UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA

APUNTES DE PROGAMACIÓN ENTERA

Idalia Flores de la Mota

Departamento de Sistemas
División de Estudios de Posgrado
Facultad de Ingeniería

SOBRE LA AUTORA

Idalia Flores De La Mota

Estudió el Doctorado en Investigación de Operaciones en la Facultad de Ingeniería de la UNAM. Obtuvo mención honorífica por su tesis de maestría así como la medalla Gabino Barreda por el mejor promedio de su generación. Ha asistido a varios congresos nacionales e internacionales, y ha sido miembro del Sistema Nacional de Investigadores. Ha publicado en memorias de congresos así como apuntes para las materias de Matemáticas Aplicadas, Teoría de Redes y Programación Entera. Ha impartido cursos en universidades del extranjero, en la Facultad de Economía de la Universidad de la Habana, Cuba así como en la Universidad Nacional Autónoma de Nicaragua. Es profesora de tiempo completo de la Facultad de Ingeniería desde 1990 y actualmente es la Jefa de la Sección de Investigación de Operaciones de la División de Estudios de Posgrado de la Facultad de Ingeniería.

Como parte de las actividades del Departamento de Sistemas de la División de Estudios de Posgrado de la Facultad de Ingeniería UNAM, nos hemos propuesto el desarrollo de una serie de apuntes que sirvan de apoyo a los diferentes cursos que se imparten y, desde luego, como material de referencia o de lectura para quienes así lo deseen.

En dichos apuntes se busca mantener un alto nivel, de manera que contribuyan a una sólida formación teórica del alumno, pero al mismo tiempo se intenta acercarlo a las aplicaciones de tales conocimientos, como es en síntesis el objetivo central del posgrado de ingeniería.

La estructura de estos apuntes consiste de siete capítulos, en los cuales se desarrolla cada uno de los temas con unas notas históricas al final de cada capítulo.

Como autora de estos apuntes quiero expresar mi agradecimiento a Griselda Aguila Ramírez, por su colaboración en la elaboración de los mismos, así como a la Lic. Guadalupe Castro por la revisión y comentarios que hicieron mejorar la calidad de este trabajo.

PRÓLOGO

La Programación Entera es una disciplina dentro de la Investigación de Operaciones, que se ha desarrollado ampliamente en las últimas dos décadas. Junto con algunas técnicas de teoría de gráficas se han desarrollado algoritmos que se pueden ubicar también dentro de la Optimización Combinatoria como veremos en estos apuntes. Por otro lado el desarrollo de la computación ha traído consigo un gran avance en las técnicas de la programación entera, es por esto que es de suma importancia que el estudiante cuente con bases sólidas en esta disciplina. Es bajo esta perspectiva, que se presentan estos apuntes donde lo que se busca es tener un apoyo didáctico para la clases y como un complemento de la bibliografía sugerida para el curso.

Los presentes apuntes van dirigidos a estudiantes de posgrado que ya tienen conocimientos de la Programación Lineal y desean profundizar en problemas de tipo entero. No se presentan todos los algoritmos que se pueden cubrir en el curso, sin embargo se sientan bases para poder trabajar aquellos que están ausentes con cierta facilidad. El propósito de estos apuntes es que los alumnos que cursan la materia y requieren de un cierto grado de conceptualización y una buena dosis de ejemplos tengan un documento de trabajo que les facilite el acceso a otras lecturas; se presentan también al final de cada capitulo unas notas históricas que buscan dar al lector una idea más completa de la programación entera.

A diferencia de la edición anterior en la presente se hace una reorganización de los temas vistos en la misma del capítulo uno al capítulo cinco y se introducen los métodos heurísticos en los capítulos seis y siete.

CAPÍTULO 1
CONCEPTOS BÁSICOS

> "Es muy difícil construir una única teoría capaz de describir todo el universo. En vez de ello, nos vemos forzados de momento, a dividir el problema en varias partes, inventando un cierto número de teorías parciales."
> Stephen W. Hawking. *Historia del Tiempo*

1.1 INTRODUCCIÓN

La toma de decisiones ha estado presente a lo largo de toda la historia de la humanidad. Sin embargo la racionalización y sistematización de esta tarea en problemas industriales y públicos de envergadura, solo fue posible hasta fines de la Segunda Guerra Mundial al desarrollarse lo que hoy conocemos como Investigación de Operaciones. Dos factores importantes confluyeron e impusieron un rápido crecimiento a la nueva disciplina, como base fundamental para la toma de decisiones en amplias esferas de la sociedad:

El primero, fue el progreso sustancial en la modelación matemática y la consolidación del método simplex para resolver problemas de programación lineal, desarrollada por el matemático G. Dantzig, así como la programación dinámica, la teoría de inventarios y la teoría de colas que estaban relativamente bien desarrolladas antes de finalizar los años 50.

El segundo factor fue, la irrupción del computador digital que proporcionó al tomador de decisiones una tremenda capacidad en velocidad de cómputo, almacenamiento y retiro de información. Permitiendo la aplicación y popularización de las herramientas convencionales de la I. de O.

Estos factores han dejado profunda huella durante estos primeros 50 años de desarrollo de la I. de O. El primero de ellos, sobre todo la programación lineal, tuvo un papel predominante en las técnicas de solución de problemas, mientras que el segundo juega un papel de apoyo al primero, al permitir efectuar largos procedimientos de cálculo. Esto ocasionó una de las primeras desvirtuaciones que tenemos de la I. de O., la de reducirla a una simple herramienta de solución de problemas, más aún, de confundir la investigación de operaciones con la programación lineal, este hecho se manifiesta en planes de estudio, de maestrías en administración y diversas licenciaturas donde existe la materia de Investigación de Operaciones, con programas correspondientes relativos a la programación lineal.

La Investigación de Operaciones, tuvo sus orígenes a lo largo de la segunda guerra mundial. La ciencia ha contribuido, desde los tiempos de Arquímedes, a aportar ideas destructoras, y en las dos grandes guerras de este siglo dio su ayuda técnica para hacer posible el desarrollo de sus principales armas, desde la ametralladora hasta la bomba atómica. En la última guerra, los analistas de

operaciones militares se encontraron trabajando en lugares extraños y en diversas circunstancias. En Burma hubo matemáticos que discutieron problemas artilleros con soldados británicos, en Princess Risborough, un cuartel seguro fuera de Londres; unos químicos, en combinación con colegas economistas valoraron la capacidad destructora de una bomba; hubo generales que consultaron la estrategia de los carros de combate en la campaña de Italia con bioquímicos y abogados; un famoso zoólogo británico fue el hombre clave en el trazado de un plan de bombardeo sobre Pantellaria; oficiales de marina pusieron bajo secreto a estadísticos y entomólogos, en relación con las pérdidas de submarinos en el Pacífico. Fue un intenso, irregular y paradójico intercambio de ideas entre aficionados y profesionales de la guerra, que produjo algunos éxitos brillantes, aunque hubo más fracasos que éxitos, pero el balance final es impresionante.

Lo que aportaron los científicos a los problemas operativos -aparte de sus conocimientos especiales- fue su aspecto científico. De hecho ésta fue su principal contribución, aportaron ideas nuevas, desecharon conceptos preconcebidos y obraron sólo ante la evidencia. El cálculo de probabilidades fue su herramienta indispensable, e hicieron uso de su teoría más sutil y de su técnica más eficaz, no estuvieron limitados por axiomas de laboratorio, sino que su guía inseparable fue el sistema experimental. Sin embargo a pesar de haber empezado en la guerra, la investigación de operaciones, se ha ido extendiendo a la ingeniería, las comunicaciones, la minería, los negocios, la manufactura y otras ramas de la industria.

En nuestro país la Investigación de Operaciones es realmente muy joven ya que hasta hace aproximadamente veinte años se comenzó a trabajar en forma sistemática. El primer paso en este sentido fue la familiarización con las técnicas básicas de la I. de O. Como era de esperarse, al principio - y actualmente todavía sucede aunque en menor grado- lo que se hizo fue copiar las aplicaciones desarrolladas mecánicamente, esto es, sin ninguna creatividad o esfuerzo por usar las herramientas de la I. de O., como lo que son, instrumentos de trabajo para la solución de problemas.

El creciente uso de la I. de O. ha hecho que sus técnicas se vayan desarrollando y diferenciando más, teniendo un desarrollo propio. Entre otras podemos identificar la Teoría de Juegos, Teoría de Colas, Simulación Digital, Procesos Estocásticos y Optimización; la tabla 1 nos muestra esquemáticamente ésta idea, y desarrollamos el modulo de optimización debido a la importancia para el presente documento de acuerdo a los diferentes problemas que resuelve. Problemas como los de distribución de un producto a costo mínimo, redes eléctricas o hidráulicas, o expansión de capacidad, entre otros.

Con relación a las técnicas de optimización podemos decir que la Programación Lineal se ha usado con éxito en la solución de problemas referentes a la asignación de personal, la mezcla de materiales, la distribución y el transporte y las carteras de inversión. La programación dinámica se ha aplicado con buenos resultados en áreas tales como la planeación de los gastos de comercialización, la estrategia de ventas y la planeación de la producción. La teoría de colas ha tenido aplicaciones en la solución de problemas referentes al congestionamiento del tráfico, al servicio de máquinas sujetas a descomposturas, a la determinación del nivel de la mano de obra, a la programación del tráfico aéreo, al diseño de presas, a la programación de la producción y a la administración de hospitales. Otras técnicas de Investigación de Operaciones, como la teoría de inventarios, la teoría de juegos y la simulación, han tenido exitosas aplicaciones en una gran variedad de contextos.

En cuanto al módulo de optimización es conveniente puntualizar que algunos problemas reales, exigen que sus soluciones sean enteras, esto es, las variables de decisión tienen sentido sólo si adquieren valores enteros. Por ejemplo, es necesario con frecuencia asignar hombres, máquinas y vehículos a las actividades, en cantidades que deben ser enteras. Esta restricción es difícil de manejarse en forma matemática, sin embargo, se han hecho algunos progresos en el desarrollo de procedimientos de solución para el caso de problemas de programación lineal, sujetos a la restricción adicional de que las variables de decisión (todas o algunas) deben tener valores enteros.

Este capítulo se desarrolla como sigue: en la segunda sección se presentan en general problemas de optimización combinatoria así como un bosquejo general de la complejidad de dichos problemas, en la tercera sección se presenta un esquema general de los diferentes métodos para resolver problemas enteros y combinatorios, tanto exactos como heurísticos y finalmente en la cuarta sección se presentan algunos ejemplos representativos del tipo de problemas que se resuelven en estos apuntes.

1.2 PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

Un problema combinatorio es aquél que asigna valores numéricos discretos a algún conjunto finito de variables X, de tal forma que satisfaga un conjunto de restricciones y minimice o maximice alguna función objetivo.

Los problemas de optimización combinatoria abundan en la vida diaria. Una área importante y extensa de aplicaciones se refiere a la administración eficiente del uso de recursos escasos para incrementar la productividad. Estas aplicaciones incluyen problemas operacionales tales como distribución de bienes, planeación de la producción y secuenciación de máquinas. También incluyen problemas de planeación tales como inversión de capital, localización de medios y selección de cartera. También problema de diseño como diseño de redes de telecomunicación

y transporte, diseño de circuitos y diseño de sistemas de producción automática. Los problemas de optimización discreta también se presentan en estadística (análisis de datos), física (determinación de estados de energía mínimos), criptografía (diseñando códigos infranqueables), política (seleccionando distritos electorales favorables) y en matemáticas (como una técnica poderosa para probar teoremas combinatorios). Como ya se ha mencionado el gran avance de las computadoras ha repercutido en el desarrollo acelerado de la optimización discreta.

COMPLEJIDAD DE LOS PROBLEMAS.

Podemos clasificar los problemas en cuatro clases de acuerdo a su grado de dificultad:

1. Problemas indecidibles. Son aquellos problemas para los cuales no se puede escribir un algoritmo. Por ejemplo, se ha probado que un programa que se detendrá en una Máquina de Turín (1937) es de esta clase. El problema de Turing al igual que el décimo problema de Hilbert y el de programación cuadrática en enteros son problemas indecidibles, es decir no sólo no existe un algoritmo polinomial para su solución sino que no existe algoritmo.

2. Problemas Intratables (problemas que se demuestran son difíciles): Son aquellos problemas para los cuales no se pueden desarrollar algoritmos polinomiales. En otras palabras, sólo se pueden resolver con algoritmos exponenciales.

3. Problemas NP (donde NP se entiende por polinomial no determinístico) Esta clase incluye problemas que se pueden resolver en tiempo polinomial si podemos *adivinar* correctamente que ruta computacional se puede seguir. El concepto de *adivinar* es extraño ya que todos los programas computacionales son determinísticos. En general, esta clase incluye todos los problemas que tienen algoritmos exponenciales pero que no se ha probado que no se puedan resolver con algoritmos de tiempo polinomial.

4. Problemas P (donde P se entiende por polinomial). Esta clase incluye todos los problemas que tienen algoritmos de tiempo polinomial. Mucha gente considera a esta clase como una subclase propia de la clase 3.

   Un problema se dice polinomial si existe un algoritmo para el cual el tiempo requerido para su solución, está acotado por una función polinomial del tamaño del problema (donde entendemos el tamaño del problema como la longitud de un código, por ejemplo binario de los datos del problema). Se tiene así por ejemplo que en una gráfica G = [X,A] con N=X nodos y M = A arcos, una ruta más corta se encuentra a lo mas en un tiempo O(MN), un flujo máximo en un tiempo $O(N^3)$, un árbol de peso mínimo en un tiempo $O(N^2)$. Sin embargo no todos los problemas combinatorios son polinomiales.

Problemas como el del agente viajero o el de la mochila se conocen como NP-completos, donde NP se entiende como polinomial no determinístico. El problema del agente viajero se puede resolver con un algoritmo determinístico como el de recursividad, donde se especifica en cada paso que arco se debe considerar desde un nodo dado.

Para la pregunta de que si existe un recorrido con una distancia total que sea menor que B, el algoritmo recursivo implícitamente prueba todos los recorridos posibles y da una respuesta afirmativa si tal recorrido existe y negativa en caso contrario.
Ya que los recorridos se prueban uno por uno, y el último puede ser el que cumple con la propiedad deseada, el algoritmo recursivo para resolver el problema del agente viajero se considera $O(c^n)$.

Un algoritmo no determinístico puede *adivinar correctamente* que arco se debe incluir en el recorrido. Si existe un recorrido con una distancia total menor que B, el algoritmo no determinístico requiere un tiempo de O(n) para calcular la distancia total del recorrido y verificar si tal recorrido existe. Si el problema del agente viajero se puede resolver por un algoritmo no determinístico en tiempo polinomial se dice que el problema pertenece a la clase NP.

Existe un subconjunto de problemas NP que son los más difíciles, en el siguiente sentido (los problemas en el subconjunto son polinomialmente equivalentes):

Cualquier problema en NP se puede reducir a cualquier problema en éste subconjunto, podemos resolver todos los problemas en NP en tiempo polinomial. Los problemas en éste subconjunto se llaman NP- completos. Esquemáticamente se puede ver en la figura 1.1

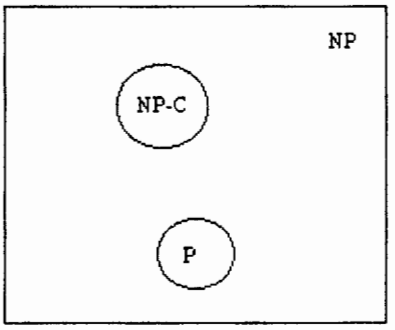


Figura 1.1

A continuación se listan algunos problemas representativos que aborda la optimización combinatoria:

El problema del agente viajero: Dada una gráfica (dirigida o no) con pesos específicos en los arcos, determine una trayectoria cerrada que incluya cada nodo de la gráfica una sola vez y que tenga un peso mínimo total.

El problema del cartero chino: Para una gráfica dada (dirigida o no) con pesos específicos en los arcos, determine una trayectoria que incluya cada arco en la gráfica al menos una vez y que su peso total sea mínimo.

El problema de la mochila: Determine un conjunto de valores enteros x_i i = 1, 2,...,n que minimice $f(x_1, x_2,...,x_n)$ sujeto a la restricción $g(x_1, x_2,...,x_n) \geq b$ donde b es un parámetro.

Planeación de máquinas en paralelo: Dado un conjunto T de tareas simples de operación, cada una con tiempo de procesamiento τ_j , $1 \leq j \leq |T|$, asignar cada tarea a exactamente m máquinas tal que el tiempo en que se realizan todas las tareas sea mínimo.

Coloración de nodos: Dada una gráfica no dirigida, determine el número mínimo de colores necesarios para colorear cada nodo de la gráfica de tal forma que ningún par de nodos adyacentes (nodos conectados por un arco) tengan el mismo color

Árbol de búsqueda: Para una gráfica dada con pesos específicos en los arcos, determinar un subconjunto de peso mínimo total de arcos que forme una gráfica acíclica conectada que tiene al menos un arco incidente a cada vértice.

Ruta más corta: Para una gráfica dada (dirigida o no) con peso o longitudes específicas en los arcos, encontrar una sucesión de arcos no repetidos de longitud total mínima que conecte dos vértices específicos y que sea factible a la dirección de los arcos.

Empacado de cajas: Para una lista de N pesos w_i, $1 \leq i \leq N$ y un conjunto de cajas, cada una de ellas con una capacidad fija, sea W, encontrar una asignación factible de pesos para las cajas que minimice el número total de cajas a usar.

Apareamiento: Dada una lista de artículos i=1,2,...,n y pesos w_{ij} asociados con aparear un artículo i con un artículo j, encontrar un esquema de peso máximo total para aparear los artículos en la lista tal que cada artículo este apareado con, a lo más, otro artículo.

Cubierta de conjuntos: Dado un conjunto finito S y una familia de subconjuntos $\{S_j \subseteq S \mid j \in J\}$ y costos asociados C_j a cada S_j, elegir una colección de subconjuntos de costo mínimo total que incluya a cada elemento de S al menos una vez.

Flujo máximo: Dada una gráfica (dirigida o no) y capacidades específicas en los arcos, encontrar un flujo máximo entre dos vértices específicos que sea factible con respecto a las capacidades.

Problema de Cargo Fijo: Dado un conjunto factible S de actividad o niveles de tráfico no negativos, $x = (x_1, x_2,...,x_n)$, costos unitarios v_j para emplear x_j, y costos fijos f_j asignados siempre que x_j sea positiva, seleccionar un costo mínimo local $x \in S$

Problemas como el del agente viajero tienen amplia aplicabilidad al asignar rutas, con un costo mínimo, por ejemplo un comerciante que desea recorrer n ciudades específicas con una distancia total mínima, entre otros. Otro caso es el problema de la mochila que entre otros, se aplica a problemas industriales, tales como: problemas de cargo fijo, selección de proyectos, corte en inventarios y control de presupuestos. Los problemas de asignación también tienen aplicación en inversiones de capital para proyectos independientes, donde además se combinan con problemas de tipo mochila 0-1.

El problema del cartero chino tiene también una amplia gama de aplicaciones tales como: el trazado de rutas de camiones de servicio público para la recolección de desechos sólidos, en el reparto de artículos para una comunidad, o en el recorrido de las plumas de un graficador para el trazado de mapas.

1.3 MÉTODOS DE SOLUCIÓN PARA OPTIMIZACIÓN ENTERA Y COMBINATORIA

Un problema de optimización discreta se puede expresar como:

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

sujeto a

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad \text{para } i = 1, 2,...,m$$

$$x_j \geq 0 \qquad \text{para } j = 1, 2,...,n$$

$$x_j \text{ entero} \qquad \text{para } j \in Z$$

Dicho problema tiene un número finito de soluciones a considerar para identificar una óptima, se puede pensar en que dicho problema se puede resolver por enumeración total, esto es, tratando todas las posibilidades. Sin embargo, el esfuerzo computacional involucrado en la enumeración total se puede convertir en prohibitivo. Por ejemplo un problema con 200 variables de la forma 0 -1 puede tener alrededor de 2^{200} o 10^{60} casos a considerar, uno con 201 variables puede

tener el doble, de aquí la necesidad de desarrollar métodos de enumeración parcial, como los de ramificación y acotamiento o métodos de descripción de poliedros como los de planos de corte; existen métodos que combinan ambos y algunos otros que se aplican en forma pura o en combinación con estos como programación dinámica con ramificación y acotamiento o teoría de grupos con planos de corte, sin embargo todos estos métodos son exactos y aunque en los útlimos años ha habido un gran desarrollo de ellos, gracias también al avance en la computación, existen otros métodos llamados heurísticos que también nos dan si no soluciones exactas si soluciones muy cercanas a ellas, estos métodos se desarrollaron debido a que los métodos exactos eran muy costosos en cuanto a tiempo y espacio de almacenamiento computacional. Con base en todo esto se han desarrollado algoritmos híbridos de los cuales hablaremos en el último capítulo de estos apuntes.

A continuación enunciamos brevemente cada uno de ellos y a grandes rasgos en que consisten.

1. *MÉTODOS EXACTOS*
a) Método de planos de corte.
b) Método de Ramificación y Acotamiento.
c) Método de la Teoría de grupos.
d) Métodos de la programación dinámica.

