Debido a que es variable el grado de cada vértice en los 1-árboles que se construyen, la lista está representada con estructuras dinámicas mediante elementos enlazados por apuntadores.

```
TYPE
    Elemento = integer;
    Lista = ^Nodo;
    Nodo = Record
               vértice : Elemento;
               sig : Lista
           end;
```

Gráficamente se tiene lo siguiente:

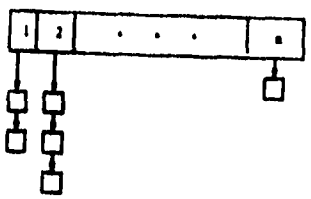


Fig. 4.11 Representación de un 1-árbol.

Para construir el 1-árbol óptimo primero se obtiene el árbol de peso mínimo sobre el conjunto de vértices $\{2,...,n\}$ y después se agregan las 2 aristas de menor peso incidentes al vértice 1. El árbol de peso mínimo se obtiene con el algoritmo de Kruskal. Sea $G = (V, E)$ una gráfica no dirigida donde $|V| = n$ y $|A| = m$. Para aplicar el algoritmo de Kruskal es conveniente ordenar las aristas en orden creciente respecto a su peso. Sean $e_1, e_2, ..., e_m$ las aristas de $G$ donde $c_{e_1} \leq c_{e_2} \leq ... \leq c_{e_m}$. Esto se hace con un algoritmo de ordenamiento conocido como burbuja. En este algoritmo se comienza con el último elemento de la lista y se comparan todas las parejas de elementos sucesivos de la lista en cada iteración. Se conoce como ordenamiento burbuja porque los elementos más pequeños suben al principio de la lista.

ORDENAMIENTO BURBUJA.

1. *For* $j := 1$ *to* $m - 1$ *Do*

```
{Una vez que los primeros j - 1 números han sido ordenados, sólo falta
  por ordenar los números c_j, c_{j+1}, ..., c_m}
    2. For i := 1 to m - j Do
  {Los números consecutivos de la lista c_j, c_{j+1}, ..., c_m se comparan de
   abajo hacia arriba y se realizan los intercambios necesarios}
        IF (c_{m+1-i} < c_{m-i}) then
        begin
          temp := c_{m-i};
          c_{m-i} := c_{m+1-i};
          c_{m+1-i} := temp
        end;
```

Una vez que se han ordenado las aristas se puede aplicar el algoritmo de Kruskal. En cada iteración del algoritmo se realiza lo siguiente. Sea $e$ una arista de menor peso tal que $e \notin S$, donde $S$ es el conjunto de aristas que se van incluyendo para formar el árbol de peso mínimo. Se debe determinar si el conjunto de aristas $S \cup \{e\}$ induce una gráfica acíclica. Para ello se define un arreglo $COMP$ de longitud $n$. Al inicio $COMP[i] = i \quad \forall 1 \leq i \leq n$. En la primera iteración se agrega la arista $e_1 = [j_1, k_1]$ a $S$ y se realiza la siguiente asignación

$$COMP[j_1] = COMP[k_1] \leftarrow \min\{j_1, k_1\}$$

Sin pérdida de generalidad supóngase que $j_1 < k_1$. Entonces $COMP[j_1] = COMP[k_1] = j_1$. Como $COMP[k_1] = j_1$ esto significa que hay una cadena del vértice $j_1$ al vértice $k_1$ en la gráfica que se está construyendo.

En general supóngase que las aristas $e_1, e_2, ..., e_{i-1}$ ya han sido consideradas y que $| S | < n - 1$. La siguiente arista que se debe analizar es $e_i = [j_i, k_i]$. Si $COMP[j_i] = COMP[k_i]$ entonces no se incluye la arista y se procede a analizar la siguiente arista $e_{i+1}$. La arista $e_i$ no se incluye porque se formaría un ciclo. Supóngase que $COMP[j_i] = COMP[k_i] = l$. Esto significa que existe una cadena del vértice $l$ al vértice $j_i, l - j_i$, y del vértice $l$ al vértice $k_i, l - k_i$. Si se incluye la arista $(j_i, k_i)$ se formaría el ciclo $l - j_i, (j_i, k_i), k_i - l$.