2. *MÉTODOS HEURÍSTICOS*
a) Métodos Constructivos
b) Métodos de Descomposición
c) Métodos de Reducción
d) Manipulación del Modelo
e) Métodos de Búsqueda por entorno

Estos métodos se engloban en lo que también se conoce como Optimización Combinatoria, término que ha emergido en años recientes para describir aquéllas áreas de la programación matemática que se refieren a la solución de problemas de optimización que tienen una estructura pronunciadamente combinatoria o discreta. Este título se usa para unificar términos que cubren Programación Entera, Teoría de Gráficas, partes de Programación Dinámica etc. A continuación se da una breve descripción de los métodos arriba mencionados.

1. MÉTODOS EXACTOS.

a) Método de Planos de Corte: la idea básica es la siguiente: se comienza resolviendo el problema contínuo lineal, si se encuentra la solución óptima en un punto extremo con coordenadas enteras, entonces el problema queda resuelto. En otro caso, se ve fácilmente que podemos cortar el conjunto solución (aumentando restricciones extras al problema) hasta eliminar este punto extremo

sin excluir una solución entera. Tal restricción se llama un plano de corte.

Si los planos se escogen adecuadamente en cada etapa, entonces el poliedro inicial τ se reducirá progresivamente hasta que coincida con la cubierta convexa de las soluciones enteras, al menos en una vecindad de la solución óptima.

Desafortunadamente y esta es la dificultad esencial, no se conoce un método sistemático para generar todas las ecuaciones o desigualdades lineales que definen la cubierta convexa de los puntos enteros contenidos en un poliedro convexo dado.

b) El Método de Ramificación y Acotamiento: consiste esencialmente en dividir el conjunto de soluciones factibles de un problema en subconjuntos donde se buscará la solución óptima, desechando aquellos que bajo un criterio o una cota no son susceptibles de tomarse en consideración, ahorrando con esto una considerable cantidad de tiempo y esfuerzo. Actualmente las técnicas de ramificación y acotamiento se están empleando en varios campos de aplicación tales como: problemas de distribución, secuenciación de instalaciones, agente viajero,rutas de vehículos, el problema de la mochila, problemas de programación no lineal; el progreso que se ha tenido se debe en gran parte al desarrollo que han tenido las computadoras, ya que por su gran velocidad se tiene una solución más exacta y rápida sin necesidad de recurrir a métodos aproximados.

Conviene mencionar que se dispone también de paquetes computacionales que usan ramificación y acotamiento(R-A) en sus procedimientos de solución, tales como el LINDO para resolver problemas de programación lineal y entera Además se han desarrollando algoritmos de R-A para computación en paralelo.

c) Métodos de la Teoría de Grupos: En un problema de programación entera los coeficientes de las desigualdades forman un conjunto finito que es cerrado bajo la suma cuando las operaciones aritméticas se toman modulo 1. Tal conjunto constituye un grupo, más aun, conforma un grupo abeliano, que además puede tener a lo más D elementos, donde D es el valor absoluto del determinante de la base actual del programa lineal y que a menudo puede generarse por un elemento en cuyo caso el grupo se llama grupo cíclico.

Un problema de grupo puede tratarse como un problema entero con una restricción y también como un problema de redes donde se desea encontrar la ruta más corta. Los algoritmos que resuelven el problema entero son usualmente del tipo de programación dinámica.

d) Métodos de la Programación Dinámica: Los problemas de optimización combinatoria a menudo pueden formularse como problemas de programación dinámica que no cuenta con una formulación matemática estándar para un problema, sino que es un enfoque de tipo general para la solución de problemas y las ecuaciones específicas que se usan se deben desarrollar para que

representen cada situación individual. Entonces, se necesita un cierto grado de creatividad y un buen conocimiento de la estructura general de los problemas de programación dinámica para reconocer cuándo un problema se puede resolver por medio de éstos procedimientos y cómo esto se puede llevar a cabo. Las características principales de los problemas de programación dinámica son a grandes rasgos las siguientes:

1. El problema se puede dividir en etapas que requieren una política de decisión en cada una de ellas.
2. Cada etapa tiene un cierto número de estados asociados a ella.
3. El efecto de la política de decisión en cada etapa es transformar el estado actual en un estado asociado con la siguiente etapa (Tal vez de acuerdo a una distribución de probabilidad)
4. El procedimiento de solución está diseñado para encontrar una política óptima para el problema completo, es decir, una receta para las decisiones de la política óptima en cada etapa para cada uno de los estados posibles.
5. Dado el estado actual, una política óptima para las etapas restantes es independiente de la política adoptada en etapas anteriores.
6. El procedimiento de solución se inicia al encontrar la política óptima para la última etapa.
7. Se dispone de una relación recursiva que identifica la política óptima para la etapa n, dada la política para la etapa (n + 1).
8. Cuando se usa esta relación recursiva el procedimiento de solución se mueve hacia atrás etapa por etapa (encontrando cada vez la política óptima para esa etapa) hasta que se encuentra la política óptima desde la etapa inicial.

2. MÉTODOS HEURÍSTICOS

Dado el carácter no polinomial de una buena parte de los problemas de optimización combinatoria, se hace necesario contar con buenos métodos heurísticos, entre los cuales destacan:

a) Métodos Constructivos. Los cuales construyen soluciones usando reglas heurísticas en forma determinística y secuencial. El más popular de estos métodos lo constituyen los algoritmos glotones (greedy), los cuales construyen paso a paso la solución buscando el máximo beneficio en cada paso.

b) Métodos de Descomposición. Consisten en subdividir el problema en pequeñas particiones. Una aplicación a un problema de programación lineal mixta, consiste en decidir de alguna forma (quizá mediante otra heurística) una solución para las variables enteras, y luego resolver el problema lineal obtenido al sustituir esas variables por su valor. Algunos autores diferencian entre métodos de descomposición (los subproblemas se resuelven en cascada) y métodos de partición (los subproblemas son independientes entre si).

c) **Métodos de Reducción**. Tratan de identificar alguna característica que presumiblemente deba poseer la solución óptima y de ese modo simplificar el problema. Así, puede detectarse que alguna variable deba tomar siempre el valor 0, que otras están correlacionadas, etc.

d) **Manipulación del Modelo.** Intentan reducir el tamaño del problema, mediante la acción combinada de planteamientos estadísticos y combinatorios. Por ejemplo, removiendo del problema rasgos comunes a varias soluciones obtenidas por métodos diferentes o diversas aplicaciones de un mismo método.Así como reducción del espacion de soluciones, linearizar funciones, agrupar variables

e) **Métodos de Búsqueda por Entornos**. Cuyo objetivo es perfeccionar una solución existente, mediante una búsqueda local en una vecindad bien definida del espacio de soluciones. Entre estos métodos encontramos, Recocido Simulado, Búsqueda Tabú, Algoritmos Genéticos, Redes Neuronales y GRASP.

1.4 ALGUNOS CASOS DE APLICACIÓN

1. CASO IBM

La IBM, se dedica al desarrollo y fabricación de minicomputadoras para negocios. Alfredo Dominguez, vicepresidente de Planeación e Investigación de la IBM, está considerando cinco posibles proyectos de investigación. Cada uno de éstos tiene gran potencial de producir buenas utilidades para la IBM, pero también son bastante costosos. En la tabla 1 se muestra el pronóstico de utilidades potenciales y de costos de desarrollo para cada uno de los cinco proyectos.

Proyecto	Utilidad potencial$	Costo de desarrollo $
A	12 Millones	2 Millones
B	15 Millones	5 Millones
C	8 Millones	4 Millones
D	5 Millones	1 Millones
E	11 Millones	6 Millones

Tabla 1. Datos de costo y utilidades para la IBM.

Al mismo tiempo que a la IBM le gustaría maximizar el potencial de utilidades, la compañía está limitada porque sólo dispone de $9 millones para investigación y desarrollo. Los proyectos que se están considerando son de tal naturaleza que el desarrollo parcial de un proyecto no es permisible. Es decir, si se elige el desarrollo de un proyecto, entonces deben asignarse fondos para su desarrollo completo.

Alfredo Dominguez debe determinar qué proyecto va a desarrollarse. Es evidente que no pueden desarrollarse todos los proyectos puesto que la suma de sus

costos es $18 millones, lo que supera los $9 millones disponibles. Se da cuenta que el problema tiene algunos aspectos de programación lineal, pero el requerimiento de que los proyectos se financien en forma completa impide el uso de la programación lineal.

PLANTEAMIENTO:

Maximizar $z = 12x_1 + 15x_2 + 8x_3 + 5x_4 + 11x_5$

Sujeto a

$$2x_1 + 5x_2 + 4x_3 + 1x_4 + 6x_5 \leq 9$$
$$x_1, x_2, x_3, x_4, x_5 = 0 \text{ ó } 1$$

Los problemas que tienen variables binarias y una sola restricción, se conocen como problemas binarios tipo mochila. El nombre viene del concepto de los exploradores que desean llenar una mochila (paquete) de tamaño limitado con tantos artículos como sea posible y en donde cada articulo tiene un peso (o volúmen) diferente y un determinado valor para el explorador.

Los problemas tipo mochila son importantes por varias razones. En primer lugar, se han expresado otros problemas diversos de programación en enteros en forma de problemas tipo mochila, permitiendo aplicar el procedimiento de solución que se deriva de éstos a otros problemas de programación en enteros. También, se han integrado problemas tipo mochila en procedimientos que permiten ayudar a resolver otros problemas diversos. Esto se hace porque los procedimientos de solución de los problemas tipo mochila son bastante eficientes; por ejemplo, en una computadora moderna se han resuelto problemas con 1,000 variables en menos de 2 segundos.

2. MEXICANA DE AVIACIÓN

Debido a una escasez de gasolina, la demanda de boletos de Mexicana de Aviación ha aumentado mucho en los últimos meses. La demanda ha crecido tanto que ahora la aerolínea está analizando la posibilidad de adquirir varios aviones nuevos. Existen tres tipos de aviones de entre los cuales se pueden elegir el DC-33, el Boeing 797 y el Lockheed-Bi-Star. En la tabla 2 se muestran el costo, capacidad y tiempo requerido de mantenimiento mensual para cada tipo. Mexicana desea adquirir los nuevos aviones al mínimo costo posible, sujeta a los requerimientos de capacidad y mantenimiento.

Los nuevos aviones deben transportar un total combinado de cuando menos 3,400 pasajeros y deben tener un tiempo combinado total de mantenimiento que no exceda las 250 horas mensuales. Se complica aún más la decisión de qué aviones adquirir porque sólo existen disponibles para su compra cinco aviones Bi-Star.

AVION	Costo (en millones)	Capacidad	Tiempo de mantenimiento (horas por mes)
DC-33	10	350	25
Boeing 797	15	450	15
Lockheed Bi-Star	12	400	15

Tabla 2

PLANTEAMIENTO:

Este problema es muy similar a uno de programación lineal, excepto porque las variables deben asumir valores enteros. No hay forma de comprar una fracción de avión o de diferir la compra a otro periodo para evitar el requerimiento de enteros. En este caso, debe comprarse a la vez un número entero de aviones. Por ello, tenemos un problema de programación de enteros.

Para plantear el problema, sean:

x_1 = número de DC-33 que se compran
x_2 = número de B -797 que se compran
x_3 = número de Bi-Star que se compran

Dado que nuestro objetivo es minimizar el costo total de adquisición, la función objetivo es

Minimizar $Z = 10x_1 + 15x_2 + 12 X_3$

Los requerimientos de capacidad y tiempo de mantenimiento por mes pueden plantearse de la siguiente manera:

REQUERIMIENTO DE CAPACIDAD: $350x_1 + 450x_2 + 400X_3 \geq 3400$

TIEMPO DE MANTENIMIENTO: $25x_1 + 15x_2 + 15X_3 \leq 250$

Además, debe incluirse la disponibilidad de Bi-Star en el modelo:

$$x_3 \leq 5$$

Por último, están las condiciones de no negatividad y los requerimientos de enteros:

$$x_1, x_2, x_3 \geq 0 \text{ y enteros}$$

En conjunto, el problema puede plantearse como:

$$\text{Min } z = 10x_1 + 15x_2 + 12X_3$$

Sujeto a

$$350x_1 + 450x_2 + 400x_3 \geq 3400$$
$$25x_1 + 15x_2 + 15x_3 \leq 250$$
$$x_3 \leq 5$$
$$x_1, x_2, X_3 \geq 0 \text{ y enteros}$$

Este es un ejemplo de problema general de programación en enteros, puesto que existen sólo variables enteras y éstas no se restringen a 0 o 1.

3. LA COMPAÑIA FERNANDEZ

La Compañía Fernández enfrenta el problema de determinar qué proyectos de crecimiento debe emprender en los proximos cuatro años. La compañía tiene una cantidad de fondos limitada para inversiones de capital; por lo tanto no puede financiar todos los proyectos.

A cada uno de estos se le ha caracterizado determinando su valor presente y el requerimiento (costo) asociado de capital. Cada proyecto tiene diferentes requerimientos de capital para los próximos cuatro años. En la tabla 3 se muestran el valor presente estimado, los requerimientos de capital y capital disponible para cada uno de ellos

Proyecto	Valor Presente neto estimado	Requerimientos de Capital			
		Año 1	Año 2	Año 3	Año 4
Expansión de la Planta	$180,000	30,000	40,000	40,000	30,000
Maquinaria nueva	$20,000	12,000	8,000	0	4,000
Nuevas Inves. s/productos	$72,000	30,000	20,000	20,000	20,000
Ampliación del Almacén	$80,000	20,000	40,000	40,000	10,000
Fondos de Capital Disponibles		65,000	80,000	80,000	50,000

Tabla 3

A los administradores de Fernández les gustaría desarrollar un plan de asignación de capital que muestre las erogaciones que debe hacer para cada uno de los 4 años y qué proyectos se deben financiar bajo el plan general.

PLANTEAMIENTO:

Se requieren cuatro variables en el planteamiento, puesto que se tienen cuatro proyectos por lo tanto

$x_j = 1$ indica que sí se financía,
$x_j = 0$ señala que no se financía, $j = 1,2,3,4$.

FUNCION OBJETIVO (ESTRUCTURA MATEMATICA)

Puesto que las variables X_j no tienen unidad de medición, el desarrollo de la función objetivo sólo requiere sumar los productos c_jx_j, en donde c_1 = \$180,000, c_2 = \$20,000, c_3 = 72,000 y c_4 = \$80,000. Para ello, la función objetivo se expresa de la siguiente manera:

Maximizar $z = 180\,000\,x_1 + 20\,000\,x_2 + 72\,000\,x_3 + 80\,000\,x_4$

RESTRICCIONES (ESTRUCTURA MATEMATICA)

Sujeto a

1. Requerimientos de capital, año 1:
   [(30,000 dólares) x X_1] + [(12,000 dólares) x X_2] +
   [(30,000 dólares) x X_3] + [(20,000 dólares) x X_4] $\leq$ 65,000 dólares

2. Requerimientos de capital, año 2:
   [(40,000 dólares) x X_1] + [(8,000 Dólares) x X_2] +
   [(20,000 dólares) x X_3] + [(40,000 Dólares) x X_4] $\leq$ 80,000

3. Requerimientos de capital, año 3:
   [(40,000 dólares) x X_1] + [(0,000 Dólares) x X_2] +
   [(20,000 dólares) x X_3] + [(40,000 Dólares) x X_4] $\leq$ 80,000

4. Requerimientos de capital, año 4:
   [(30,000 dólares) x X_1] + [(4,000 Dólares) x X_2] +
   [(20,000 dólares) x X_3] + [(10,000 Dólares) x X_4] $\leq$ 50,000 dólares

5. Financiamiento de proyectos, año con año:

Al seleccionar y definir las variables x_j como se señaló antes, nos aseguramos de que el valor x_1 que se obtiene para el año 1 será igual al valor x_1 para los años 2,3 y 4. Lo mismo es cierto para x_2, x_3 y x_4. Por tanto, no se requiere ninguna estructura matemática para satistacer esta restricción.

PLANTEAMIENTO MATEMÁTICO

MAXIMIZAR $Z = 180{,}000x_1 + 20{,}000x_2 + 72{,}000x_3 + 80{,}000x_4$

SUJETO A

$$30{,}000x_1 + 12{,}000x_2 + 30{,}000x_3 + 20{,}000x_4 \leq 65{,}000$$
$$40{,}000x_1 + 8{,}000x_2 + 20{,}000x_3 + 40{,}000x_4 \leq 80{,}000$$
$$40{,}000x_1 + \quad + 20{,}000x_3 + 40{,}000x_4 \leq 80{,}000$$
$$30{,}000x_1 + 4{,}000x_2 + 20{,}000x_3 + 10{,}000x_4 \leq 50{,}000$$

donde $X_j = 1,2,3.4$

1 si se elige el proyecto j

0 si no es así

En este planteamiento, al igual que antes, la función objetivo es la suma de los valores actuales netos y estimados para cada proyecto, al tiempo que cada restricción se refiere a la disponibilidad de fondos de capital para cada año.

El uso de variables binarias cero-uno en este planteamiento permite asegurar que se asignen fondos suficientes a un proyecto o se le rechace en su totalidad.

4. EL BUEN TABACO

Leonardo Moreno es el gerente de la tienda El Buen Tabaco. En la actualidad está considerando utilizar 50 pies de espacio de estantes, para varias exhibiciones de tabaco. Las exhibiciones requieren diferentes cantidades de espacio en los estantes y al señor Moreno le gustaría que le pagaran diferentes cantidades por cada una de las exhibiciones de las compañías tabacaleras. Leonardo quiere maximizar sus ingresos provenientes de las compañías tabacaleras, al mismo tiempo que no desea utilizar más de los 50 pies cuadrados que ha asignado de espacio para las exhibiciones.

En la tabla 4 se muestran los posibles pagos de cada una de las compañías tabacaleras y el espacio de estantes que se requiere. Al plantear este problema, debemos observar varias características. En primer lugar, es un problema de todo o nada, puesto que no son factibles las exhibiciones parciales. En segundo lugar, es imposible dar espacio a todas las exhibiciones; por tanto, el señor Moreno debe elegir las exhibiciones que le resulten más benéficas. Por último, **se tiene una sola función objetivo y una sola restricción.** El planteamiento debe comenzar con una definición de las variables.

Sí

$$x_j = \begin{cases} 1 & \text{si se exhibe el producto j} \\ 0 & \text{si no se exhibe el producto j} \end{cases}$$

Producto No.	Compañía tabacalera	Pagos ($)	Espacio que se requiere (pies cuadrados)
1	Tabacalera Mex	100	17
2	Cigatam	75	15
3	El Aguila	115	20
4	La Moderna	50	15
5	Tabamex	135	20

Tabla 4

PLANTEAMIENTO:

Maximizar: $Z = 100x_1 + 75x_2 + 115x_3 + 50\ x_4 + 135\ x_5$

Sujeto a

$$17x_1 + 15x_2 + 20x_3 + 15x_4 + 20x_5 \leq 50$$

cada $x_j = 0$ ó 1

5. PROBLEMAS DE ASIGNACIÓN DE CAPITAL

Si un problema tiene todas las características de un problema tipo mochila, excepto que tiene más de una restricción, se denomina con frecuencia problema de asignación de capital. El nombre proviene del hecho de que con frecuencia ocurren problemas de este tipo en la asignación de capital (o fondos para desarrollo) a diversos proyectos. La presupuestación o asignación de capital puede estar relacionada con proyectos de construcción, proyectos de investigación o cualesquiera circunstancias en las que deben asignarse fondos a un proyecto en forma de una cifra determinada. En los presupuestos de capital no es posible asignar fondos parciales a los proyectos.

La forma general de los problemas de presupuestos de capital es:

$$\text{MAXIMIZAR} : Z = \sum_{j=1}^{n} C_j X_j$$

$$\text{Sujeto a}: \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

$$i = 1,\ldots, m$$

$$x_j = 0 \quad 1,$$

$$j = 1,\ldots, n$$

Donde:
c_j = la utilidad del proyecto j
b_i = la cantidad disponible de capital de tipo i
a_{ij} = el nivel de fondos de tipo i que se requieren para apoyar el proyecto.

Los diversos tipos de capital pueden referirse ya sea a los fondos disponibles en diferentes períodos o a la disponibilidad de ciertos recursos que deben utilizarse en cantidades fijas, por ejemplo, fuerza de trabajo. Un ejemplo de un problema de asignación de capital es el problema de la Compañía Fernández.

6. MODIFICACIÓN DEL PROBLEMA DE LA DIETA

La Señora Hernández, dietista del Hospital General, es responsable de la planeación y administración de los requerimientos alimenticios de los pacientes. Las señora Hernández examina en este momento un caso de un paciente que se le ha restringido a una dieta especial que consta de dos fuentes alimenticias. Al paciente no se le ha restringido la cantidad de los dos alimentos que puede consumir: sin embargo, se deben satisfacer los siguientes requerimientos nutritivos mínimos por día: 1000 unidades de nutriente A, 2000 del nutriente B y 1500 unidades del nutriente C. Cada onza de la fuente alimenticia número 1 contiene 100 unidades de nutriente A, 400 unidades de nutriente B y 200 unidades de nutriente C, cada onza de la fuente alimenticia No. 2 contiene 200 unidades de nutriente A, 250 unidades del nutriente B y 200 unidades de nutriente C. Ambas fuentes alimenticias son algo costosas (la fuente No. 1 cuesta $6.00 por libra y la fuente No. 2 cuesta $8.00 por libra); por lo tanto, la señora Hernández desea determinar la combinación de fuentes alimenticias que arroje el menor costo y que satisfaga todos los requerimientos nutritivos.