Si $COMP[j_i] \neq COMP[k_i]$ entonces se realiza la asignación $S \leftarrow S \cup \{e_i\}$ y las componentes del arreglo se modifican como sigue:

$$m \leftarrow \min\{COMP[j_i], COMP[k_i]\}$$
$$M \leftarrow \max\{COMP[j_i], COMP[k_i]\}$$

para cada vértice $s$ tal que $COMP[s] = M$ hacer $COMP[s] = m$. Esto significa que existe una cadena del vértice $m$ a todos los vértices $s$ tales que $COMP[s] = M$.

Una vez calculado el 1-árbol óptimo con respecto al valor $\lambda$ se tiene una cota inferior $Z_{LB}$ para el valor óptimo del problema del agente viajero. Esta cota se puede mejorar al actualizar $\lambda$ y calcular el 1-árbol de peso mínimo con respecto al nuevo vector de multiplicadores:

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

donde $t_m = \frac{\pi(Z_{UB} - Z_{LB})}{\|v_k^s\|^2}$, $0 < \pi \leq 2$. En la primera iteración se fija $\pi = 2$. Si $Z_{\max}$ (la máxima cota inferior obtenida hasta el momento) no se ha incrementado en las últimas 5 iteraciones, entonces $\pi$ se reduce a la mitad. El algoritmo termina cuando $Z_{\max} = Z_{UB}$, o cuando $\pi < 1.25$. El valor de $Z_{\max}$ en la última iteración es una aproximación de la solución para el problema dual Lagrangeano.

## 4.3 Eficiencia del Paquete de Cómputo.

En esta sección se discute la rapidez de los algoritmos implementados en el paquete de cómputo. El algoritmo de ramificación y acotamiento se probó con 9 ejemplos del problema del agente viajero. A continuación se muestra una tabla que indica el tamaño del *ejemplo* y en cuánto tiempo se obtuvo la solución al usar este algoritmo. La máquina que se usó es una PC **486DX2/62MHz.**

| Ejemplo | Tamaño del Ejemplo | Tiempo (seg.) |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 5 | 2 |
| 3 | 6 | 2 |
| 4 | 6 | 4 |
| 5 | 6 | 6 |
| 6 | 6 | 16 |
| 7 | 7 | 2 |
| 8 | 7 | 3 |
| 9 | 8 | 21 |

Obsérvese que el tiempo requerido para resolver diferentes ejemplos del mismo tamaño puede variar ya que los árboles de ramificación y acotamiento

generados no son los mismos. Pero en general, a medida que crece el tamaño de la instacia aumenta el tiempo. También se puede observar que el tiempo no aumenta en forma lineal. Esto se debe a que el problema del agente viajero es $NP - completo$.

El método de relajación Lagrangeana se probó con 4 ejemplos y se obtuvieron los siguientes resultados:

| Ejemplo | Tamaño | Tiempo(seg.) | Núm. de It. | It. ($z_{max}$) | $Z^*$ | $Z_{max}$ |
|---|---|---|---|---|---|---|
| 1 | 5 | 3 | 4 | 4 | 11 | 11 |
| 2 | 5 | 12 | 46 | 27 | 171 | 160 |
| 3 | 6 | 17 | 46 | 1 | 26 | 24 |
| 4 | 7 | 20 | 46 | 1 | 162 | 153 |

La notación usada en la tabla anterior es la siguiente:

- Núm. de It. Es el número de iteraciones que se realizaron en cada ejemplo.
- It.($z_{max}$). Es la iteraciones en la que se obtuvo el valor de la cota inferior $z_{max}$.
- $Z^*$. Es el valor óptimo del ejemplo.
- $Z_{max}$. Es el valor de la cota inferior generada.

En este caso también se puede observar que a medida que crece el tamaño del ejemplo, el tiempo en el que se obtiene una cota inferior también aumenta de manera no lineal. De los ejemplos anteriores únicamente se obtuvo la solución óptima en el primero y por esta razón el número de iteraciones que se realizaron fueron menos. En los demás ejemplos se realizaron 46 iteraciones, que corresponde al número máximo de iteraciones que realiza el algoritmo. Sin embargo, la máxima cota inferior se obtuvo en la primera iteración para los ejemplo 3 y 4 y en la iteración 27 para el segundo ejemplo. Entonces, para estos ejemplos no es necesario realizar muchas iteraciones puesto que se puede obtener la mínima cota inferior en menos tiempo. Por otro lado, se puede observar que las cotas inferiores generadas están cercanas al valor óptimo.

# Conclusiones

El problema del agente viajero es uno de los problemas de la clase $NP - difícil$ más famosos. Por esta razón se han desarrollado muchas técnicas para tratar de resolverlo eficientemente. Aunque muchas de éstas generan soluciones muy cercanas a la óptima, todavía no se cuenta con un algoritmo polinomial para resolverlo. Por esto es razonable suponer que no se podrá encontrar ningún algoritmo polinomial que resuelva el problema del agente viajero y por lo tanto que la clase $P \neq NP$.

Para resolver una instancia de un problema $NP-difícil$ se pueden seguir varios caminos. Si el tamaño de la instancia es pequeño, se puede resolver con algún algoritmo exacto. Si la instancia pertenece a un caso particular del problema para el cual sí existe un algoritmo polinomial, entonces también se puede encontrar la solución óptima. Pero si no se tiene ninguna de las situaciones anteriores, lo más conveniente es usar un algoritmo heurístico para encontrar una solución aproximada en un periodo de tiempo razonable.

Cuando se aplica una heurística para resolver una instancia de un problema, es importante saber qué tan buena es la solución que se obtiene con este método. Por esta razón es importante generar cotas inferiores del valor óptimo, para el caso de problemas de minimización, con algún algoritmo. El algoritmo más usado es la relajación Lagrangeana debido a que se obtienen cotas inferiores muy cercanas al valor óptimo. Con las heurísticas se generan cotas superiores del valor óptimo. Entonces si el intervalo formado por la cota inferior y la cota superior del valor óptimo es pequeño, la heurística genera soluciones cercanas a la óptima.

En general, se puede observar que si la heurística es muy simple no se generan buenas soluciones. Este es el caso de los algoritmos glotones. Entonces, para resolver problemas de la clase $NP - difícil$ se requieren heurísticas más sofisticadas como el recocido simulado. Este algoritmo es muy aceptado porque se puede usar para resolver cualquier problema combinatorio. Además, si se elige una vecindad conocida, una tasa gométrica de

enfriamiento de 0.95 y una temperatura inicial determinada en base a pocos experimentos, esta técnica resulta fácil de implementar y genera soluciones razonables. Sin embargo, también se puede observar que los algoritmos producen mejores resultados cuando aprovechan la estructura del problema bajo consideración. Entonces, si se aplica esta técnica con un esquema de enfriamiento determinado en base a una amplia experimentación y con una vecindad determinada por un amplio conocimiento del problema, entonces se generan soluciones mucho más cercanas a la óptima.

La técnica de recocido simulado ha dado lugar al desarrollo de nuevos algoritmos donde se hace énfasis en controlar los movimientos a peores soluciones de una manera "inteligente" en lugar de aleatoriamente. El método más conocido es probablemente la búsqueda tabú. Este método proporciona una búsqueda heurística en la cual se implementa un especie de "memoria".

Los algoritmos heurísticos que se han desarrollado recientemente tienen la característica de que simulan procesos naturales. Por ejemplo, el recocido simulado imita un proceso termodinámico. Los algoritmos genéticos se formulan en base a una analogía con las estructuras genéticas. El área de redes neuronales artificiales imita las propiedades computacionales del cerebro humano. Este tipo de algoritmos a dado buenos resultados al resolver problemas difíciles.

Con respecto al paquete de cómputo, se concluye que al implementar el método de relajación Lagrangeana no siempre es necesario realizar muchas iteraciones. En algunos casos se puede disminuir el número de iteraciones y se obtiene la misma cota inferior en menos tiempo. Por lo tanto, al implementar un algoritmo es conveniente hacer varios experimentos con ejemplos de un determinado problema para disminuir lo más posible el tiempo de ejecución.

enfriamiento de 0.95 y una temperatura inicial determinada en base a pocos experimentos, esta técnica resulta fácil de implementar y genera soluciones razonables. Sin embargo, también se puede observar que los algoritmos producen mejores resultados cuando aprovechan la estructura del problema bajo consideración. Entonces, si se aplica esta técnica con un esquema de enfriamiento determinado en base a una amplia experimentación y con una vecindad determinada por un amplio conocimiento del problema, entonces se generan soluciones mucho más cercanas a la óptima.