Se requieren dos variables, puesto que se desea determinar la cantidad que debe consumirse de dos fuentes alimenticias:

x_1 = número de onzas de la fuente alimenticia no. 1 que debe consumirse diariamente.

x_2 = número de onzas de la fuente alimenticia no. 2 que debe consumirse diariamente.

En la función objetivo sólo tenemos que ajustar los coeficientes de costos. Como estos estan expresados en libras y no en onzas entonces c_1 = 6.00/16 = $0.375 por onza y c_2 = $0.50 por onza, puesto que existen 16 onzas en cada libra de las respectivas fuentes alimenticias.

Las restricciones se pueden estructurar a partir de la tabla 5

Nutriente	UNIDADES POR ONZA Fuente Alimenticia No.1	Fuente Alimenticia No.2	Unidades totales que se requieren
A	100	200	1000
B	400	250	2000
C	200	200	1500

Tabla 5

Entonces, se obtiene:

Minimizar $Z = 0.375x_1 + 0.50x_2$

Sujeto a

$$100x_1 + 200x_2 \geq 1000$$
$$400x_1 + 250x_2 \geq 2000$$
$$200x_1 + 200x_2 \geq 1500$$
$$x_1, x_2 \geq 0$$

En este problema nuestro objetivo era determinar una mezcla de costo mínimo que satisfaciera ciertos estándares dietéticos. Además que los únicos costos eran los de los ingredientes. Sin embargo, los costos de los pedidos también pueden ser un factor de importancia. En general, los costos de los pedidos pueden variar de un ingrediente a otro, dependiendo del procedimiento que se sigue para efectuar los pedidos. Supongamos que en este problema, los costos de los pedidos para las fuentes alimenticias son $5.00 y $7.50, respectivamente. Los costos no dependen de cuántos alimentos se ordenan, sino que, más bien, se incurre en el costo al realizar un pedido.

En la tabla 6 se presenta la información pertinente para este problema. El objetivo es determinar el número de onzas de cada uno de los alimentos que debe pedirse para alcanzar las unidades requeridas de cada nutriente a un costo total mínimo.

Si

x_1 = número de onzas del alimento 1 que se utilizan
x_2 = número de onzas del alimento 2 que se usan
y_1 = 1 si se utiliza el alimento 1, y 0 si no es así
y_2 = 1 si se usa el alimento 2 y 0 si no es así

	Unidades de nutrientes en la fuente alimenticia 1	Unidades de nutrientes en la fuente alimenticia 2	Unidades que se requieren de los nutrientes
Nutriente A B C	100 400 200	200 250 200	1000 2000 1500
Costo por onza de fuente alimenticia	$0.375	$0.50	
Costo de pedidos por fuente alimenticia	$5.00	$7.50	

Tabla 6

Entonces, el planteamiento es

MINIMIZAR $Z = 0.375x_1 + 0.50x_2 + 5y_1 + 7.50y_2$

SUJETO A:

$$100x_1 + 200x_2 \geq 1000$$
$$400x_1 + 250x_2 \geq 2000$$
$$200x_1 + 200x_2 \geq 1500$$
$$x_1 \leq M_1y_1$$
$$x_2 \leq M_2y_2$$
$$x_1, x_2 \geq 0$$
$$y_1, y_2 = 0 \text{ ó } 1$$

En este planteamiento, M_1 y M_2 son las cotas superiores para el uso de las fuentes alimenticias 1 y 2, respectivamente (estas cotas pueden ser parte del planteamiento del problema o, por otro lado, puede utilizarse un valor arbitrariamente grande).

Observe que las primeras tres restricciones son restricciones de programación lineal, mientras que las últimas dos sirven para asociar las variables continuas y las variables 0-1. Si una x_j es positiva, entonces se obliga a la correspondiente y_j a asumir un valor positivo para satisfacer estas restricciones. A este tipo de problemas se les denomina **problemas de cargo fijo**.

Observe que esta estructura de costos difiere de un problema estándar de PL en que existe una discontinuidad en la función objetivo ocasionada por el costo de preparación.

En general, el problema de cargo fijo puede plantearse de la siguiente manera:

$$Maximizar : Z = \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{n} f_j y_j$$

$$\text{Sujeto a} : \sum_{i=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1,\ldots, m$$

$$x_j \leq M_j y_j$$

$$x_j \geq 0 \text{ y}$$

$$y_j = 0 \quad 1,$$

$$j = 1,\ldots, n$$

donde:

x_j = número de unidades que se fabrican en el proceso j
c_j = costos unitarios en el proceso j
f_j = costos fijos o de preparación para el proceso j
b_i = requerimientos para el i-énesimo producto
a_{ij} = unidades del producto i que se fabrican en el proceso j
M_j = cota superior para xj

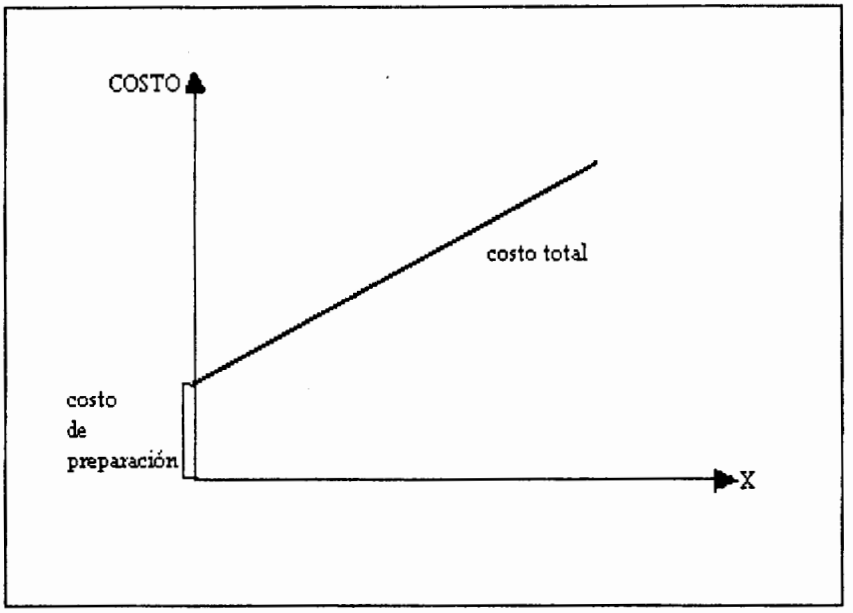


Figura 1.2

7. WATKINSVILLE FOUNDRY

Considere la situación de la Watkinsvile Foundry. Esta compañía tiene tres alternativas para ubicar un nuevo almacén que dé servicio a la parte noroeste de Estados Unidos. Existen cinco clientes importantes en esta región. En la tabla 7 se muestran los datos pertinentes de oferta, demanda y costo (los costos de transporte están expresados en $/ton.). Si se añade una columna artificial o ficticia a la tabla, puede presentarse en una tabla estándar de transporte (tabla 8).

Parece que este problema es muy similar al problema estándar de transporte excepto que deben considerarse los costos de la ubicación. En este ya no se tiene un conjunto de ofertas y demandas que deben satisfacerse utilizando el conjunto de rutas de menor costo. Más bien, se tiene que decidir entre diferentes localizaciones que pueden servir como puntos de abastecimiento y que tienen un costo asociado con cada una de ellas. Si se envía una cantidad positiva desde un almacén, entonces debe pagarse el costo total de ubicación para ese almacén.

Ubicación del almacén	Costo de ubicación	Ubicación del cliente					Capacidad (millares de Toneladas)
		Nueva York	Boston	Filadelfia	Pittsburgh	Búfalo	
Albany,	$50,000	$20	$20	$ 40	$45	$35	200
Lancaster,	30,000	30	40	15	20	45	150
Nueva York	90,000	5	25	30	35	35	300
Pronóstico de la demanda		75	50	35	75	35	

Tabla 7. Datos para el Watkinsville Foundry

Ubicación del almacén	Costo de ubicación	Ubicación del cliente						Capacidad (millares de Toneladas)
		Nueva York	Boston	Filadelfia	Pittsburgh	Búfalo	Artificial	
Albany,	$50,000	$20	$20	$ 40	$45	$35	0	200
Lancaster,	30,000	30	40	15	20	45	0	150
Nueva York	90,000	5	25	30	35	35	0	300
Pronóstico de la demanda		75	50	35	75	35	380	650

Tabla 8. Planteamiento de transporte para el Walkinsville Foundry

Utilizando esta tabla, es necesario observar que si se envía cualquier cantidad de un almacén determinado a cualquier destino que no sea el artificial, entonces debe pagarse el costo total de la ubicación. Observe también que la capacidad de

los almacenes es lo suficientemente grande para hacer que no sea necesario utilizarlos todos para satisfacer la demanda total de los clientes. De hecho, el almacén de Nueva York por sí mismo podría mejorar toda la demanda. Entonces el objetivo es minimizar la suma de los costos de transporte y de ubicación escogiendo un almacén o una combinación de almacenes y rutas entre almacenes y clientes.

Los problemas de este tipo son un caso especial del problema de cargo fijo que se analizó antes, el cual a su vez es un tipo de problema mixto en enteros. Aunque con frecuencia los problemas de ubicación de almacenes se resuelven de manera similar a los de transporte, difieren en que no deben utilizarse todas las ubicaciones disponibles. La alternativa del costo mínimo podría ser usar sólo uno de los lugares para atender todos los puntos de demanda, y en particular si hay un costo grande de ubicación asociado con cada punto de abastecimiento. A los problemas de este tipo se les denomina con frecuencia **problemas de localización de almacenes** y son un tipo de un conjunto mayor de problemas a los que se les conoce como problemas de ubicación o localización. El planteamiento matemático de los problemas de ubicación de almacenes es una combinación de los planteamientos de los problemas de cargo fijo y de los de transporte.

Sean

n = número de clientes o puntos de demanda
m = número de ubicaciones de almacenes que se están considerando
S_i = capacidad de almacén i
D_j = demanda del cliente j
c_{ij} = costo de transporte por unidad del almacén i al cliente j
f_i = costo de ubicación para el almacén i
x_{ij} = cantidad que se envía del almacén i al cliente j.

PLANTEAMIENTO:

Entonces, el modelo general para el problema de ubicación del almacén puede expresarse de la siguiente manera:

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i$$

Sujeto a

$$\sum_{i=1}^{m} x_{ij} = D_j \qquad j = 1,\ldots, n$$

$$\sum_{j=1}^{n} x_{ij} \le S_i y_i \quad i = 1,\ldots, m$$

$$x_{ij} \ge 0 \qquad \text{para todo } i, j$$

$$y_i = 0 \text{ o } 1, \quad i = 1,\ldots, m$$

La restricción (1) obliga a que y_i sea 1 si se envía cualquier cantidad desde el punto de abastecimiento i, lo cual a su vez ocasiona que sea necesario pagar el costo de la i-ésima ubicación.

Si las capacidades de todos los almacenes son lo suficientemente grandes para satisfacer todas las demandas, entonces el problema se convierte en un problema de ubicación de almacenes sin capacidades. Si no es así, al problema se le denomina con capacidades. Se han desarrollado algoritmos para resolver problemas sin capacidades que tienen hasta 100 posibles ubicaciones para los almacenes y 100 clientes.[4] Se han resuelto también problemas con capacidades hasta con 25 ubicaciones y 50 clientes.[5] Es evidente que los problemas sin capacidades son más sencillos de resolver.

8. CIGARROS MARLBORO

David Ramos trabaja como vendedor para la Compañía Marlboro y tiene como base la ciudad de Tampa. Debe viajar a otras cuatro ciudades de Florida cada semana antes de volver a casa. Las ciudades y sus distancias desde Tampa y entre sí se presentan en la tabla 9.

	Tampa	Jacksonville	Ft. Myers	Miami	Orlando
Tampa	-	194	125	268	83
Jacksonville	-	-	319	349	145
Ft. Myers	-	-	-	143	208
Miami	-	-	-	-	236

Tabla 9. Distancias entre ciudades

No deseando viajar más de lo necesario, David desea encontrar cuál es la ruta más corta para visitar cada una de las ciudades antes de volver a casa. David considera que es ineficiente visitar cualquier ciudad más de una vez a la semana, por lo que la ruta elegida debe tomar en consideración esta restricción.

El problema de David Ramos pertenece a una amplia clase de problemas de programación en enteros conocidos como problemas de planeación. Esta clase de problemas, tal como su nombre indica, se refieren a determinar la mejor secuencia o programa de actividades.

El término mejor puede referirse a costo, tiempo, distancia o algún otro criterio que se desea minimizar. Las actividades pueden ser tareas que deben llevarse a cabo en una o más máquinas, o las ciudades que el vendedor debe visitar. Esta última situación es la que se analiza aquí y se le denomina problema del agente viajero.

En general, el objetivo de los problemas del agente viajero es determinar la ruta de menor distancia que el vendedor puede seguir para salir de una ciudad base,

visitar cada una de otras diversas ciudades una y sólo una vez, y después volver a su ciudad base. A esta secuencia de visitas a ciudades se le denomina viaje. Si el vendedor visita una ciudad más de una vez o regresa a la ciudad base antes de visitar todas las demás ciudades, esto implica una derivación. La diferencia se muestra en las figuras 1.3 y 1.4.

En la figura 1.3, el vendedor realiza un viaje desde la "Base"a A-B-C-D-E-Base. En la figura 1.4 se muestra un vendedor que regresa a su base después de visitar las ciudades A, B y C y después visita las ciudades E y D, en este orden. El segundo caso implica dos derivaciones y no es una solución factible para el problema del agente viajero.

Este problema puede plantearse en forma matemática como un problema de redes con enteros 0-1, pero es mucho más común verlo planteado en forma similar al problema de asignación de personal.

Las distancias entre ciudades se colocan en una tabla en al que se considera enforma implícita que el requerimiento del viaje es parte del planteamiento.

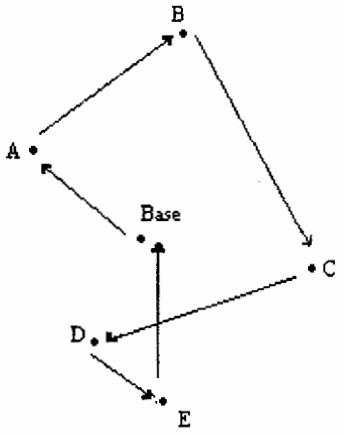


Figura 1.3

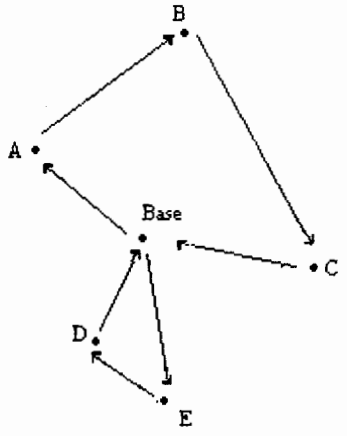


Figura 1.4

Utilizando este enfoque, el problema de David (visitar cada ciudad una y sólo una vez, al tiempo que se viaja la menor distancia posible) se ha planteado en la tabla 8. En este planteamiento, se considera que las ciudades del lado izquierdo son de donde se va a partir (orígenes), mientras que las que encabezan las columnas son los destinos. Observe que hemos utilizado una distancia de infinito para un viaje de una ciudad hacia ella misma. Esto asegura que el agente no regresa a la ciudad que acaba de visitar haciendo que esa decisión sea demasiado costosa.

Observe también que este es un problema simétrico de agente viajero, puesto que las distancias entre ciudades permanecen constantes ya sea que se provenga de una ciudad o se vaya a ella. Más adelante analizaremos un problema asimétrico.

	Tampa	Jacksonville	Ft. Myers	Miami	Orlando
Tampa	∞	194	125	268	83
Jacksonville	194	∞	319	349	145
Ft. Myers	125	319	∞	143	208
Miami	268	349	143	∞	236
Orlando	83	145	208	236	∞

Tabla 10

El problema del agente viajero es también un ejemplo de un problema de combinaciones, que recibe este nombre porque es posible que sean óptimas un gran número de combinaciones. Recuerde que existían n! soluciones para el problema de asignación de personal; para el problema del agente viajero, habrá (n-1)! soluciones para un problema con n ciudades. Se presenta esta diferencia porque la solución debe comenzar en la misma ciudad origen. Para el problema de David Ramos, habrá (5-1)! -=24 soluciones posibles. La función factorial (!) aumenta con gran rapidez al aumentar n. Por ejemplo, si n = 11 ciudades, entonces (n - 1)! = 3,628.800 soluciones posibles.

9. PINTURAS COMEX

Aunque la aplicación original del problema del agente viajero se hizo en problemas como el que se analizó antes, algunos trabajos recientes han mostrado que muchos otros problemas pueden plantearse como problemas del agente viajero. Por ejemplo, considere una compañía que utiliza una máquina mezcladora para combinar diferentes tipos de pinturas y colorantes. La máquina debe limpiarse por completo antes de realizar un trabajo diferente para evitar la contaminación de los productos, y el tiempo de limpieza depende en gran medida del orden en el que se emplean las pinturas. Por ejemplo, si se utiliza una pintura con base de látex antes de una pintura con base de aceite, el tiempo de limpieza puede ser más breve que si se hace al contrario. Si es necesario terminar un conjunto de trabajos de preparación de diferentes tipos de pintura usando la misma máquina, la secuencia en la que se llevan a cabo las tareas tendrá efecto sobre el tiempo total necesario para acabar todos los trabajos.

De esta manera, el problema se convierte en un problema del agente viajero.

Por ejemplo, considere a Pinturas Comex, que tiene un contrato para llevar a cabo varios trabajos de preparación de pinturas utilizando una máquina de alta velocidad. Los trabajos tienen etiquetas de la A a la D en la tabla 11. En la tabla se presentan los tiempos de limpieza (expresados en minutos) para todos los posibles órdenes de los trabajos. En este caso, el objetivo es minimizar la suma de los tiempos de limpieza eligiendo la mejor secuencia para las tareas. Observe que este es un ejemplo de problema asimétrico de agente viajero, puesto que las "distancias" dependen del sentido en que se viaja.

Trabajo	Trabajo			
	A	B	C	D
A	---	30	15	40
B	25	---	45	20
C	35	15	---	30
D	20	50	25	---

Tabla 11. Tiempos de limpieza

10. ARSENAL

El ARSENAL es una instalación del gobierno que lleva a cabo investigaciones sobre armamento. En estos momentos, deben realizarse cinco proyectos y existen cuatro equipos de investigadores disponibles para hacer el trabajo. En la tabla 12 se presentan los costos en que incurriría cada equipo para llevar a cabo cada uno de los proyectos (expresados en cientos de dólares) y el tiempo que se requiere (expresado en horas), junto con la cantidad de tiempo (expresada en horas) de que dispone cada equipo para trabajar.

	Trabajos					
Equipo	1	2	3	4	5	Tiempo disponible
1 costo	2	5	3	6	5	4
Tiempo	1	2	2	3	3	
2 costo	3	7	4	6	7	8
Tiempo	2	4	3	4	5	
3 costo	5	6	3	5	8	7
Tiempo	4	2	3	3	5	
4 costo	4	5	5	4	5	6
Tiempo	1	2	4	2	3	

Tabla 12

El valor de la parte superior de cada celda de la tabla se refiere al costo, y el de la inferior al tiempo requerido para realizar el trabajo. Dado que cada equipo de investigación tiene sólo una cantidad finita de tiempo disponible para estos proyectos nuevos, el tiempo (en horas) con el que cuenta cada equipo se muestra en la parte derecha de la tabla.

El problema que enfrenta ahora Maximo Benitez, Director del Arsenal, es determinar la asignación de costo mínimo de equipos a proyectos, para terminar con todos éstos. Además, tiene la restricción que impone el requerimiento de

que sólo un equipo de investigación trabaje en un proyecto. Parece que esta situación es similar a la de los problemas de asignación donde no se puede trabajar en más de un proyecto.

Recuerde que para los problemas de asignación de personal, había n trabajadores y n trabajos. Se supuso que cada trabajador podía realizar sólo una tarea y que cada tarea debía llevarla a cabo sólo un trabajador. En muchas situaciones, estas consideraciones pueden no ser realistas, dado que hay muchas circunstancias en las que un trabajador puede efectuar más de una tarea o en las que una tarea puede ser ejecutada por una combinación de trabajadores. Si eliminamos ambos supuestos, el problema se convierte en un problema estándar de transporte. Sin embargo, si sólo eliminamos el primer supuesto, que un trabajador sólo puede trabajar en una tarea, entonces tenemos lo que se conoce como un problema generalizado de asignación, el cual resulta ser un problema altamanente estructurado de programación en enteros.