La técnica de recocido simulado ha dado lugar al desarrollo de nuevos algoritmos donde se hace énfasis en controlar los movimientos a peores soluciones de una manera "inteligente" en lugar de aleatoriamente. El método más conocido es probablemente la búsqueda tabú. Este método proporciona una búsqueda heurística en la cual se implementa un especie de "memoria".

Los algoritmos heurísticos que se han desarrollado recientemente tienen la característica de que simulan procesos naturales. Por ejemplo, el recocido simulado imita un proceso termodinámico. Los algoritmos genéticos se formulan en base a una analogía con las estructuras genéticas. El área de redes neuronales artificiales imita las propiedades computacionales del cerebro humano. Este tipo de algoritmos a dado buenos resultados al resolver problemas difíciles.

Con respecto al paquete de cómputo, se concluye que al implementar el método de relajación Lagrangeana no siempre es necesario realizar muchas iteraciones. En algunos casos se puede disminuir el número de iteraciones y se obtiene la misma cota inferior en menos tiempo. Por lo tanto, al implementar un algoritmo es conveniente hacer varios experimentos con ejemplos de un determinado problema para disminuir lo más posible el tiempo de ejecución.

# Apéndice

# Manual del Paquete de Cómputo.

Al correr el programa aparece la siguiente información acerca del paquete de cómputo.

**PROBLEMA DEL AGENTE VIAJERO**

Este programa resuelve el problema del agente viajero mediante un algoritmo de ramificación y acotamiento que usa como relajación el problema de asignación. El paquete también acota inferiormente el valor óptimo de un problema SIMETRICO con el método de la relajación Lagrangeana.

El número mínimo de ciudades que debe tener el problema es 3 y el máximo es 8.

El programa lee los datos de entrada de un archivo y manda los datos de salida ( la solución ) a otro archivo.

Pulsa la tecla "Enter" para continuar.

Obsérvese que sólo se pueden obtener cotas inferiores del valor óptimo para problemas simétricos. Por otro lado, el número mínimo de ciudades debe ser 3 puesto que de otra manera la solución es trivial.

Al pulsar la tecla "Enter" aparece otra ventana con las operaciones que se pueden realizar:

Operaciones:

1. Resolver el problema con un algoritmo de ramificación y acotamiento.
2. Acotar inferiormente el problema (Relajación Lagrangeana).
3. Salir del programa.

¿Qué operación deseas efectuar?(1,2 o 3)

Si se elige la opción 1 0 2 aparece el siguiente mensaje:

Dame el nombre del archivo donde están los datos de entrada.

Se debe especificar la dirección en la máquina donde se encuentra el archivo. Si se le da un archivo que no existe o una ruta errónea entonces aparece el siguiente mensaje:

Nombre del archivo incorrecto, intenta otra vez.

El archivo con los datos de entrada debe contener la siguiente información:

1. El número de ciudades del problema.
2. Un número $M >> 0$.
3. La matriz de costos del problema. Si se elige la opción 2 del menú sólo se deben meter las entradas $(i, j)$ de la matriz de costos tales que $i < j$.

Para acotar inferiormente el valor óptimo del problema con la relajación Lagrangeana se requieren además los siguientes datos:

1. El vector inicial $\lambda$.
2. Una cota superior $Z_{UB}$ del valor óptimo.

Después de resolver el problema se le pregunta al usuario en qué archivo quiere que aparezca la solución:

| |
|---|
| Dame el nombre del archivo donde quieras que aparezca la solución. |

Este archivo debe ser diferente al de los datos de entrada y además se debe especificar la dirección en la máquina donde se desea que aparezca.

Los datos de salida para la opción 1 del menú son los siguientes:

1. La solución y el valor óptimo del problema.
2. El árbol de ramificación y acotamiento.

Si se elige la opción 2, aparecerá la siguiente información correspondiente a cada iteración del método de optimización subgradiente:

1. El vector $\lambda$.
2. El valor de $\pi$.
3. Una matriz con las distancias actualizadas $c_{ij} + \lambda_i + \lambda_j$.
4. La cota inferior $w(\lambda)$.
5. La mayor cota inferior hasta esta iteración, $Z_{\max}$.
6. El 1-árbol óptimo.
7. La solución del problema dual Lagrangeano, es decir la mejor cota inferior obtenida.