PLANTEAMIENTO:

Si suponemos que existen m trabajadores, y cada uno de ellos tiene cierta cantidad de recursos (tiempo, etc.) disponibles y que existen n tareas que deben llevarse a cabo, el problema puede plantearse de la siguiente manera. Sí

b_i = cantidad de recursos para el i-ésimo trabajador
r_{ij} = cantidad del recurso del trabajador i-ésimo necesario para realizar la j-ésima tarea
c_{ij} = costo para que el trabajador i-ésimo lleve a cabo la j-ésimo tarea
entonces tenemos

$$\text{Minimizar } Z = \sum\sum c_{ij} x_{ij}$$

Sujeto a

$$\sum r_{ij} x_{ij} \leq bi \qquad i = 1,...,m$$
$$\sum x_{ij} = 1 \qquad j = 1,...,$$

x_{ij} =1 si el i-ésimo trabajador ejecuta la j-ésima tarea
0 si no es asi

El primer conjunto de restricciones asegura que no se utilizan mas recursos de los que están disponibles para cada trabajador: el segundo conjunto de restricciones asegura que cada uno de los trabajos lo lleva a cabo sólo un trabajador. Para éste problema, los costos en que incurre cada equipo al realizar cada proyecto son los valores c_{ij} de este planteamiento. Los tiempos necesarios para que cada equipo lleve a cabo cada proyecto son los valores r_{ij} y los tiempos disponibles para cada equipo son los valores b_i.

CAPÍTULO 2
COMPLEJIDAD COMPUTACIONAL

"Ver un mundo en un grano de arena
Y un cielo en una flor silvestre
Prender el infinito en la palma de tu mano
Y la Eternidad en sólo una hora"
William Blake

2.1 INTRODUCCIÓN

Para que una computadora lleve a cabo una tarea es preciso decirle qué operaciones debe realizar, es decir, debemos describir como debe realizar la tarea. Dicha descripción se llama **algoritmo**.

Un algoritmo describe el método mediante el cual se realiza una tarea. Un algoritmo consiste en una secuencia de instrucciones, las cuales, realizadas adecuadamente, dan lugar al resultado deseado.

La noción de algoritmo no es exclusiva de la computación o de las matemáticas. Existen algoritmos que describen toda clase de procesos de la vida real, por ejemplo, las recetas de cocina, las partituras musicales, etc.

En todos los casos anteriores el ejecutor o procesador de las instrucciones que realiza la tarea correspondiente es el hombre. Sin embargo, el agente que interpreta y realiza las instrucciones de un algoritmo se llama **procesador**.

Un procesador puede ser una persona, una computadora, o cualquier otro sistema electrónico o mecánico. Un procesador realiza un proceso siguiendo, o ejecutando, el algoritmo correspondiente. La ejecución de un algoritmo requiere la ejecución de cada uno de los pasos o instrucciones que lo constituyen. De aquí se desprende que una computadora no es mas que un tipo particular de procesador.

Como se ha dicho ya, para que un procesador lleve a cabo un proceso debe estar previamente provisto de un algoritmo adecuado. Por ejemplo, el cocinero debe conocer la receta, el pianista, la partitura, etc.

Si el procesador de un algoritmo es una computadora, el algoritmo se debe expresar en forma de programa. Un programa se escribe en un lenguaje de programación, y la actividad que consiste en expresar un algoritmo en un lenguaje de programación se llama **programar.**

Cada paso del algoritmo se expresa mediante una instrucción o sentencia en el programa. Por tanto, un programa consiste en una secuencia de instrucciones, cada una de las cuales especifica ciertas operaciones a realizar por la computadora.

Podría por tanto afirmarse que son más importantes los algoritmos que los lenguajes de programación e incluso las mismas computadoras, considerando que un lenguaje de programación es simplemente un medio conveniente para expresar un algoritmo y una computadora es simplemente un procesador para ejecutarlo; es decir, tanto los lenguajes de programación como las computadoras son medios para lograr un fin: ejecutar un algoritmo.

Con esto no se pretende restar importancia a las computadoras y los lenguajes de programación. Los avances en la tecnología informática permiten día a día ejecutar algoritmos más rápidamente y con más precisión y a mejor precio.

Un aspecto importante es el **diseño de algoritmos**. ¿Cómo diseñar buenos algoritmos? El diseño de buenos algoritmos requiere creatividad e ingenio y no existen, en general, reglas para diseñar algoritmos. En otras palabras, no existe un algoritmo para diseñar algoritmos.

Si existen varios algoritmos para resolver un problema ¿Cuál de ellos es el "mejor", en el sentido de que necesita menos recursos informáticos? ¿Cuáles son los mínimos recursos informáticos necesarios para llevar a cabo una tarea determinada? (es decir, ¿Qué recursos utilizará el mejor algoritmo posible para realizar dicha tarea?) ¿Se puede averiguar cuál es el mejor algoritmo? ¿Existen problemas para los cuales el mejor algoritmo posible requerirá tantos recursos que hará inviable su ejecución incluso con la computadora más grande y rápida existente?

Todas estas cuestiones se engloban para su estudio bajo el título: **Complejidad de Algoritmos.**

Este capítulo se desarrolla como sigue: en la segunda sección se presenta a través de varios ejemplos, la complejidad de los algoritmos en cuanto a tiempo y espacio requerido de almacenamiento en una computadora, en la tercera sección se habla del peor caso y el caso probabilístico para el análisis de los algoritmos. En la cuarta sección se trata el análisis asintótico de funciones donde se presentan varios resultados en cuanto a dichas funciones, para en la quinta sección medir el tiempo de ejecución de varios programas. Finalmente se presenta una sexta sección de Notas Históricas.

2.2 COMPLEJIDAD DE ALGORITMOS: TIEMPO Y ESPACIO

Como sabemos, un programa es una representación de un algoritmo en un lenguaje de programación que puede interpretar y ejecutar una computadora.

La manera de representar un algoritmo en forma de programa no es en general única. Asimismo, para resolver un problema para el cual existe un algoritmo, no es único el algoritmo que lo resuelve. Cabe por tanto preguntarse cuál es el mejor algoritmo de entre dos algoritmos dados que resuelven el mismo problema.

Una posible alternativa para contestar dicha cuestión consiste en representar los dos algoritmos mediante un lenguaje de programación, a continuación ejecutarlos en una computadora y medir el tiempo requerido por cada uno de ellos para obtener la solución de un mismo problema particular.

El **tiempo** de ejecución requerido por un algoritmo para resolver un problema es uno de los parámetros importantes en la práctica para medir la bondad de un algoritmo pues, entre otros factores, el tiempo de ejecución equivale a tiempo de utilización de la computadora y, en consecuencia, coste económico.

Es más, si el tiempo de ejecución es demasiado grande, puede suceder que el algoritmo sea en la práctica inútil, pues el tiempo necesario para su ejecución puede sobrepasar el tiempo disponible de la computadora.

Resulta evidente que el tiempo real requerido por una computadora para ejecutar un algoritmo es directamente proporcional al número de operaciones básicas elementales que la misma debe realizar en su ejecución, medir por tanto el tiempo real de ejecución equivale a medir el número de operaciones elementales realizadas. (Nosotros supondremos desde ahora que todas las operaciones básicas se ejecutan en una unidad de tiempo. Para una mayor precisión habría que distinguir los tiempos de ejecución de cada una de las distintas operaciones elementales). Por esta razón se suele llamar **tiempo** de ejecución no al tiempo real físico, sino al número de operaciones elementales realizadas.

Otro de los factores importantes, en ocasiones el decisivo, para comparar algoritmos es la cantidad de memoria de máquina requerida para almacenar los datos durante el proceso.

La cantidad de memoria utilizada por un algoritmo durante el proceso se suele llamar **espacio** requerido por el algoritmo.

Para entender la diferencia del espacio requerido por dos algoritmos que resuelven el mismo problema veamos un ejemplo.

Consideremos el siguiente problema:

EJEMPLO 2.1
Dados n números naturales en forma secuencial, es decir, dados de uno en uno con un intervalo de tiempo entre uno y otro, encuentre cuál es el máximo de todos ellos.

Veamos entonces, dos algoritmos que resuelven el mismo problema:

ALGORITMO 1

PASO 1 Almacenar los n números utilizando n variables $a_1,...,a_n$
PASO 2 Asignar $m=a_1$, $i=2$
PASO 3 Si $m>a_i$ entonces $i=i+1$ en caso contrario $m= a_i$, $i=i+1$
PASO 4 Si $i>n$ el máximo es m; FIN. En caso contrario regresar al paso 3.

ALGORITMO 2

PASO 1 Asignar el primer elemento a la variable m.
PASO 2 Asignar el siguiente elemento a la variable a.
PASO 3 Si a>m entonces m=a.
PASO 4 Mientras queden elementos volver al paso 2.
PASO 5 El máximo es m FIN.

Los dos algoritmos anteriores resuelven el problema del máximo. Sin embargo, el espacio requerido por el primero de ellos depende del valor de n. Cuanto mayor sea n mayor es el número de variables a las que hay que asignar valores en el paso 1 y, en consecuencia, mayor será la cantidad de espacio necesario.

El algoritmo 2 por el contrario utiliza sólo las variables m y a independientemente de la cantidad de números, pues este segundo algoritmo antes de recibir un nuevo número calcula el máximo de los números anteriores manteniéndolo en la variable m.

Está claro que para que el segundo algoritmo pueda ejecutarse, el intervalo de tiempo que transcurre entre la entrada de dos números debe ser mayor o a lo sumo igual al tiempo requerido para efectuar las operaciones del paso 3.

En adelante nos ocuparemos solamente del **tiempo** requerido por un algoritmo para su ejecución.

Si observamos la fórmula propuesta para medir el tiempo requerido por un algoritmo, observamos que, al no ser única la manera de representarlo mediante un programa, y al no ser única tampoco la computadora en donde se ejecuta, resulta que dicha medida será variable dependiendo fundamentalmente de los siguientes factores:

1. El lenguaje de programación elegido
2. El programa que representa el algoritmo.
3. La computadora que lo ejecuta.

En definitiva, al existir gran cantidad de lenguajes, técnicas de programación y computadoras diferentes, resulta que la forma propuesta de medir el tiempo, y en consecuencia la bondad de un algoritmo, no resulta adecuada.

Por ésta razón surge la necesidad de medir el tiempo requerido por un algoritmo independientemente de su representación y del procesador que lo ejecute.

Por otra parte, si la cantidad de datos del problema a resolver es pequeña, prácticamente cualquier algoritmo que lo resuelva lo hará en un espacio corto de tiempo, siendo, en este caso, prácticamente indiferente el elegir uno u otro algoritmo para resolverlo.
Cuando aparece la dificultad en cuanto al tiempo requerido por un algoritmo es cuando el volumen de datos del problema a resolver aumenta.

Cuando el volumen de datos es suficientemente grande es cuando un algoritmo puede realmente aventajar a otro para resolver el mismo problema.

Veamos un ejemplo:

EJEMPLO 2.2
Dada una lista $\{a_1, a_2,...,a_n\}$ de números ordenados de menor a mayor, verifique si hay algún número repetido.

Los siguientes algoritmos resuelven el problema:

ALGORITMO 3

PASO 1 i=1, j=2
PASO 2 Si $a_i = a_j$ entonces la respuesta es SI. FIN
PASO 3 Si $j < n$ entonces $i = i + 1$; $j = j + 1$ y volver al paso 2.
PASO 4 La respuesta es NO. FIN

ALGORITMO 4

PASO 1 $i = 1$; $j = 2$
PASO 2 Si $a_i = a_j$ entonces la respuesta es SI. FIN
PASO 3 Si $j < n$ entonces $j = j + 1$ y volver al paso 2
PASO 4 Si $i < n-1$ entonces $i = i + 1$; $j = i + 1$ y volver al paso 2.
PASO 5 La respuesta es NO. FIN.

Si contamos sólo las comparaciones que se efectúan en el paso 2 de ambos algoritmos, observamos que el primero compara cada número de la lista con el inmediatamente posterior, por lo que efectúa n-1 comparaciones como máximo (el peor caso será cuando no haya repeticiones o bien cuando estén repetidos los dos últimos números solamente); mientras que en el segundo algoritmo puede suceder el caso en que compare cada número con todos los que le siguen, por lo que el número de comparaciones puede llegar a ser:

$$(n - 1)+(n - 2) + (n - 3) +...+ 2 + 1 = n^2 /2 - n/2.$$

Teniendo en cuenta la tabla siguiente en la que se observan los valores que toman
$n - 1$ y $n^2/2 - n/2$ para distintos valores de n se deduce inmediatamente que el algoritmo 3 es claramente más ventajoso que el 4 8especialmente cuando el valor n es grande.)

n	n-1	$n^2/2 - n/2$
2	1	1
10	9	45
100	99	4950
1000	999	499500
100000	99999	49995000
1000000	999999	4999950000

2.3 PEOR CASO Y CASO PROBABILÍSTICO.

En el estudio anterior del tiempo de ejecución de los algoritmos hemos analizado el número de comparaciones que podrían llegar a realizarse en algún caso extremo. A estos casos en que el tiempo es el mayor posible de entre todos los casos que se pueden presentar se les llama **el peor caso.**

En ocasiones el peor caso se presenta a menudo al resolver un problema y en otras se presenta con poca frecuencia. Por esta razón tiene interés el estudio del tiempo de ejecución de un algoritmo en el caso medio o caso probabilístico. Este estudio entra dentro del campo del cálculo de probabilidades y de la estadística y, siendo de gran interés en la evaluación de los algoritmos, resulta en ocasiones sumamente complejo.

En adelante el análisis que hagamos de los algoritmos será siempre estudiando el comportamiento del mismo en el peor caso.

A pesar de la ventaja que esto supone, en ocasiones un algoritmo cuyo comportamiento en el peor caso es desastroso puede ser útil en una gran

cantidad de casos si se presenta el peor caso con una frecuencia muy pequeña.
El análisis de algoritmos se encarga del estudio del **tiempo y espacio** requerido por un algoritmo para su ejecución. Ambos parámetros, como hemos visto, pueden ser estudiados respecto del **peor caso** (o caso general) o respecto del caso probabilístico (o caso esperado). En la práctica, casi siempre es más difícil determinar el tiempo de ejecución promedio que el del peor caso, pues el análisis se hace intratable en matemáticas. Así pues, se utilizará el tiempo de ejecución del peor caso como medida principal de la complejidad del tiempo, aunque se mencionará la complejidad del caso promedio cuando pueda hacerse en forma significativa.

Tanto el tiempo como el espacio de un algoritmo son parámetros que, en general, dependen del tamaño n de los datos de entrada del algoritmo. En consecuencia son funciones T(n) y E(n) enteras de variable entera.

En general no resulta sencillo determinar el valor exacto de la función tiempo T(n) de un algoritmo. Para poder hacerlo es preciso conocer con exactitud cuáles y cuántas veces se realizan las operaciones básicas cuya ejecución requiere un tiempo constante conocido.

En definitiva, lo que aparece al analizar un algoritmo es un problema de combinatoria. No obstante, como hemos dicho, no siempre resulta fácil hacer el cálculo exacto del número de operaciones requeridas por un algoritmo. Por esta razón es por la que en muchas ocasiones el análisis de algoritmos se reduce a estudiar el **comportamiento** de la función tiempo T(n) y no cuál es su valor exacto.

2.4 ANÁLISIS ASINTÓTICO DE FUNCIONES.

En ésta sección el objeto de estudio serán las funciones reales de variable natural $f: N \rightarrow R$. La idea fundamental consiste en comparar funciones de este tipo para poder decir cuál tiene mejor comportamiento asintótico, es decir, cuál es menor cuando la variable independiente es suficientemente grande.

Si sabemos hacer esto con dos funciones podremos utilizar las funciones de tiempo de dos algoritmos y así determinar cuál de ellos tiene mejor comportamiento asintótico.

El problema que en ocasiones resulta complicado es el de hallar explícitamente la función tiempo de un algoritmo. El análisis asintótico que haremos permitirá, en ocasiones, conocer cómo se comporta una función aún sin conocer ésta.

Definición 2.1
Si f y g son dos funciones de N en R, se dice que g domina asintóticamente a f (o simplemente que g domina a f) si existen enteros $k \geq 0$ y $m \geq 0$ tales que se verifica la desigualdad: $|f(n)| \leq k\,|g(n)|$ para todo entero $n \geq m$.

Observe que si g domina a f y $g(n) \neq 0$, entonces $|f(n)/g(n)| \leq k$ para casi todos los enteros n (es decir, para todos salvo una cantidad finita). En concreto, para todos los valores $n \geq m$ se verifica dicha desigualdad.

En ésta situación, si f y g son funciones de tiempo de dos algoritmos F y G respectivamente, resulta que el algoritmo F nunca tardará mas de k veces el tiempo que tarda el algoritmo G en resolver un problema del mismo tamaño.

Por ejemplo si $f(n) = n$ y $g(n) = n^3$, se verifica que g domina a f: En efecto, si $m = 0$ y $k = 1$ se verifica que para todo $n \geq m$ se cumple la desigualdad

$$|n| \leq k\,|n^3|$$

En este caso, $|n| \leq |n^3|$ ya que $k = 1$ pues el cubo de un número natural es siempre mayor o igual que dicho número.

La definición de dominación establece una relación binaria en el conjunto de las funciones de N en R. El siguiente teorema da propiedades de dicha relación.

TEOREMA 2.1
La relación de dominación < definida por $f < g \leftrightarrow$ g domina a f, es una relación reflexiva y transitiva. La demostración de este teorema queda de ejercicio, así mismo proponemos como ejercicio el comprobar que dicha relación no es simétrica, es decir, si g domina a f, no se verifica necesariamente que f domina a g. En consecuencia, la relación de dominación no es una relación de equivalencia.

Por otra parte, esta relación tampoco es relación de orden pues no verifica la propiedad antisimétrica (compruébelo también como ejercicio).

Si f es una función de N en R denotaremos por O(f) al conjunto de todas las funciones dominadas por f.

Con notación matemática:

$$O(f) = \{ g \mid g : N \to R \text{ es aplicación y } g < f\}$$

Si una función g pertenece al conjunto O(f) se suele decir que g es una o mayúscula de F o que **g es de orden f.**

Por ejemplo, decir que el tiempo de ejecución de un programa T(n) es $O(n^2)$, significa que existen constantes enteras positivas m y k tales que para $n \geq m$ se tiene $T(n) \leq kn^2$

EJEMPLO 2.3
Suponga que T(0) = 1, T(1) = 4 y en general $T(n) = (n + 1)^2$. Entonces se observa que T(n) es $O(n^2)$ cuando m=1 y k = 4, es decir para $n \geq 1$, se tiene que $(n + 1)^2 \leq 4n^2$ que es fácil de demostrar. Observe que no se puede hacer m = 0 pues T(0) = 1 no es menor que $k0^2 = 0$ para ninguna constante k.

Se dice que T(n) es O(f(n) si existen m y k tales que $T(n) \leq kf(n)$ cuando $n \geq m$. Cuando el tiempo de ejecución de un programa es O(f(n)) se dice que tiene velocidad de crecimiento f(n)

TEOREMA 2.2
Se verifican las siguientes propiedades de los conjuntos O(f):

1. $f \in O(f)$
2. Si $f \in O(g)$ entonces $cf \in O(g)$ para todo $c \in R^+$
3. Si $f \in O(g)$ y $h \in O(g)$ entonces $f + g \in O(g)$
4. $f \in O(g)$ si y sólo si $O(f) \subset O(g)$
5. Si $f \in O(g)$ y $g \in O(f)$ entonces $O(f) = O(g)$
6. Si $f \in O(g)$ y $g \in O(h)$ entonces $f \in O(h)$

Algunos conjuntos O(f) que aparecen con frecuencia al analizar algoritmos son:

$$O(1), O(\log n), O(n), O(n \log n), O(n^2),...,O(n^p),...,O(c^n), O(n!)$$

Si la función tiempo de un algoritmo es de orden 1 se dice que dicho algoritmo tiene complejidad **constante**, si es de orden log n se dice que es **logarítmica**, si es de orden n se dice que es **lineal**, si es de orden n^p, siendo p un número natural, se dice que es **polinómica**, si es de orden c^n con c > 1 se dice que es **exponencial** y si es de orden n! se dice que es **factorial**.

Observación: Mientras no se diga lo contrario, se entenderá que los logaritmos que aparecen son en base 2.

EJEMPLO 2.4
La función $T(n) = 3n^3 + 2n^2$ es $O(n^3)$. Para comprobar esto, sean $m = 0$ y $k = 5$ entonces para $n \geq 0$, $3n^3 + 2n^2 \leq 5n^3$. También se podría decir que $T(n)$ es $O(n^4)$ pero sería una afirmación más débil que decir que $T(n)$ es $O(n^3)$

EJEMPLO 2.5
La función 3^n no es $O(2^n)$

Demostración:
Suponga que existen m y k tales que para toda $n \geq m$, se tiene $3^n \leq k\, 2^n$, entonces $k \geq (3/2)^n$ para cualquier $n \geq m$, pero $(3/2)^n$ se hace arbitrariamente grande conforme n crece y por lo tanto, ninguna constante k puede ser mayor que $(3/2)^n$ para toda n.