Después de resolver el problema aparece otra vez el menú principal con las operaciones. Si se elige la opción 3, que corresponde a salir del programa, aparece el siguiente mensaje:

| |
|---|
| FIN DEL PROGRAMA:<br>PROBLEMA DEL AGENTE VIAJERO |

En este momento se debe pulsar la tecla "Enter" para salir del programa.

A continuación se muestran los archivos de entrada y de salida al elegir la opción 1 y 2 del menú para resolver el ejemplo 2.2.2 del problema del agente viajero.

Archivo con los datos de entrada para resolver este ejemplo con el método de ramificación y acotamiento:

```
5
100
100   1    2    2    3
1     100  2    3    100
2     2    100  2    2
2     3    2    100  100
3     100  2    100  100
```

**Archivo de salida:**

La solución y el valor óptimo son:
X15 = X21 = X34 = X42 = X53 = 1, Z_2 = 11.00

El árbol de ramificación y acotamiento es:

```
        2
        SUBP 12
        var. ram. : x35 = 0
        X15 = X21 = X34 = X42 = X53 = 1
        Z_2 = 11.00
        SOL. CANDIDATA

1
SUBP 1
var. ram. :
X12 = X24 = X35 = X41 = X53 = 1
Z_1 = 10.00

        3
        SUBP 11
        var. ram. : x35 = 1
        X12 = X24 = X35 = X43 = X51 = 1
        Z_3 = 11.00
        No Prometedor
```

**Archivo con los datos de entrada para acotar inferiormente el valor óptimo de este ejemplo:**

```
5
100
1   2   2   3
    2   3   100
        2   2
            100
0   0   0   0   0
12
```

Archivo de salida:

ITERACIÓN 1
Lamda: (0.0,0.0,0.0,0.0,0.0)
Pi: 2.000
Las distancias actualizadas son

| | | | | |
|---|---|---|---|---|
| 100.00 | 1.00 | 2.00 | 2.00 | 3.00 |
| 1.00 | 100.00 | 2.00 | 3.00 | 100.00 |
| 2.00 | 2.00 | 100.00 | 2.00 | 2.00 |
| 2.00 | 3.00 | 2.00 | 100.00 | 100.00 |
| 3.00 | 100.00 | 2.00 | 100.00 | 100.00 |

La cota inferior,w(Lamda), es: 9.00
Zmax: 9.00

Vértices adyacentes al vértice1: 3, 2
Vértices adyacentes al vértice2: 1, 3
Vértices adyacentes al vértice3: 1, 5, 4, 2
Vértices adyacentes al vértice4: 3
Vértices adyacentes al vértice5: 3

ITERACIÓN 2
Lamda: (0.0,0.0,2.0,-1.0,-1.0)
Pi: 2.000
Las distancias actualizadas son:

| | | | | |
|---|---|---|---|---|
| 100.00 | 1.00 | 4.00 | 1.00 | 2.00 |
| 1.00 | 100.00 | 4.00 | 2.00 | 99.00 |
| 4.00 | 4.00 | 104.00 | 3.00 | 3.00 |
| 1.00 | 2.00 | 3.00 | 98.00 | 98.00 |
| 2.00 | 99.00 | 3.00 | 98.00 | 98.00 |

La cota inferior,w(Lamda), es: 10.00
Zmax: 10.00

Vértices adyacentes al vértice1: 4, 2
Vértices adyacentes al vértice2: 1, 4
Vértices adyacentes al vértice3: 5, 4
Vértices adyacentes al vértice4: 1, 3, 2
Vértices adyacentes al vértice5: 3

ITERACIÓN 3
Lamda: (0.0,0.0,2.0,1.0,-3.0)
Pi: 2.000
Las distancias actualizadas son:

| | | | | |
|---|---|---|---|---|
| 100.00 | 1.00 | 6.00 | 2.00 | -1.00 |
| 1.00 | 100.00 | 6.00 | 3.00 | 96.00 |
| 6.00 | 6.00 | 108.00 | 6.00 | 2.00 |
| 2.00 | 3.00 | 6.00 | 100.00 | 96.00 |
| -1.00 | 96.00 | 2.00 | 96.00 | 92.00 |

La cota inferior,w(Lamda), es: 11.00
Zmax: 11.00

Vértices adyacentes al vértice1: 2, 5

Vértices adyacentes al vértice2: 1, 3, 4
Vértices adyacentes al vértice3: 2, 5
Vértices adyacentes al vértice4: 2
Vértices adyacentes al vértice5: 1, 3