Cuando se dice que T(n) es O(f(n)) se sabe que f(n) es una cota superior para la velocidad de crecimiento de T(n). Para especificar una cota inferior para la velocidad de crecimiento de T(n) se usa la notación Ω (g(n)) que se lee "T(n) es omega de g(n)" lo cual significa que existe una constante c tal que $T(n) \geq cg(n)$ para un número infinito de valores de n.

EJEMPLO 2.6
Para verificar que la función $T(n) = n^3 + 2n^2$ es $\Omega(n^3)$, sea $c = 1$, entonces $T(n) \geq cn^3$ para $n = 0, 1, \ldots$

TEOREMA 2.3
Dadas las funciones de tiempo, se verifican las siguientes contenciones:

$$O(1) \subset O(\log n) \subset O(n) \subset O(n\log n) \subset O(n^2) \subset O(c^n) \subset O(n!)$$

siendo $c > 1$.

Además dichas contenciones son propias, en ninguna de ellas se da la igualdad.

Demostración:
a) $O(1) \subset O(\log n)$, en efecto si $m = 2$ y $k = 1$ se verifica que para todo $n > 2$ es $1 \leq \log n$ por lo que la función constante 1 pertenece a O(log n), por lo tanto, por el apartado 4 del teorema 2.2, $O(1) \subset O(\log n)$.

Además la contención es propia: si O(log n) estuviera contenido en O(1) se verificaría que log n pertenecería a O(1), y por lo tanto, log n estaría dominada por

la función constante 1.
Esto significa que existen $m \geq 0$ y $k \geq 0$ tales que para todo $n > m$ seria

$$|\log n| \leq k\,|1|$$

es decir:

$$\log n \leq k$$

lo cual es una contradicción, pues como

$$\lim_{n\to\infty} \log n = \infty$$

la función log n no puede estar acotada por una constante k.

b) $O(\log n) \subset O(n)$: Considerando $m = 0$ y $k = 1$ se verifica que para todo $n > m$ se cumple:

$$|\log n| \leq k|n|$$

ya que para todo $n > 0$ es $\log n < n$. Por lo tanto $\log n \in O(n)$ y por la misma razón de antes $O(\log n) \subset O(n)$

c) $O(n) \subset O(n \log n)$: si $n > 2$ se verifica que $\log n > 1$y, por tanto, $n \log n > 1.n = n$; por lo que tomando $m = 2$ y $k = 1$ se verifica que para todo $n > m$ se tiene:

$$|n| \leq k\,|n \log n|$$

Luego $n \in O(n \log n)$ y por lo tanto $O(n) \subset O(n \log n)$.

d) $O(n \log n) \subset O(n^2)$: Como $\log n < n$ si $n > 0$ resulta que $n \log n < n.\ n = n^2$. Por lo tanto, considerando $m = 0$ y $k = 1$ se verifica que para todo $n > m$ es

$$|n \log n| < k\,|n^2|$$

luego $n \log n \in O(n^2)$ y por lo tanto, $O(n \log n) \subset O(n^2)$

e) $O(n^2) \subset O(c^n)$ (si $c > 1$): Como en casos anteriores, es suficiente probar que para una n suficientemente grande se verifica la desigualdad

$$n^2 < c^n$$

lo que equivale, tomando logaritmos (tenga en cuenta que como $c > 1$ es $\log c > 0$), a la igualdad

$$\log n^2 < \log c^n$$

o bien

$$(\log n) / n < (\log c)/2$$

Como:

$$\lim_{n\to\infty} \frac{\log n}{n} = 0$$

y (log c)/2 es una constante positiva, resulta que existe un valor m tal que para todo valor n > m es

$$(\log n)/n < (\log c)/2$$

como queríamos probar.

f) $O(c^n) \subset O(n!)$: Sean k = [c] y m = max {[c^k], k} (donde [c] representa al menor entero mayor que c)

Entonces, si n > m se verifica

$$n! = n\,(n-1)\,(n-2)\,...(k+1)k(k-1)...2\,1$$

y como $k \geq [c]$ y $[c] \geq c$, se verifica

$$(n-1)(n-2)...(k+1)k \geq c^{n-k}$$

Por otra parte, como $n > c^k$, resulta

$$n! > c^k\, c^{n-k} = c^n$$

En consecuencia $n! > c^n$ si n > m y por lo tanto $O(c^n) \subset O(n!)$ como antes.

TEOREMA 2.4 Si c es una constante, se verifica:

a) $\Sigma\, c \in O(n)$

b) $\Sigma\, i \in O(n^2)$

c) $\Sigma\, i^2 \in O(n^3)$

La demostración se deja como ejercicio.

2.5 VELOCIDAD DE CRECIMIENTO Y CÁLCULO DE TIEMPO DE EJECUCIÓN DE UN PROGRAMA.

Partiremos del supuesto de que es posible evaluar programas comparando sus funciones de tiempo de ejecución sin considerar las constantes de proporcionalidad.. Es decir, un programa con tiempo de ejecución $O(n^2)$ es mejor que uno con tiempo de ejecución $O(n^3)$, sin embargo, además de los factores constantes debidos al compilador y la máquina, existe un factor constante debido a la naturaleza del programa mismo. Es posible, que con una combinación

determinada de compilador y máquina, el primer programa tarde $100n^2$ milisegundos, mientras el segundo tarda $5n^3$ milisegundos. En éste caso ¿No es preferible el segundo programa al primero?.

La respuesta a esto depende del tamaño de las entradas que se espera que procesen los programas. Para entradas de tamaño $n < 20$, el programa con tiempo de ejecución $5n^3$ será mas rápido que el de tiempo de ejecución $100n^2$. Así pues, si el programa se va a ejecutar con entradas pequeñas, será preferible el programa cuyo tiempo de ejecución es $O(n^3)$. No obstante, conforme n crece, la razón de los tiempos de ejecución que es $5n^3/100n^2 = n/20$, se hace arbitrariamente grande. Así, a medida que crece el tamaño de la entrada, el programa $O(n^3)$ requiere un tiempo significativamente mayor que $O(n^2)$. Pero si hay algunas entradas grandes en los problemas para cuya solución se están diseñando estos dos programas, será mejor optar por el programa cuyo tiempo de ejecución tiene la menor velocidad de crecimiento.

EJEMPLO 2.7

En la figura 2.1 se muestran gráficamente los tiempos de ejecución de cuatro programas de distintas complejidades de tiempo, medidas en segundos, para una combinación determinada de compilador y máquina

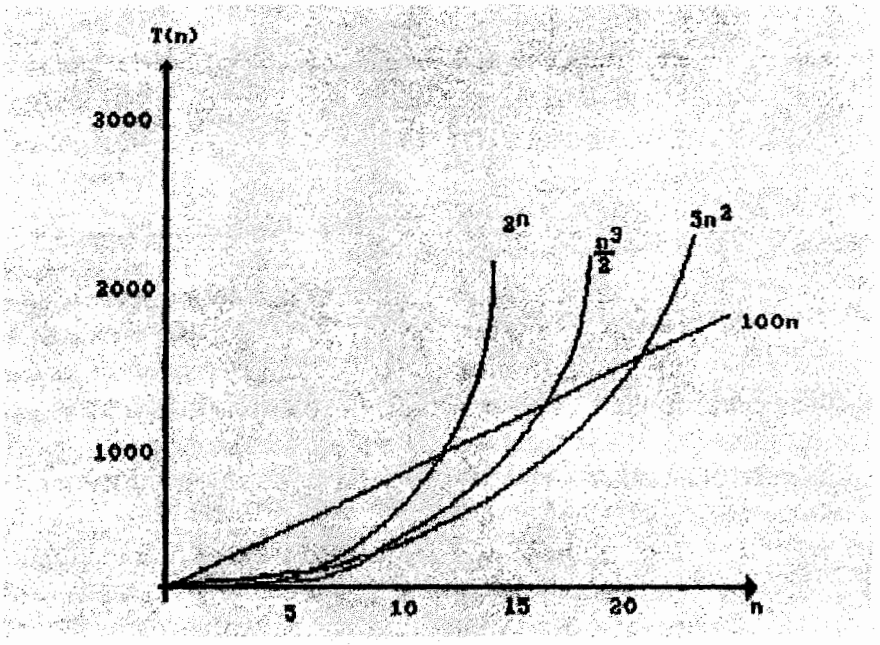


Figura 2.1

Suponga que se dispone de 1000 segundos o alrededor de 17 minutos para resolver un problema determinado ¿Qué tamaño de problema se puede resolver? Considerando estos tiempos, en la segunda columna de la siguiente tabla se muestra que los cuatro algoritmos pueden resolver problemas de un tamaño similar en 10^3 segundos. Si se adquiere una máquina que funciona diez veces mas rápido sin costo adicional, entonces es posible dedicar 10^4 segundos a la solución de problemas que antes requerían 10^3 segundos. El tamaño máximo de problema que es posible resolver ahora con los cuatro algoritmos se muestra en la tercer columna de la tabla, y la razón entre los valores de la segunda y tercer columnas se muestran en la cuarta.
Se observa que un aumento de 1000% en la velocidad de la computadora origina apenas un incremento del 30% en el tamaño del problema que se puede resolver con el programa $O(2^n)$. Los aumentos adicionales de un factor de diez en la rapidez de la computadora a partir de este punto originan aumentos porcentuales aún menores en el tamaño de los problemas. De hecho, el programa $O(2^n)$ sólo puede resolver problemas pequeños, independientemente de la rapidez de la computadora.

Tiempo de ejecución T(n)	Tamaño máximo del problema para 10^3 segundos	Tamaño máximo del problema para 10^4 segundos	Incremento en el tamaño máximo del problema
100 n	10	100	10.0
$5n^2$	14	45	3.2
$n^3/2$	12	27	2.3
2^n	10	13	1.3

En la tercer columna de la tabla se puede apreciar una superioridad evidente del programa O(n), éste permite un aumento de 1000% en la rapidez de la computadora. Y se observa que los programas $O(n^3)$ y $O(n^2)$ permiten aumentos de 230% y 320% respectivamente en el tamaño de problema, para un incremento del 1000% en la rapidez de la computadora. Estas razones se mantendrán vigentes para incrementos adicionales en la rapidez de la computadora.

A medida que la computadoras aumenten su rapidez y disminuyan su precio, también el deseo de resolver problemas mas grandes y complejos seguirá creciendo. Así la importancia del descubrimiento y el empleo de algoritmos eficientes (cuyas velocidades de crecimiento sean pequeñas) irá en aumento.

Observaciones.

1. La velocidad de crecimiento del tiempo de ejecución del peor caso no es el único criterio, ni necesariamente el más importante para evaluar un algoritmo.

2. De acuerdo con el punto anterior, si un programa sólo se va a usar algunas veces, el costo de su escritura y depuración es el dominante, de manera que el tiempo de ejecución raramente influirá en el costo total. En tal caso debe elegirse el algoritmo que sea más fácil de aplicar correctamente.

3. Un algoritmo eficiente pero complicado puede no ser apropiado porque posteriormente puede suceder que tenga que darle mantenimiento otra persona distinta del escritor. Se espera que al difundir el conocimiento de las principales técnicas de diseño de algoritmos eficientes, se podrán utilizar libremente algoritmos más complejos, pero debe considerarse la posibilidad de que un programa resulte inútil debido a que nadie entiende sus sutiles y eficientes algoritmos.

4. Existen ejemplos de algoritmos eficientes que ocupan mucho espacio para poder aplicarlos sin un almacenamiento secundario lento, lo cual puede anular la eficiencia.

5. En los algoritmos numéricos, la precisión y la estabilidad son tan importantes como la eficiencia.

TIEMPO DE EJECUCIÓN DE UN PROGRAMA

Comenzaremos con algunas reglas básicas de suma y producto en notación asintótica:

Sean $T_1(n)$ y $T_2(n)$ los tiempos de ejecución de dos fragmentos P_1 y P_2 de un programa y que $T_1(n)$ es $O(f(n))$ y $T_2(n)$ es $O(g(n))$. Entonces

$$T_1(n) + T_2(n)$$

el tiempo de ejecución de P_1 seguido de P_2 es $O(\text{máx } [f(n), g(n)])$. Esto se puede verificar observando que para algunas constantes, c_1, c_2, n_1, n_2 si $n \geq n_1$, entonces

$$T_1(n) \leq c_1 f(n)$$

y si $n \geq n_2$ entonces

$$T_2(n) \leq c_2 g(n)$$

Sea $m = \max(n_1, n_2)$, si $n \geq m$, entonces:

$$T_1(n) + T_2(n) \leq c_1 f(n) + c_2 g(n)$$

De aquí se concluye si $n \geq m$ entonces

$$T_1(n) + T_2(n) \leq (c_1 + c_2)\text{máx } [f(n), g(n)]$$

Por lo tanto : $T_1(n) + T_2(n)$ es O(máx [f(n), g(n)]).

EJEMPLO 2.8

La regla de la suma anterior se puede usar para calcular el tiempo de ejecución de una secuencia de pasos de programa, donde cada paso puede ser fragmento de un programa arbitrario con ciclos y ramificaciones. Suponga que se tienen tres pasos cuyos tiempos de ejecución son respectivamente, $O(n^2)$, $O(n^3)$ y O(n log n). Entonces el tiempo de ejecución de los dos primeros pasos ejecutados en secuencia es $O(\max(n^2, n^3))$ que es $O(n^3)$, el tiempo de ejecución de los tres juntos es $O(\max(n^3, n\log n))$ que es $O(n^3)$.

En general el tiempo de ejecución de una secuencia fija de pasos, dentro de un factor constante, es igual al tiempo de ejecución del paso con mayor tiempo de ejecución. En raras ocasiones dos pasos pueden tener tiempos de ejecución inconmensurables (ninguno es mayor que el otro, ni son iguales). Por ejemplo, puede haber pasos con tiempo de ejecución O(f(n)) y O(g(n)), donde:

$$f(n) = \begin{cases} n^4 & \text{si n es par} \\ n^2 & \text{si n es impar} \end{cases} \qquad g(n) = \begin{cases} n^2 & \text{si n es par} \\ n^3 & \text{si n es impar} \end{cases}$$

En tales casos, la regla de la suma debe aplicarse directamente, en el ejemplo, el tiempo de ejecución es O(máx [f(n), g(n)]), esto es n^4 si n es par y n^3 si n es impar.

Otra observación útil sobre la regla de la suma es que si $g(n) \leq f(n)$ para toda $n \geq m$ entonces O(f(n) + g(n)) es lo mismo que O(f(n)). Por ejemplo $O(n^2 + n)$ es lo mismo que $O(n^2)$

La regla del producto es la siguiente: Si $T_1(n)$ y $T_2(n)$ son O(f(n)) y O(g(n), respectivamente, entonces $T_1(n)T_2(n)$ es O(f(n)g(n)). Su demostración se deja como ejercicio. Según la regla del producto, O(cf(n)) significa lo mismo que O(f(n)) si c es una constante positiva cualquiera. Por ejemplo $O(n^2/2)$ es lo mismo que $O(n^2)$

Una vez que se han estudiado las herramientas necesarias para analizar el comportamiento de la función tiempo de un algoritmo, se hará el análisis de

algunos algoritmos

EJEMPLO 2.9
Encuentre el máximo de un conjunto de n números naturales $[a_1, a_2,...,a_n]$

Solución:

ALGORITMO MÁXIMO SECUENCIAL

Entrada: La lista $L = \{ a_1,...,a_n\}$
PASO 1 $m = a_1$ $i = 2$
PASO 2 Si $m < a_i$ entonces $m = a_i$
PASO 3 $i = i + 1$
PASO 4 Si $i > n$ FIN; en caso contrario volver al paso 2
Salida: el máximo es m

Análisis del Algoritmo:

En el paso 1 se realizan dos operaciones, en el paso 2, en el peor caso, otras dos, en el paso 3 una y en el paso 4, en el peor caso 2. Por lo tanto, cada vez que del paso 4 volvemos al paso 2 serán necesarias 5 operaciones; como, tras aplicar el paso 1, el ciclo 2-4 se ejecuta n-1 veces (hasta que i toma el valor n+1), el número total de operaciones que requiere este algoritmo en el peor caso será:

$$T(n) = 2 + 5(n-1)$$

y el comportamiento asintótico queda dado por la expresión:

$$T(n) \in O(n)$$

Comparamos éste algoritmo con el siguiente algoritmo recursivo

ALGORITMO MÁXIMO BINARIO

Entrada: La lista $L = \{ a_1,...,a_i, a_{i+1},...,a_j\}$ con n elementos, $i = 1$
PASO 0 Si $i = j$ entonces $m = a_1$ e ir al paso 5
PASO 1 $k= [i+j/2]$ (donde [ ] indica parte entera de $i+j/2$)
PASO 2 $L_1 = \{ a_i,...,a_k\}$, $L_2 =\{a_{k+1},...,a_j\}$
PASO 3 m_1 = MAXIMO BINARIO (L_1), m_2 = MAXIMO BINARIO (L_2)
PASO 4 Si $m_1 > m_2$ entonces $m = m_1$. En caso contrario $m = m_2$
PASO 5 FIN
Salida: el máximo es m

Análisis del Algoritmo:

Si llamamos T(n) al tiempo que requiere el algoritmo para hallar el máximo de una lista de n números, se verifica que el tiempo requerido para ejecutar el paso 3 será 2 T(n/2) (cuando n sea potencia de 2). Como el número de operaciones que requieren los pasos 0, 1, 2, 4 y 5 es constante se verificará que

$$T(n) \leq 2\,T(n/2) + c$$

donde c es una constante. Como t(1) es constante, podemos suponer que $T(1) \leq c$ por lo que se verifica que $T(n) \in O(n)$.

De esto último se concluye que el comportamiento asintótico de ambos algoritmos es el mismo. Observe que en el análisis asintótico de éste segundo algoritmo no fue necesario obtener la función de tiempo del mismo.

2.6 NOTAS HISTÓRICAS

En algún lugar, entre 400 y 300 a.c. el gran matemático griego Euclides inventó un algoritmo para encontrar el máximo común divisor (m.c.d.) entre dos números naturales. El m.c.d. de X y Y es el mayor entero que divide a ambos. Por ejemplo, el m.c.d. de 80 y 32 es 16. Los detalles del algoritmo no nos interesan por el momento, pero el algoritmo de Euclides se considera el primer algoritmo no trivial desarrollado.

La palabra **algoritmo** se deriva del nombre del matemático persa Mohammed Al-Khowârizmî quien vivió durante el siglo noveno y quién tiene el mérito de haber elaborado paso a paso reglas para sumar, restar, multiplicar y dividir números decimales ordinarios. El nombre de éste matemático en latín se convirtió en Algorismus, de donde declinó en algoritmo. Claramente Euclides y al- Khowârizmî fueron algorítmicos por excelencia.

CAPÍTULO 3
MÉTODOS DE PLANOS DE CORTE

"Primero, deben abrazarse de una ojeada todas las ideas particulares desparramadas acá y allá, y reunirlas bajo una sola idea general, para hacer comprender, por una definición exacta, el objeto que se quiere tratar."
" Después se debe saber dividir de nuevo la idea general en sus elementos, como otras tantas articulaciones naturales, guardándose, sin embargo, de mutilar ninguno de éstos elementos primitivos, como acostumbra un mal cocinero cuando trincha".
Fedro, Platón.

3.1 INTRODUCCIÓN

Como vimos en el capítulo 1 existen varios métodos para resolver problemas de programación entera, algunos de ellos se enmarcan dentro de las técnicas de planos de corte. Estos tiene una razón histórica de estudio ya que fueron los primeros en usarse para abrir la brecha de la programación entera y sirvieron para cimentar el camino que más adelante rompería con muchas barreras existentes cuando fueron publicados. Este capítulo se desarrolla como sigue: En la segunda sección se plantea en términos generales el método, en la tercera sección se desarrolla el algoritmo dual fraccionario, en la cuarta sección el algoritmo entero puro de Gomory, en la quinta el algoritmo dual fraccionario mixto para problemas enteros mixtos todas ellas con ejemplos ilustrativos desarrollados ampliamente. Y finalmente en la sexta sección las Notas Históricas.

El objetivo general de los algoritmos de planos de corte consiste en deducir desigualdades suplementarias a partir de las restricciones de variable entera que eventualmente producen un programa lineal cuya solución óptima es entera.

3.2 MÉTODOS DE PLANOS DE CORTE

Los métodos de planos de corte se aplican a problemas de la forma general

$$\begin{aligned} &\text{minimizar } c^t x \\ &\text{sujeto a} \\ &\qquad x \in S \end{aligned} \tag{3.1}$$

Donde $S \subset E^n$ es un conjunto convexo. Los problemas que incluyen minimización de una función convexa sobre un conjunto convexo, como

$$\begin{aligned} &\text{minimizar } f(y) \\ &\text{sujeto a} \\ &\qquad y \in R \end{aligned} \tag{3.2}$$

donde $R \subset E^{n-1}$ es un conjunto convexo y f, una función convexa, se puede convertir fácilmente a la forma (3.1) escribiendo (3.2) de manera equivalente como

$$\begin{aligned} &\text{minimizar } r \\ &\text{sujeto a} \\ &\qquad f(y) - r \leq 0 \end{aligned}$$

lo cual con x = (r, y) es un caso especial de (3.1)

FORMA GENERAL DEL ALGORITMO

La forma general de un algoritmo de plano cortante para el problema (3.1) es la siguiente:

Dado un politope $P_k \supset S$

PASO 1 Minimice $c^t x$ sobre P_k, y obtenga un punto x_k en P_k. Si $x_k \in S$, se para; x_k es el óptimo, en otro caso vaya al paso 2

PASO 2 Encuentre un hiperplano H_k separando el punto x_k de S, esto es, encuentre $a_k \in E^n$, $b_k \in E^1$ tales que

$$\begin{aligned} S &\subset \{ x \mid a_k^t x \leq b_k \} \\ x_k &\in \{ x \mid a_k^t x > b_k \} \end{aligned}$$

Actualice P_k para obtener P_{k+1} incluyendo la restricción $a_k^t x \leq b_k$

Este proceso se ilustra en la figura 3.1

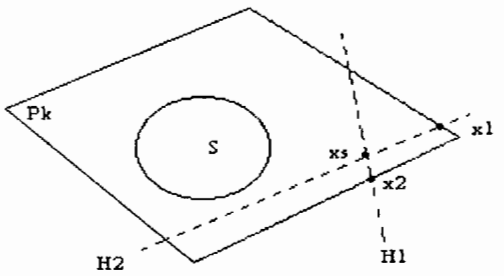


Figura 3.1

Los algoritmos específicos difieren principalmente en la manera de seleccionar el hiperplano que separa el punto actual x_k del conjunto de restricciones S. Por supuesto, esta selección es el aspecto más importante del algoritmo, pues es la profundidad del corte asociado al hiperplano de separación, o la distancia del hiperplano al punto actual, lo que determina cuánta mejora hay en la aproximación al conjunto de restricción y, por tanto, la rapidez de convergencia del método.

La desventaja de los métodos de planos de corte, como se verá mas adelante es que resultan muy ineficientes para resolver problemas enteros de tamaño medio. Estos métodos generan en cada iteración una restricción y una variable extra. Sin embargo, su ventaja es que ilustran lo que se pretende hacer con la región de factibilidad del problema entero para lograr su solución.

DUALIDAD

El algoritmo general de plano de corte se puede considerar como una aplicación ampliada de la dualidad en programación lineal, y aunque este punto de no es especialmente útil en el análisis del método, revela la conexión básica entre los métodos de planos de corte y dual. La base de éste punto de vista es el hecho de que S puede escribirse como la intersección de todos los semiespacios que lo contienen, así:

$$S = \{x \mid a_i^t x \leq b_i \; i \in I\}.$$

Donde I es un conjunto de índices (infinito) correspondiente a los semiespacios que contienen a S. Con S considerado de este modo el problema (3.1) se puede tomar como un problema de programación lineal (infinito).

Para este programa lineal existe (al menos formalmente) el problema dual

$$\begin{aligned} &\text{maximizar } \Sigma_{i \in I} \alpha_i b_i \\ &\text{sujeto a} \\ &\quad \Sigma_{i \in I} \lambda_i \mathbf{a}_i = c \\ &\quad \lambda_i \geq 0, \qquad i \in I \end{aligned}$$

Seleccionando un subconjunto finito de **I**, por ejemplo **I**, y formando

$$P = \{x \mid a^T_i x \leq b_i, \; i \in I\}.$$

Resulta un politope que contiene S. Al minimizar $\mathbf{c}^T\mathbf{x}$ sobre este politope, se generan un punto y un subconjunto correspondiente de restricciones activas I_A, Entonces el problema dual con la restricción adicional λ_i=o para $i \in I$, tendrá una solución factible, pero esta solución general, no será óptima. Así, una solución a un problema de politope corresponde a una solución factible, pero no optima del dual. Por esta razón, se puede considerar que el método de plano cortante busca la optimalidad del dual (de dimensión infinita).

A continuación se presentan algunos conceptos que se usarán continuamente en los algoritmos de planos de corte.

Se entiende por [X] al entero más grande menor que el número X, es decir

$$[X] = \text{Máx } Y \leq X, \; Y \text{ entero} \qquad (3.3)$$

Por ejemplo [6.51] = 6, [-6.51] = -7, [0.3] = 0, [-0.3] = -1.

Se considera el siguiente problema entero

$$\begin{aligned} &\text{Max } cx \\ &\text{sujeto a} \\ &Ax = b \\ &x \geq 0, \text{ entero} \end{aligned} \qquad (3.4)$$

El problema lineal correspondiente a (3.4) es

$$\begin{aligned} &\text{Max } cx \\ &\text{sujeto a} \\ &Ax = b \\ &x \geq 0 \end{aligned} \qquad (3.5)$$

Una restricción típica de (3.3) es

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}X_n + x_{N+i} = x_{Bi},$$

Donde $x_1,\ldots,x_n$ son variables no básicas; $x_{n+1},\ldots,x_{n+m}$ son las variables básicas y x_{Bi} es la variable básica correspondiente.

Esta restricción se puede escribir

$$\sum_{j=1}^{n} \lfloor a_{ij} \rfloor x_j + \sum_{j=1}^{n} (a_j - \lfloor a_{ij} \rfloor) x_j + x_{n+i} = \lfloor x_{Bi} \rfloor + (x_{Bi} - \lfloor x_{Bi} \rfloor)$$

reagrupando se tiene:

$$\sum_{j=1}^{n} (a_{ij} - \lfloor a_{ij} \rfloor) x_j + x_{n+i} - \lfloor x_{Bi} \rfloor = (x_{Bi} - \lfloor x_{Bi} \rfloor) - \sum_{j=1}^{n} (a_{ij} - \lfloor a_{ij} \rfloor) x_j$$

además se demuestra que:

$$\sum_{j=1}^{n} (a_{ij} - \lfloor a_{ij} \rfloor) x_j \geq (x_{Bi} - \lfloor x_{Bi} \rfloor)$$

es un corte.

EJEMPLO 3.1

Se deriva un corte de la siguiente restricción

$$2.8x_1 - 3.6x_2 - 0.7x_3 + 4x_4 + x_5 = 8.93$$

Donde x_5 es la variable básica. Se tiene, separando la parte entera de la fraccionaria:

$$2x_1 + 0.8x_1 - 4x_2 + 0.4x_2 - x_3 + 0.3x_3 + 4x_4 + x_5 = 8 + 0.93$$

Se divide en parte entera y fraccionaria, la parte entera es:

$$2x_1 - x_3 + 4x_4 + x_5 \leq 8$$

y la parte fraccionaria proporciona el corte:

$$0.8x_1 + 0.4x_2 + 0.3x_3 \geq 0.93$$

A continuación se analizan varios de estos métodos. Al final del análisis se presentará una comparación de los mismos, relativa al tiempo que tardan en resolver un mismo problema *tipo*, en una computadora y veremos un algoritmo que utiliza el método dual simplex y que permite números fraccionarios en los cálculos.

3.3 ALGORITMO DUAL FRACCIONARIO.

Propósito: Resolver el problema de programación entera.

Descripción

PASO 1. Se resuelve el problema entero como un problema de programación lineal olvidándose por el momento de las condiciones de variable entera.

PASO 2. Si el resultado óptimo del paso 1 o del paso 3 es entero, pare. Se ha obtenido la solución óptima del problema original. De otra manera continúe en el paso 3.

PASO 3. Seleccione el máximo $(x_{Bi} - [x_{Bi}])$ fraccionario y genere un corte. Donde $[x_{Bi}]$ es el entero más grande menor que x_{Bi}. Añada este corte como una restricción adicional, y resuelva el problema por el método dual simplex y regrese al paso 2.

$$\sum_{j=1}^{n} (a_{ij} - \lfloor a_{ij} \rfloor) x_j \geq (x_{Bi} - \lfloor x_{Bi} \rfloor)$$

EJEMPLO 3.2
Resuelva usando el algoritmo fraccionario de Gomory el siguiente problema entero

$$\max z = 5x_1 + 2x_2$$

sujeto a

$$2x_1 + 2x_2 + x_3 = 9$$
$$3x_1 + x_2 + x_4 = 11$$
$$x_i \geq 0 \quad x_i \text{ enteras}$$

PASO 1. La solución del programa lineal genera el siguiente tableau óptimo

z	x_1	x_2	x_3	x_4	x_B = LD
1	0	0	0.25	1.5	18.75
x_1	1	0	-0.25	0.5	3.25
x_2	0	1	0.75	-0.5	1.25

PASO 3. Como $x_1 = 3.25$ y $x_2 = 1.25$ no son enteros, se debe generar un corte. El máximo $x_{Bi} - [x_{Bi}]$ corresponde al primer renglón del tableau (sombreado), cuya restricción se lee

$$z + 0.25x_3 + 1.5x_4 = 18.75$$

por lo tanto el corte es:

$$(0.25)-0)x_3 + (1.5-1)x_4 \geq 18.75 - 18$$

$$\mathbf{0.25x_3 + 0.5x_4 \geq 0.75} \qquad \textbf{Primer Corte}$$

Para escribir este corte en términos de las variables no básicas se sustituyen $x_3 = 9 - 2x_1 - 2x_2$ y $x_4 = 11 - 3x_1 - x_2$ en la ecuación quedando

$$\mathbf{2x_1 - x_2 \leq 7} \qquad \textbf{Primer Corte (con } \mathbf{x_1} \textbf{ y } \mathbf{x_2}\textbf{)}$$

agregando el corte y una variable superflua S_1, el tableau queda:

z	x_1	x_2	x_3	x_4	s_1	x_B = LD
1	0	0	0.25	1.5	0	18.75
x_1	1	0	-0.25	0.5	0	3.25
x_2	0	1	0.75	-0.5	0	1.25
s_1	0	0	-0.25	-0.5	1	-0.75

Aplicando el dual simplex (2 iteraciones más) se obtiene el siguiente tableau óptimo (para el problema lineal) pero que aún no es entero.

z	x_1	x_2	x_3	x_4	s_1	x_B = LD
1	0	0.5	0	0	2.5	17.5
x_4	0	-0.5	0	1	-1.5	0.5
x_1	1	0.5	0	0	0.5	3.5
x_3	0	1	1	0	-1	2

$$x_1 = 3.5 \qquad x_3 = 2 \quad x_4 = 0.5$$

De esta tabla podemos ver que hay empate entre 17.5, 0.5 y 3.5 que serían los candidatos para ser renglón fuente ya que son fraccionarios, los empates se rompen arbitrariamente, por lo tanto el nuevo corte se toma de cualquiera de estos tres renglones, así tenemos:

$$z + 0.5x_2 + 2.5\ S_1 = 17.5.$$

De aquí el corte queda:

$$(0.5\text{-}0)x_2 + (2.5\text{-}2)S_1 \geq 17.5 - 17$$

es decir

$$\mathbf{0.5\ x_2 + 0.5\ S_1 \geq 0.5} \qquad \textbf{Segundo Corte}$$

en términos de las variables no básicas originales procedemos como en el corte anterior

$$\mathbf{x_1 - x_2 \leq 3} \qquad \textbf{Segundo Corte (con } x_1 \textbf{ y } x_2\textbf{)}$$

El nuevo tableau a resolver por medio del dual simplex es

z	x_1	x_2	x_3	x_4	s_1	s_2	x_B = LD
1	0	0.5	0	0	2.5	0	17.5
x_4	0	-0.5	0	1	-1.5	0	0.5
x_1	1	1	0	0	0.5	0	3.5
x_3	0	1	1	0	-1	0	2
s_2	0	-0.5	0	0	-0.5	1	-0.5

cuyo resultado óptimo da

z	x_1	x_2	x_3	x_4	s_1	s_2	x_B = LD
1	0	0	0	0	2	1	17
x_4	0	0	0	1	-1	-1	1
x_1	1	0	0	0	0	1	3
x_3	0	0	1	0	-2	2	1
x_2	0	1	0	0	1	-2	1

La ilustración gráfica del efecto de los cortes se ven en la figura 3.2

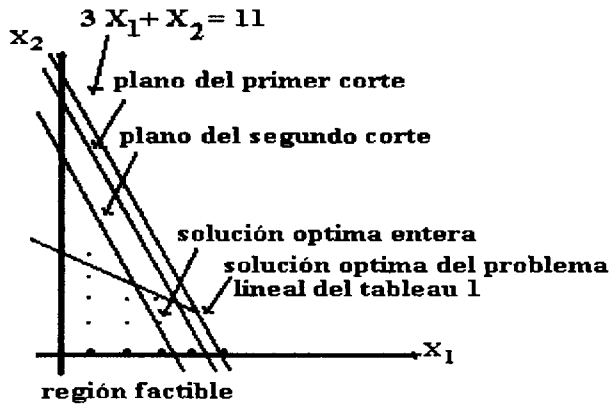


Figura 3.2

Se ve del ejemplo anterior que aunque a cada iteración le corresponde un aumento de una restricción y una variable superflua, el número de variables no básicas se mantiene constante e igual a n. Por lo tanto, nunca habrá más de n variables de holgura fuera de la base, y se puede eliminar cualquier corte cuya variable de holgura se haya convertido en básica. Esto evita el crecimiento del tableau. Se puede por lo tanto trabajar con un tableau compactado de la siguiente manera. Con cada corte, se añade la restricción de no-negatividad en la forma $-x_j \leq 0$, se omite la matriz identidad y se elimina la restricción de corte después de haber efectuado la iteración. La restricción de no negatividad se convierte en el corte actualizado únicamente cambia el valor de las variables no básicas. También, cuando una variable se convierte en no básica, se restará en la fila correspondiente una restricción de no-negatividad.

EJEMPLO 3.3

Si tomamos las columnas asociadas a las variables no básicas, x_3, x_4 del tableau 1 del ejemplo anterior, la columna correspondiente a la variable de holgura del nuevo corte S_1, la columna del lado derecho y añadiendo las filas correspondientes a las condiciones de no-negatividad de las variables no básicas x_3 y x_4 se tiene:

	z	x_1	x_4	s_1	x_B = LD
	1	0.25	1.5	0	18.75
Vectores en la base	x_2	-0.75	-0.5	0	1.25
	x_1	-0.25	0.5	0	3.25
Condiciones de no negatividad de las variables no básicas	x_3	-1	0	0	0
	x_4	0	-1	0	0
corte		-0.25	0.5	1	-0.75

Aplicando el dual simplex se pivotea en el elemento -0.25 produciendo el siguiente tableau

z	x_3	x_4	s_1	LD
1	0	1	1	18
x_2	0	-2	3	-1
x_1	0	1	-1	4
x_3	0	2	-4	3
x_4	0	-1	0	0
↑ Se elimina esta columna y fila→	1	2	-4	3

A este tableau se le elimina la última fila (el corte) si corresponde a una variable de holgura (si no, no se elimina), y la columna básica (en este caso) la x_3, al empezar la siguiente iteración.

EJEMPLO 3.4

Resuelva el siguiente modelo de programación entera, usando planos de corte

$$\text{Max } z = 2x_1 + x_2$$

Sujeto a

$$x_1 + x_2 + x_3 = 5$$
$$-x_1 + x_2 + x_4 = 0$$
$$6x_1 + 2x_2 + x_5 = 21$$

usando el simplex se tiene:

z	x_1	x_2	x_3	x_4	x_5	LD
1	0	0	0.5	0	0.25	7.75
x_2	0	1	1.5	0	-0.25	2.25
x_1	1	0	-0.5	0	0.25	2.75
x_4	0	0	-2	1	0.5	0.5

Se generan los cortes de la siguiente manera

$$z + .5x_3 + 0.25\,x_5 = 7.75$$

por lo tanto el corte es

$$(0.5-0)x_3 + 0.25-0)x_5 \geq 7.75-7$$

o bien

$$0.5x_3 + 0.25x_5 \geq 0.75$$

también se tiene que

$$-0.5x_3 + 0.25x_5 = 2.75$$

Por lo tanto el corte es:

$$(-1+0.5)x_3 + (0.25-0)x_5 \geq 2.75 - 2$$

$$\mathbf{0.5x_3 + 0.25x_5 \geq 0.75} \qquad \textbf{Primer Corte}$$

que es igual al primero entonces se escoge cualquiera de ellos y se agrega al tableau.

Otro aspecto importante es que es posible identificar geométricamente los cortes, por ejemplo el corte

$$0.5x_3 + 0.25x_5 \geq 0.75$$

es equivalente a $2x_1 + x_2 \leq 7$ ya que si $x_3 = 5 - x_1 - x_2$ y $x_5 = 21 - 6x_1 - 2x_2$ sustituimos en el corte y obtenemos el resultado anterior, que escribimos como:

$$\mathbf{2x_1 + x_2 \leq 7} \qquad \textbf{Primer Corte (con } x_1 \textbf{ y } x_2\textbf{)}$$

Agregamos el corte y resolvemos usando LINDO y obtenemos la siguiente tabla:

z	x_1	x_2	x_3	x_4	x_5	x_6	LD
1	0	0	0	0	0	1	7
x_3	0	0.5	1	0	0	-0.5	1.5
x_4	0	1.5	0	1	0	0.5	3.5
x_5	0	-1	0	0	1	-3	0
x_1	1	0.5	0	0	0	0.5	3.5

De aquí podemos ver que hay tres renglones candidatos para ser fuente del corte:

$$0.5\,x_2 + x_3 - 0.5\,x_6 = 1.5$$
$$-0.5x_2 + x_4 + 0.5\,x_6 = 3.5$$
$$0.5\,x_2 + 0.5\,x_6 = 3.5$$

Seleccionamos arbitrariamente el primero (podemos observar que en cualquiera de los tres casos obtenemos el mismo corte) y nos queda el corte:

$$\mathbf{0.5\,x_2 + 0.5\,x_6 \geq 0.5} \qquad \textbf{Segundo Corte}$$

En términos de las variables originales se tiene:

$$\mathbf{x_1 \leq 3} \qquad \textbf{Segundo Corte (con } x_1 \textbf{ y } x_2\textbf{)}$$

Obtenemos el resultado entero **z = 7** con $\mathbf{x_1 = 3}$ y $\mathbf{x_2 = 1}$.

Si analizamos esto geométricamente podemos ver el efecto de los cortes en la figura 3.3

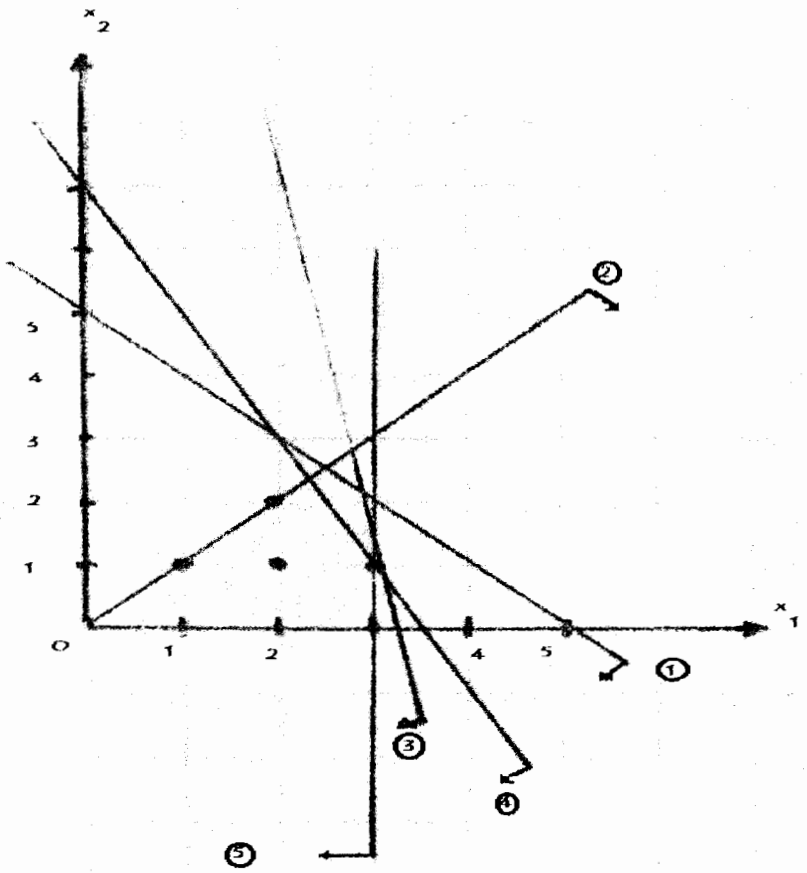


Figura 3.3

REGLAS LEXICOGRÁFICAS

Un vector es lexicográficamente positivo si la primera componente diferente de cero es positiva. Un vector lexicográficamente positivo se escribe:
Por ejemplo:

$$x = (0.3, -2.9) \succ 0$$
$$y = (108, -3, 12) \succ 0$$
$$z = (0, 0, -3, 12) \prec 0$$

Definir un vector X es lexicográficamente mayor que otro vector Y, si el vector

$$x - y \succ 0$$

Por ejemplo:

x = (0, 3, -2); y = (1, 2 ,2) entonces y -x = (1, -1, 4) $\succ$ 0 pero
x – y = (-1, 1, - 4) no es $\succ$ 0.