ITERACIÓN 4
Lamda: (0.0,1.0,2.0,0.0,-3.0)
Pi: 2.000
Las distancias actualizadas son:

| | | | | |
|---|---|---|---|---|
| 100.00 | 2.00 | 8.00 | 2.00 | -4.00 |
| 2.00 | 102.00 | 9.00 | 4.00 | 94.00 |
| 8.00 | 9.00 | 112.00 | 8.00 | 1.00 |
| 2.00 | 4.00 | 8.00 | 100.00 | 93.00 |
| -4.00 | 94.00 | 1.00 | 93.00 | 86.00 |

La cota inferior,w(Lamda), es: 11.00
Zmax: 11.00

Vértices adyacentes al vértice1: 2, 5
Vértices adyacentes al vértice2: 1, 4
Vértices adyacentes al vértice3: 4, 5
Vértices adyacentes al vértice4: 3, 2
Vértices adyacentes al vértice5: 1, 3

El valor óptimo es: 11.00

[11] Papadimitriou Christos H. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., New Jersey 1982.

[12] Parker R. Gary, Rardin Ronald L. *Discrete Optimization.* Academic Press, Inc., U.S.A. 1988.

[13] Reeves Colin R. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, Inc., Great Britain 1993.

[14] Sakarovitch Michel *Graphes et Programmation Lineare.* Hermann, Paris 1984.

[15] Salkin Harvey M., Mathur Kamlesh. *Foundations of Integer Programming.* North-Holland, U.S.A. 1989.

[16] Taha. *Investigación de Operaciones, una Introducción.* Representaciones y Servicios de Ingeniería, México 1988.

[17] Winston Wayne L. *Operations Research: Applications and Algorithms.* Duxbury, E.U.A. 1987.

[18] Wirth Niklaus. *Algoritmos y Estructuras de Datos.* Prentice-Hall Hispanoamericana, México 1987.

# Bibliografía


[1] Chartrand Gary, Oellermann Ortrud R. *Applied and Algorithmic Graph Theory.* McGraw-Hill International Editions, 1993.

[2] Garey Michael R., Johnson David S. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeeman and Company, U.S.A. 1979.

[3] M. Held, R.M. Karp (1970). *The Traveling-Salesman problem and Minimum Spanning Trees.* Oper. Res. **18**, 1138-1162.

[4] M. Held, R.M. Karp (1971). *The Traveling-Salesman Problem and Minimum Spanning Trees: Part II.* Math. Programming 1, 6-25.

[5] M. Held, P. Wolfe, H.P. Crowder (1974). *Validation of Subgradient Optimization.* Math. Programming **6** , 62-88.

[6] Hillier, Gerald J. Lieberman. *Introducción a la Investigación de Operaciones.* McGraw Hill. México 1991.

[7] Katta Murty. *Linear and Combinatorial Programming.* John Wiley & Sons, U.S.A. 1976.

[8] Lawler, Lenstra, Kan, Shmoys. *The Traveling Salesman Problem. A Guide of Combinatorial Optimization.* John Wiley & Sons, Great Britain 1990.

[9] T.S. Motzkin, I.J. Schoenberg (1954). *The Relaxation Method for Linear Inequalities.* Canad. J. Math. **6**, 393-404

[10] Nering Evar P., Tucker Albert W. *Linear Programs and Related Problems.* Academic Press, Inc., U.S.A. 1993.

[11] Papadimitriou Christos H. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., New Jersey 1982.

[12] Parker R. Gary, Rardin Ronald L. *Discrete Optimization.* Academic Press, Inc., U.S.A. 1988.

[13] Reeves Colin R. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, Inc., Great Britain 1993.

[14] Sakarovitch Michel *Graphes et Programmation Lineare.* Hermann, Paris 1984.

[15] Salkin Harvey M., Mathur Kamlesh. *Foundations of Integer Programming.* North-Holland, U.S.A. 1989.

[16] Taha. *Investigación de Operaciones, una Introducción.* Representaciones y Servicios de Ingeniería, México 1988.

[17] Winston Wayne L. *Operations Research: Applications and Algorithms.* Duxbury, E.U.A. 1987.

[18] Wirth Niklaus. *Algoritmos y Estructuras de Datos.* Prentice-Hall Hispanoamericana, México 1987.