3.4 ALGORITMO ENTERO PURO DE GOMORY

Este método es una variante del anterior y pertenece a un proceso de optimización con punto de partida que es dual factible. Todos los coeficientes de la matriz A deben ser enteros (requisito que no se tiene en el algoritmo anterior). El método asegura que esta condición de variables enteras permanece constante durante todas las iteraciones. Los pasos a seguir son los siguientes:

Descripción

PASO 1. Se comienza con un tableau entero simplex que contiene una solución dual lexicográfica. Se escoge la x_{Bi} más negativa. Se designa ésta fila con r, y vaya al Paso 2.

PASO 2. Se escoge aquella columna no básica con $a_{ij}<0$ que lexicográficamente sea la menor. Se designa a esta columna k. Al primer elemento distinto de cero de dicha columna se le designa por $a_{pk}(.>0)$ siendo su fila correspondiente la p.

PASO 3 Para las columnas con $a_{rj} < 0$, se calcula el índice $u_j=a_{pj}/a_{pk}$ si es que a_{pj} es el primer elemento distinto de cero en la columna p. De otra manera $u_j=\infty$.

PASO 4. Se calcula λ= max $\{|a_{rj}| / u_j\}$, para $a_{rj}<0$ y $u_j \neq \infty$

PASO 5. Se derivan un corte

$$\sum_{j\,01}^{n} \left[\frac{a_{rj}}{\lambda} \right] x_j \leq \left[\frac{x_{Bi}}{\lambda} \right]$$

PASO 6. Se anexa este corte al tableau, junto con su variable de holgura correspondiente, y se aplica el método dual simplex pivoteando en algún elemento del corte Si el resultado es $x_B \geq 0$ (entero) entonces es la solución óptima. Si no, se regresa al paso 1. Para efectos de notación, el elemento del tableau en la fila i columna j se le designa a_{ij}.

EJEMPLO 3.5

Sea

$$\max z = -10x_1 - 14x_2 - 21x_3$$

Sujeto a

$$8x_1 + 11x_2 + 9x_3 \geq 12$$
$$2x_1 + 2x_2 + 7x_3 \geq 14$$
$$9x_1 + 6x_2 + 3x_3 \geq 10$$
$$x_i \geq 0 \text{ entero } i=1,2,3$$

Tableau inicial:

	x_1	x_2	x_3	LD
Z	10	14	21	0
x_4	-8	-11	-9	-12
x_5	-2	-2	-7	-14
x_6	-9	-6	-3	-10

Iteración 1.

PASO 1. Se elige $x_B = -14$. Por lo tanto r=3 (fila 3).

PASO 2. Los candidatos a_{rj} (<0) son: $a_{31} = -2$, $a_{32} = -2$, $a_{33} = -7$. De las 3 columnas la primera es la lexicográficamente más pequeña, por lo tanto k=1. El elemento $a_{pk} = a_{11} = 10$ por lo que p = 1.

PASO 3.

$$u_1 = \left[\frac{a_{11}}{a_{11}}\right] = \left[\frac{10}{10}\right] = 1$$

$$u_2 = \left[\frac{a_{12}}{a_{11}}\right] = \left[\frac{14}{10}\right] = 1$$

$$u_3 = \left[\frac{a_{13}}{a_{11}}\right] = \left[\frac{21}{10}\right] = 2$$

PASO 4.

$$\lambda = max\left\{\frac{|a_{rj}|}{u_j}\right\} = max\left\{\frac{2}{1}, \frac{2}{1}, \frac{7}{2}\right\} = 3.5$$

PASO 5. El corte se deriva de la restricción

$$\left[\frac{-2}{3.5}\right]x_1 + \left[\frac{-2}{3.5}\right]x_2 + \left[\frac{-7}{3.5}\right]x_3 \leq \left[\frac{-14}{3.5}\right]$$

y es igual a

$$-x_1 - x_2 - 2x_3 \leq -4 \qquad \textbf{Primer Corte}$$

que añadida al tableau junto con su variable superflua S_1, da:

PASO 6.

	s_1	x_1	x_2	x_3	LD
Z	0	10	14	21	0
x_4	0	-8	-11	-9	-12
x_5	0	-2	-2	-7	-14
x_6	0	-9	-6	-3	-10
corte	1	-1	-1	-2	-4

Se aplica el dual simplex pivoteando en el elemento a_{51}=-1 (donde s se refiere al renglón del corte). Como la columna x_1 se va a convertir en e_5 después de la iteración, se agrega una columna e_5 (denominada S_1 que corresponde a la primera variable superflua). El nuevo tableau es:

	s_1	x_1	x_2	x_3	LD
Z	10	0	4	1	-40
x_4	-8	0	-3	7	20
x_5	-2	0	0	-3	-6
x_6	-9	0	3	15	26
x_1	-1	1	1	2	4

Iteración 2

PASO 1

	s_1	x_2	x_3	LD
Z	10	4	1	-40
x_4	-8	-3	7	20
x_5	-2	0	-3	-6
x_6	-9	3	15	26
x_1	-1	1	2	4

Se elige x_{Bi} =-6 y por lo tanto r=3.

PASO 2

Los candidatos a_{rj} (<0) son a_{31}=-2 y a_{33}=-3. De las dos columnas con a_{rj}<0, la lexicográficamente más pequeña es la que corresponde a j= 3. Por lo tanto k = 3. El elemento a_{pk} =a_{13}=1 y p=1.

PASO 3

$$u_1 = \left[\frac{10}{1}\right] = 10, u_2 = \infty, u_3 = \left[\frac{1}{1}\right] = 1$$

PASO 4

$$\lambda = max\left\{\frac{2}{10}, \frac{3}{1}\right\} = 3$$

PASO 5. El corte se deriva de la restricción

$$\left[\frac{-2}{3}\right]s_1 + \left[\frac{-3}{3}\right]x_3 \leq \left[\frac{-6}{3}\right]$$

Y es igual a:

$$-s_1 - x_3 \leq -2 \qquad \textbf{Segundo Corte}$$

Este corte se añade al tableau quedando:

PASO 6

	s_1	x_2	x_3	LD
Z	10	4	1	-40
x_4	-8	-3	7	20
x_5	-2	0	-3	-6
x_6	-9	3	15	26
x_1	-1	1	2	4
x_3	-1	0	-1	-2

Se aplica el método dual simplex pivoteando en el elemento $a_{53} = -1$, y se agrega una columna unitaria a la que se denomina s_2 (que es también la variable superflua del nuevo corte), antes de la iteración se tiene la siguiente tabla:

	s_2	s_1	x_2	x_3	LD
Z	0	10	4	1	-40
x_4	0	-8	-3	7	20
x_5	0	-2	0	-3	-6
x_6	0	-9	3	15	26
x_1	0	-1	1	2	4
x_3	1	-1	0	-1	-2

Después de la iteración se tiene:

	s_2	s_1	x_2	x_3	LD
Z	1	9	4	0	-42
x_4	7	-15	-3	0	6
x_5	-3	1	0	0	0
x_6	15	-24	3	0	-4
x_1	2	-3	1	0	0
x_3	-1	1	0	1	2

Se elimina la columna x_3.

Iteración 3

PASO 1

Se elige x_{Bi} =-4 y por lo tanto r = 4.

PASO 2

El candidato a_{rj} (<0) es a_{42}=-24. Por lo tanto k=2. El elemento a_{pk}=9 y p=1.

PASO 3

$$u_1 = \infty, u_2 = \left[\frac{9}{9}\right] = 1, u_3 = \infty$$

PASO 4

$$\lambda = max\left\{\frac{24}{1}\right\} = 24$$

PASO 5.
El corte se deriva de la restricción

$$\left[\frac{15}{24}\right]s_2 + \left[\frac{-24}{24}\right]s_1 + \left[\frac{3}{24}\right]x_2 \leq \left[\frac{-4}{24}\right]$$

Y es igual a:

$-s_1 \leq -1$ **Tercer Corte**

Este corte se añade al tableau quedando:

PASO 6

	s_3	s_2	s_1	x_2	LD
Z	0	1	9	1	-42
x_4	0	7	-15	-3	6
x_5	0	-3	1	0	0
x_6	0	15	-24	3	-4
x_1	0	2	-3	1	0
x_3	0	-1	1	0	2
s_3	1	0	-1	0	-1

Se aplica el método dual simplex y se tiene la siguiente tabla:

Iteración 4.

PASO 1.

	s_3	s_2	s_1	x_2	LD
Z	9	1	0	4	-51
x_4	-15	7	0	-3	21
x_5	1	-3	0	0	-1
x_6	-24	15	0	3	20
x_1	-3	2	0	1	3
x_3	1	-1	0	0	1
s_3	-1	0	1	0	1

Se elige $x_B = -1$ por lo que r=3

PASO 2

El candidato a_{rj} (<0) es a_{32}=-3. Por lo tanto k=2. El elemento a_{pk}=1 y p=1.

PASO 3

$$u_1 = \infty, u_2 = \left[\frac{1}{1}\right] = 1, u_3 = \infty$$

PASO 4

$$\lambda = max\left\{\frac{3}{1}\right\} = 3$$

PASO 5 El corte se deriva de la restricción

$$\left[\frac{1}{3}\right]s_3 + \left[\frac{-3}{3}\right]s_2 + \left[\frac{0}{3}\right]x_2 \leq \left[\frac{-1}{3}\right]$$

Y es igual a:

$-s_2 \leq -1$ **Cuarto Corte**

Este corte se añade al tableau quedando:

Paso 6

	s_4	s_3	s_2	x_2	LD
Z	0	9	1	4	-51
x_4	0	-15	7	-3	21
x_5	0	1	-3	0	-1
x_6	0	-24	15	3	20
x_1	0	-3	2	1	3
x_3	0	1	-1	0	1
s_4	1	0	-1	0	-1

Se aplica el método dual simplex

	s_4	s_3	s_2	x_2	LD
Z	-1	9	0	4	-52
x_4	7	-15	0	-3	14
x_5	1	1	0	0	2
x_6	-3	-24	0	3	5
x_1	15	-3	0	1	1
x_3	2	1	0	0	2
s_4	-1	0	1	0	1

Como todas las componentes de la columna del lado derecho (a excepción de la función objetivo), son no-negativas y enteras, la solución óptima es:

$x_1=1$, $x_2=0$, $x3=2$, $x_4=14$, $x_5=2$, $x_6=5$,
$s_4=1$

Y el máximo de la función objetivo $z^* = -52$, por lo cual el mínimo es 52.

EJEMPLO 3.6
Sea

$$\max z = -4x_1 - 5x_2$$

sujeto a
$- x_1 - 4x_2 \leq -5$
$-3x_1 - 2x_2 \leq -7$
$x_1, x_2 \geq 0 \quad x_1, x_2 \in Z$

Iteración 1

Paso 1 r = 3

Paso 2

Los candidatos a_{rj} (<0) son: $a_{31} = -3$ $a_{32} = -2$ de las 2 columnas la primera es lexicográficamente más pequeña por lo tanto k=1. El elemento $a_{pk}=a_{11}=4$ por lo que p=1.

Paso 3

$$u_1 = \left\lceil \frac{a_{11}}{a_{11}} \right\rceil = \left\lceil \frac{4}{4} \right\rceil = 1$$

$$u_2 = \left\lceil \frac{a_{12}}{a_{11}} \right\rceil = \left\lceil \frac{5}{4} \right\rceil = 1$$

Paso 4

$$\lambda = max\left\{ \frac{2}{1}, \frac{3}{1} \right\} = 3$$

Paso 5

El corte se deriva de la restricción y es igual a

$$\mathbf{x_5 = - x_1 - x_2 \geq 3} \qquad \textbf{Primer Corte}$$

Que añadido al tableau junto con su variable superflua S_1 da:

	s_1	x_1	x_2	LD
z	0	4	5	0
x_3	0	-1	-4	-5
x_4	0	-3	-2	-7
x_5	1	-1	-1	-3

Se aplica el dual simplex pivoteando en $a_{41} = -1$ como x_1 se va a convertir en básica (denominada S_1) queda

	x_5	x_2	LD
z	4	1	-12
x_3	-1	-3	-2
x_4	-3	1	2
x_1	-1	1	3

Iteración 2.

Como 1<4 la columna 2 es el pivote k, $u_2=1$ y u_1 es el mayor entero para el cual $u_1 = 4$.

El corte es: $-s_1 - x_2 \leq -1$ (verifíquelo)o bien

$$\mathbf{x_6 = x_1 -+ 2x_2 \geq 4} \qquad \textbf{Segundo Corte}$$

Se agrega al tableau

	s_1	x_5	x_2	LD
z	0	4	1	-12
x_3	0	-1	-3	-2
x_4	0	-3	1	2
x_1	0	-1	1	3
x_6	-1	0	-1	-1

se pivotea en $a_{53} = -1$ y se obtiene la tabla óptima

	x_5	x_6	LD
z	3	1	-13
x_3	2	-3	1
x_2	1	-1	1
x_1	-2	1	2
x_4	-4	1	1

$x_5 \geq 0$ y $x_6 \geq 0$
$x_5 = -3 + x_1 + x_2 \geq 0$
$x_6 = -4 + x_1 + 2x_2 \geq 0$

3.5 ALGORITMO DUAL FRACCIONARIO MIXTO.

El siguiente algoritmo para un programa entero mixto, es una extensión directa del algoritmo de planos de corte fraccionario. Como el anterior utiliza el método dual simplex y permite números fraccionarios en sus cálculos y no requiere que el primer tableau sea entero puro.

Paso 1 Resuelva el programa entero mixto como uno lineal. Si no es factible entonces el programa entero mixto tampoco lo es, termine. Si la solución óptima es entera en las variables restringidas a ser enteras el programa está resuelto, termine. De otra forma vaya al paso 2.

Paso 2 Desde un renglón correspondiente a una variable restringida entera que no tiene un valor entero, derive una nueva desigualdad que corte el punto óptimo actual pero que no elimine cualquier solución mixta entera. Aumente esta nueva desigualdad al final del tableau simplex que entonces exhibe una infactibilidad primal (la variable de holgura asociada con el nuevo renglón será negativa).

Paso 3. Reoptimice usando el método dual simplex lexicográfico si el nuevo programa lineal no satisface (el dual no será acotado), el problema entero mixto no tiene solución, termine. Si la nueva solución óptima es entera en las variables restringidas a enteras, el programa entero mixto está resuelto, termine. De otra forma vaya al paso 2.

LA FORMA DEL CORTE

En el tableau simplex óptimo $a_{0j} \geq 0$ (j=1,...,n) $a_{i0} \geq 0$ (i=1,...,n) $a_{i0} \geq 0$ (i=1,...,n+m) y α_j □ 0 (mayor lexicográficamente i=1,...,n). Más aún, si a_{i0} (i=0,1,..,l) es entero, el programa entero mixto está resuelto. De otra forma, hay un renglón fuente v ($0 \leq v \leq l$) con x_v como sigue y a_{v0} fraccionario.

$$x_v = a_{v0} + \sum_{j=1}^{n} a_{vj}(-x_j)$$

Entonces el k-ésimo corte de Gomory tiene la forma:

$$x_{n+m+k} = -f_{v0} + \sum_{j=1}^{n} -g_{vj}(-x_j) \geq 0$$

Y $f_{vj} = a_{vj} - [a_{vj}]$ (j=0,1,...,n). Observe que $0 < f_{v0} < 1$.

$$g_{vj} = \begin{cases} a_{vj} & \text{si } a_{vj} \geq 0 \text{ y } x_j \text{ es una variable continua} \\ \dfrac{f_{v0}}{f_{v0}-1} a_{vj} & \text{si } a_{vj} < 0 \text{ y } x_j \text{ es una variable continua} \\ f_{vj} & \text{si } f_{vj} \leq f_{v0} \text{ y } x_j \text{ es una variable entera} \\ \dfrac{f_{v0}}{1-f_{v0}}(1-f_{vj}) & \text{si } f_{vj} > f_{v0} \text{ y } x_j \text{ es una variable entera} \end{cases}$$

Así cuando la desigualdad se agrega al final del tableau, se introduce una infactibilidad primal. La variable de Gomory $x_{m+n+k} = -f_{v0} < 0$.

Ejemplo 3.7

Sea

$$\max x_0 = -4x_1 - 5x_2$$

sujeto a

$-x_1 - 4x_2 \leq -5$ (x_3)

$-3x_1 - 2x_2 \leq -7$ (x_4)

$x_1, x_2 \geq 0$

$x_0, x_1 \in \wedge$

El programa lineal se puede escribir en la forma estándar aumentando las variables de holgura x_3 y x_4. Este programa ya fue resuelto y el primer tableau óptimo usando LINDO está dado por (pasando las expresiones de decimales).

TABLA 1

			j = 2	j = 1	j = 0	
x_0	x_1	x_2	x_3	x_4	LD	
x_0	0	0	7/10	11/10	-112/10	v =0
x_1	1	0	2/10	-4/10	18/10	v = 1
x_2	0	1	-3/10	1/10	8/10	v = 2

Seleccionamos el renglón correspondiente a x_1 como renglón fuente, puede ser el renglón de x_0 o el de x_1 ya que requerimos que cualquiera de esas dos variables sea entera. Si usamos el criterio de la variable cuyo lado derecho tiene la fracción mayor entonces corresponde a x_1

derivación del corte

$$f_{10} = \frac{18}{10} - \left[\frac{18}{10}\right] = \frac{8}{10}$$

$$g_{11} = \frac{8/10}{8/10 - 1}\left(\frac{-4}{10}\right)$$

$$g_{12} = \frac{2}{10}$$

$$x_5 = \frac{-8}{10} - \frac{16}{10}(-x_4) - \frac{2}{10}(-x_3) \geq 0$$

Podemos rescribir el corte como:

$0.2\, x_3 + 1.6\, x_4 \geq 0.8$ **Primer Corte**

En términos de las variables no básicas x_1 y x_2 queda:

$-5\, x_1 - 4\, x_2 \leq -13$ **Primer Corte (con x_1 y x_2)**

Se agrega este corte a la tabla quedando

TABLA 2

	x_1	x_2	x_3	x_4	x_5	LD
x_0	0	0	9/16	0	11/16	-188/16
x_1	1	0	4/16	0	-4/16	2
x_2	0	1	-5/16	0	1/16	12/16
x_4	0	0	2/16	1	-10/16	8/16

De aquí vemos que el único renglón fuente que podemos seleccionar es el renglón v = 0, ya que para $x_1 = 2$ se tiene que la solución ya es entera

$$f_{00} = \frac{-188}{16} - \left[\frac{188}{16}\right] = \frac{4}{16}$$

$$g_{01} = \frac{11}{16}$$

$$g_{02} = \frac{9}{16}$$

$$x_6 = \frac{-4}{16} - \frac{11}{16}(-x_5) - \frac{9}{16}(-x_3) \geq 0$$

Podemos rescribir el corte como:

$$\mathbf{9\,x_3 + 11\,x_5 \geq 4} \qquad \textbf{Segundo Corte}$$

En términos de las variables no básicas x_1 y x_2 queda:

$$\mathbf{4\,x_1 + 5\,x_2 \geq 12} \qquad \textbf{Segundo Corte (con } x_1 \textbf{ y } x_2\textbf{)}$$

TABLA 3

	x_1	x_2	x_3	x_4	x_5	$-x_6$	1
x_0	0	0	0	0	0	1	-12
x_1	1	0	5/11	0	0	-4/11	23/11
x_2	0	1	-4/11	0	0	1/11	8/11
x_4	0	0	7/11	1	0	-10/11	8/11
x_5	0	0	9/11	0	1	-16/11	4/11

Ahora $x_0 = 12$ tiene un valor entero y x_1 no, por lo que lo tomamos como renglón fuente para el corte

$$f_{10} = \frac{23}{11} - \left[\frac{23}{11}\right] = \frac{1}{11}$$

$$g_{11} = \frac{1/11}{1/11 - 1}\left(\frac{-4}{11}\right) = \frac{4}{110}$$

$$g_{12} = \frac{5}{11}$$

$$x_7 = \frac{-1}{11} - \frac{4}{110}(-x_6) - \frac{5}{11}(-x_3) \geq 0$$

Podemos rescribir el corte como:

$$\mathbf{5\,x_3 + 0.4\,x_6 \geq 1} \qquad \textbf{Tercer Corte}$$

En términos de las variables no básicas x_1 y x_2 queda:

$$\mathbf{(33/5)\,x_1 + 22\,x_2 \geq 154/5} \qquad \textbf{Tercer Corte (con } x_1 \textbf{ y } x_2\textbf{)}$$

Finalmente se obtiene el resultado en la siguiente tabla óptima:

TABLA 4

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	LD
x_0	0	0	0	0	0	0.062	0	-12
x_1	1	0	0	0	0	-0.025	1	2
x_2	0	1	0	0	0	0.007	-0.8	0.8
x_3	0	0	1	0	0	0.005	-2.2	0.2
x_4	0	0	0	1	0	-0.060	1.4	0.6
x_5	0	0	0	0	1	-0.095	1.8	0.2

La tabla 4 despliega la solución entera mixta óptima x_0=-12, x_1=2, x_2 = 4/5. Expresando las desigualdades aumentadas en términos de las variables originales no básicas x_1 y x_2 se obtiene:

$$x_5 = -13 + 5x_1 + 4x_2 \geq 0$$
$$x_6 = -12 + 4x_1 + 5x_2 \geq 0$$
$$x_7 = -14/5 + (3/5)x_1 + 2x_2 \geq 0$$

Observación: Si en este ejemplo se procede con un paquete como LINDO para efectuar todos estos cálculos, es muy probable que se cicle y no se obtenga la solución exacta sino aproximada debido a errores de redondeo en el paquete, ya que trabaja con decimales.

Observe que las variables de holgura de Gomory no son necesariamente una combinación lineal entera de las variables originales no básicas. Entonces no tienen que ser enteras en toda solución entera mixta. Por lo tanto, como esas restricciones (renglones) no son elegibles para generar desigualdades es razonable adaptar la estrategia de suprimir a la variable de holgura de Gomory y definir el renglón inmediatamente después de que las variables de holgura correspondientes a la columna pivote.

Los algoritmos de planos de corte compuestos usan ambos, los cortes fraccionarios y los enteros puros. Por ejemplo, se podría usar primero el método simplex para obtener -al menos cercanamente- una solución óptima continua. Entonces desde el punto en que estemos, dependerá de cual es aplicable el fraccional o el entero puro.

3.6 NOTAS HISTORICAS

La idea de generar restricciones fue propuesta por Dantzig, Fulkerson y Johnson en 1954 en su trabajo del problema del agente viajero, posteriormente en 1957 Markowitz y Manne lo propusieron también. Sin embargo, no fue hasta 1958 que Gomory desarrolló el primer algoritmo de planos de corte aplicable a cualquier programa entero. Poco tiempo después junto en Beale generalizó sus resultados al caso mixto entero. En 1960 Gomory propuso un segundo algoritmo de planos de corte para el programa entero que requiere sólo sumas y restas en sus cálculos (una técnica "completamente entera"). Todos estos métodos mantienen programas líneas que son duales factibles y a menudo se clasifican como algoritmos de planos de corte duales.

Glover y Young desarrollaron algoritmos de planos de corte para el programa entero que mantienen problemas lineales que son primales factibles. Como las soluciones enteras que se producen sucesivamente sin primales factibles, la técnica se conoce como algoritmo de planos de corte primales.

CAPÍTULO 4
MÉTODOS DE RAMIFICACIÓN Y ACOTAMIENTO

> "El segundo (principio), era la división de cada una de las dificultades con que tropieza la inteligencia al investigar la verdad, en tantas partes como fuera necesario para resolverlas. Y el último (principio), consistía en hacer enumeraciones tan completas y generales, que me dieran la seguridad de no haber incurrido en ninguna omisión"
> Descartes, *Discurso del Método*, Vol II.

4.1 INTRODUCCIÓN

El procedimiento de ramificación y acotamiento (R-A) proporciona una metodología de búsqueda de la solución óptima en un problema de optimización discreta. En el método de R-A, el conjunto de soluciones factibles se parte en subconjuntos más simples (Esto es lo que se debería hacer en la práctica, sí uno está buscando por ejemplo una aguja en un pajar. El pajar es grande y es imposible buscar en todo simultáneamente, así que se puede dividir visualmente en lado derecho e izquierdo y seleccionar uno de ellos para buscar la aguja, manteniendo el otro lado en espera para buscar después si es necesario). A continuación un subconjunto prometedor se selecciona y se hace un esfuerzo para encontrar la mejor solución factible y se almacena esta información, en algunos casos podría darse por terminado el método. Se parte nuevamente el subconjunto en dos o más subconjuntos más simples (bajo la operación denominada ramificación) y se repite el mismo proceso

Este capítulo se desarrolla como sigue: En la segunda sección se describe el problema de optimización discreta y los elementos básicos del método de R-A. La tercera sección presenta el algoritmo básico de ramificación y acotamiento y el árbol de búsqueda que registra el desarrollo del algoritmo. La tercera sección describe las distintas opciones que hay para ramificar y desarrollar la búsqueda de la solución óptima. En la cuarta sección se presentan las estrategias operativas del algoritmo básico, en la quinta estrategias de búsqueda, para finalmente en la sexta sección desarrollar tres algoritmos básicos para programación entera mixta. Como siempre se incluye una sección de Notas Históricas al final del capítulo.

4.2 DESCRIPCIÓN CONCEPTUAL DE RAMIFICACIÓN Y ACOTAMIENTO

El método de Ramificación y Acotamiento (R-A), surgido hace tres décadas, ha resultado ser una de las mejores herramientas prácticas para la solución de problemas de optimización discreta. Su atractivo radica en la habilidad de eliminar implícitamente grupos grandes de soluciones potenciales sin evaluarlos explícitamente. A semejanza de la programación dinámica, la técnica de ramificación y acotamiento es una estrategia, y como tal se debe combinar con la estructura del problema específico que se desea resolver, para así formar un algoritmo de solución adecuado.

Hay muchos problemas de optimización discreta para los cuales los métodos "directos" o no existen o son ineficaces. Los problemas pueden ser tales que las restricciones o función(es) objetivo(s) son no convexas, o que todos o algunos de los valores están restringidos a valores discretos. La técnica de ramificación y acotamiento nos posibilita resolver problemas "difíciles" usando los métodos existentes para la solución de problemas "fáciles".

Suponga que se desea resolver un problema "difícil" de optimización discreta, como el siguiente:

$$
\begin{aligned}
&\text{Minimizar } C^0(x)\\
&\text{Sujeto a}\\
&g_1^0(x) \geq 0,\\
&g_2^0(x) \geq 0,\\
&\qquad \ldots\\
&g_m^0(x) \geq 0\\
&x \in X^0,
\end{aligned}
$$

donde el conjunto X^0 denota el dominio factible de optimización por ejemplo el octante positivo del espacio n- euclideano, x denota un vector $(x_1, x_2,\ldots,x_n)$ y $g_i^0(x)$ son las funciones que restringen el problema. Un vector x que satisfaga las restricciones y se encuentre en el dominio de optimización se dice que es una solución factible, y uno que además minimiza la función objetivo es una solución óptima.

Existen cuatro razones importantes para aceptar ampliamente los algoritmos de ramificación y acotamiento en la programación entera:

a) El método es conceptualmente simple y fácil de entender
b) Es fácilmente adaptable a un amplio rango de situaciones problemáticas
c) Es fácilmente adaptable para su implantación computacional y
d) Los métodos alternativos usualmente no están disponibles.

La estrategia de aplicación de método de R-A se basa en la premisa de que el problema que se va a resolver tenga los siguientes atributos:

a) **Naturaleza Combinatoria**. Un problema combinatorio tiene como mínimo las siguientes propiedades:
a.1) Se da un conjunto finito de objetos
a.2) Cada objeto puede tomar un cierto rango de atributos.
a.3) Se desarrolla una solución al problema fijando los valores de atributos para todos los objetos
a.4) Solo se permiten ciertas combinaciones de los valores de atributos.

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

b.1) Construir un conjunto finito y contable que contenga todas las soluciones del problema (esto se sigue de 1);

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

c) **Racionalidad**. Un problema racional es tal que:

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

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

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

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

d.2) Se hace un mínimo esfuerzo para obtener dicha estimación;

d.3) La estimación es razonablemente cercana al valor actual.

El concepto de R-A explota éstas propiedades para poder implícita y explícitamente construir un árbol que describa todas las operaciones realizadas en el problema, y efectuar una búsqueda para encontrar la mejor solución.

4.3 EL ÁRBOL DE BÚSQUEDA.

El proceso de ramificación del problema de optimización discreta puede representarse por medio de un árbol de búsqueda. Cada nodo del árbol está asociado a un subconjunto de T, conjunto de todas las soluciones originales del problema. Sea T_i un subconjunto de T, esto es, una colección de soluciones del problema. La Ramificación es el proceso de partir un subconjunto T_i en m subconjuntos disjuntos $v_1, v_2,...,v_m$ donde

$$v_1 \cup v2 \cup ... \cup v_m = T_i,$$
$$v_i \cap v_j = \phi, \quad i \neq j.$$

O bien, la unión de ellos es T_i y la intersección entre ellos es vacía.

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

En la figura 4.1 se ilustra un árbol que exhibe estos conceptos para el caso m = 2. El nodo N =1 corresponde a T, el conjunto de todas las soluciones. Los nodos 2 y 3 están asociados con los conjuntos ajenos T_1 y T_2 de T. Note que no hay ramificaciones desde los nodos 3, 4 y 5.

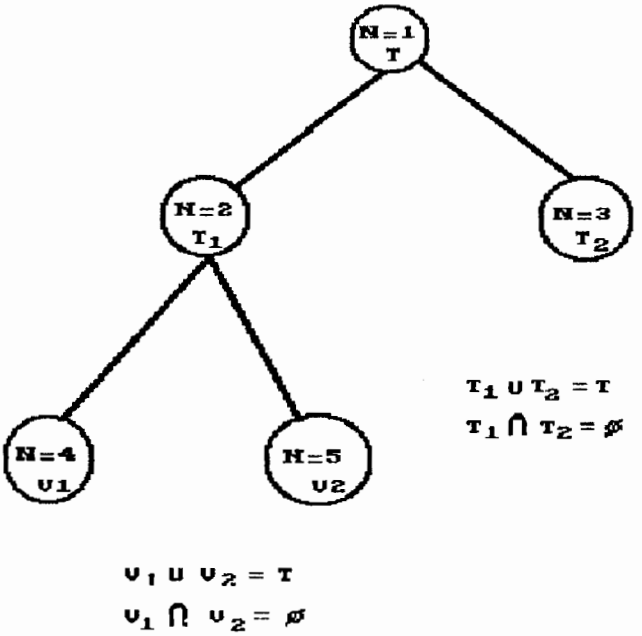


Figura 4.1

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

$T_i(N)$ es el subconjunto T_i de T representado por el nodo N.

Un Nodo Intermedio es un nodo N en el cual no ha habido ramificación.

Un Nodo Final es un nodo intermedio N para el cual Ti(N) consiste de una solución única.

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

Una forma conceptual y al mismo tiempo general de describir el método de Ramificación y Acotamiento es como sigue.

EL ALGORITMO BÁSICO DE RAMIFICACIÓN Y ACOTAMIENTO

PROPÓSITO: Encontrar la solución óptima de un problema de optimización discreta

DESCRIPCIÓN

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

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

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

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

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

Si el objetivo es maximizar solo se invierten los papeles de las cotas y se invierten las direcciones de las desigualdades.

CONVERGENCIA DEL ALGORTIMO.

La operación de ramificación del algoritmo garantiza una solución óptima en un número finito de iteraciones. Como T es finito el proceso de ramificación conducirá eventualmente a una solución T_i, como nodo final, a menos que se detenga previamente. El proceso de ramificación produce una partición de T para la cual cada subconjunto Ti, obedece, a la definición de partición. Así todos lós nodos finales posibles se pueden generar y obtenerse así la solución óptima. Las características de la ramificación prometen una enumeración completa para el algoritmo de R-A a menos que se pueda encontrar una solución óptima antes de hacerlo. Las características de acotamiento del algoritmo de R-A proporcionan la posibilidad de reconocer una solución óptima antes de completar la enumeración.

4.4 ESTRATEGIAS OPERATIVAS EN EL ALGORITMO BÁSICO.

Empezaremos primero con las distintas opciones para efectuar la operación de ramificación. Identificamos tres formas distintas de ramificar: ordenada, inmediata del sucesor y ramificación del sucesor. En aras de hacer más sencilla la exposición planteamos un problema de permutación y a través de él se exponen las distintas técnicas de ramificación.

Se desea ordenar un conjunto de n artículos $A=\{a_1,...,a_n\}$, esto es, buscamos un ordenamiento que sea óptimo de los n artículos bajo condiciones dadas. El conjunto de soluciones factibles consiste de vectores n-dimensionales.

$$F = \{x \mid x_i \in A \text{ y } x_i \neq x_j \text{ si } i \neq j\},$$

Donde el vector x, representa un arreglo de los n artículos. La exigencia de que x_i sea diferente a x_j si i es distinto de j, se debe a que un mismo artículo a_i, no puede ocupar dos lugares en un mismo arreglo.

A continuación se discuten las posibles reglas de ramificación para el caso en que n=4 y A={a, b, c, d}

RAMIFICACIÓN ORDENADA

Vamos a llamar X al conjunto de soluciones factibles, que contiene los artículos ya seleccionados, llamaremos R(X) al rango de X, es el artículo que vamos a seleccionar para introducir a la solución factible que se construye, llamaremos nivel de X, al número de artículos seleccionados, así sea $R=\{r_1,...,r_p\}$ un subconjunto de los índices de los artículos {1,2,...,n} que contiene p elementos distintos, que se han introducido al arreglo.

Un nodo X de un árbol de búsqueda se determina al imponer las siguientes restricciones.

$$x_{r1}=a_{j1},\ x_{r2}=a_{j2},...,x_{rp}=a_{jp}$$

Para algún conjunto $\{a_{j1},...,a_{jp}\}$ de p artículos distintos (los seleccionados), de donde, la siguiente selección de X la haremos sobre los restantes n-p artículos a ordenar, lo que nos genera una partición de X en n-p subconjuntos ajenos, al especificar el artículo que tenga orden r(X) para alguna r(X) que no está en R, seleccionamos también el nuevo conjunto de soluciones factibles a partir.

La regla de ramificación ordenada, nos especifica las reglas explícitas de selección, para asignar un rango a X: r(X), de esta forma r(X) se considera sólo en función del nivel de X (donde p = nivel de X). Por ejemplo: r(X) = p +1 consiste en seleccionar primero, el artículo que irá en primer lugar del arreglo, después el del

segundo lugar y así sucesivamente, de ésta forma, si se tiene un conjunto X={x_1, x_2, x_3, x_4} donde la regla de selección es "mayor que" se tiene x_1 x_2 x_3 x_4. Si la regla fuera r(X) = n-p entonces empezaríamos por seleccionar el último del arreglo, y terminar en el primero, en el ejemplo anterior la solución es la misma pero la selección se hace considerando primero a x_4, después a x_3 y así hasta x_1.

La figura 4.2 muestra un árbol de enumeración completa, para n=4 y A={a, b, c, d} generado por ramificación ordenada.

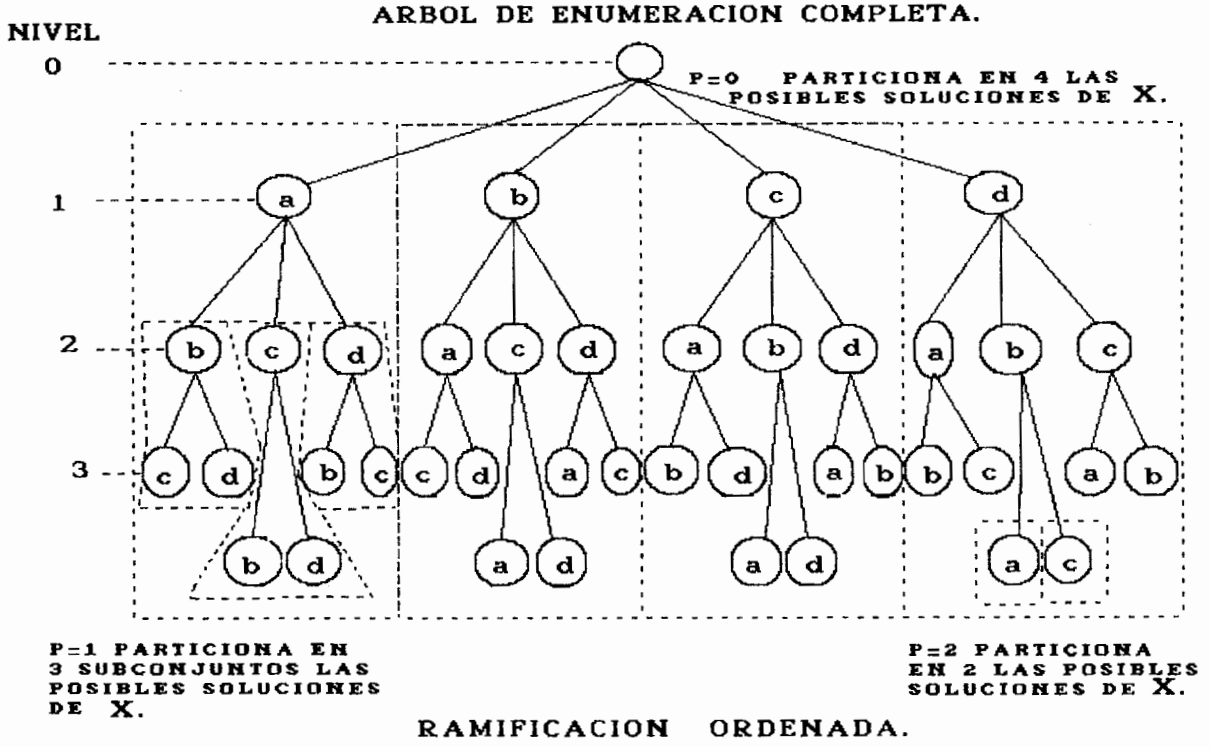


figura 4.2

Otra regla de selección consiste en ir insertando en forma ordenada cada uno de los artículos, de la manera siguiente, al primer artículo le asignamos su lugar en el arreglo solución. Por ejemplo considere los nodos a, b, c y d en los niveles 0, 1 2 y 3 respectivamente. Entonces Z corresponde a la solución

(b, d, a, c) si r(a)=3, r(b)=1 r(c)=4, r(d)=2
(b, a, d, c) si r(a)=2, r(b)=1 r(c)=4, r(d)=3
(a, b, c, d) si r(X)= (Nivel de X)+1 para toda X
(d, c, b, a) si r(X)= n-(Nivel de X) para toda X.

RAMIFICACIÓN INMEDIATA DEL SUCESOR

Sea X un nodo del árbol especificado por p(= nivel de X) relaciones de la forma $r_j = r_i+1$ que puede interpretarse como el renglón a_j ordenado inmediatamente después del renglón a_i). Entonces X se puede particionar en dos conjuntos:

$$X(kl) = \{x \mid x \in X \text{ y } r_l=r_k+1\}$$
$$X(\underline{kl}) = X-X(kl) = \{x \mid x \in X \text{ y } r_l \neq r_k+1\}$$

Se requiere una regla de selección para decidir el par de índices k y l. La figura 4.3 da un ejemplo de un árbol de enumeración completa para n=4 y A={a, b ,c, d}.

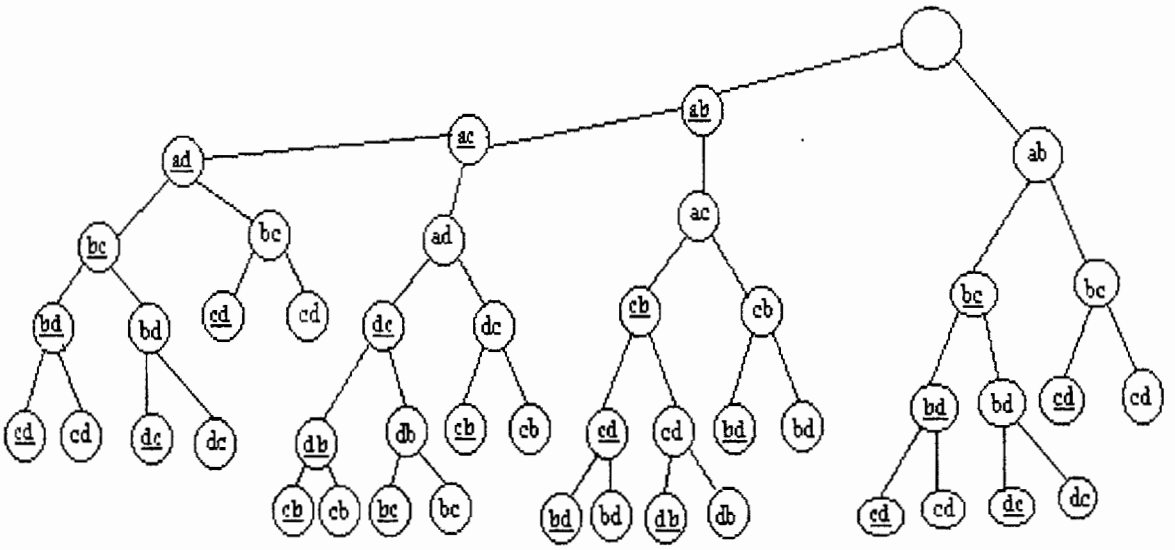


Figura 4.3

RAMIFICACIÓN DEL SUCESOR

Sea X un nodo del árbol especificado por p relaciones $r_j > r_i$ (que se interpretan como un renglón a_j ordenado posteriormente a un renglón a_i). Donde X puede particionarse en dos conjuntos

$$X(kl) = \{x \mid x \in X \text{ y } x_l > x_k\}$$
$$X(lk) = X - X(kl) = \{x \mid x \in X \text{ y } x_k > x_l\}$$

Nuevamente se requiere de una regla de selección para escoger la pareja (k,l), $k \neq l$. La figura 4.4 da un ejemplo de un árbol de enumeración completa.

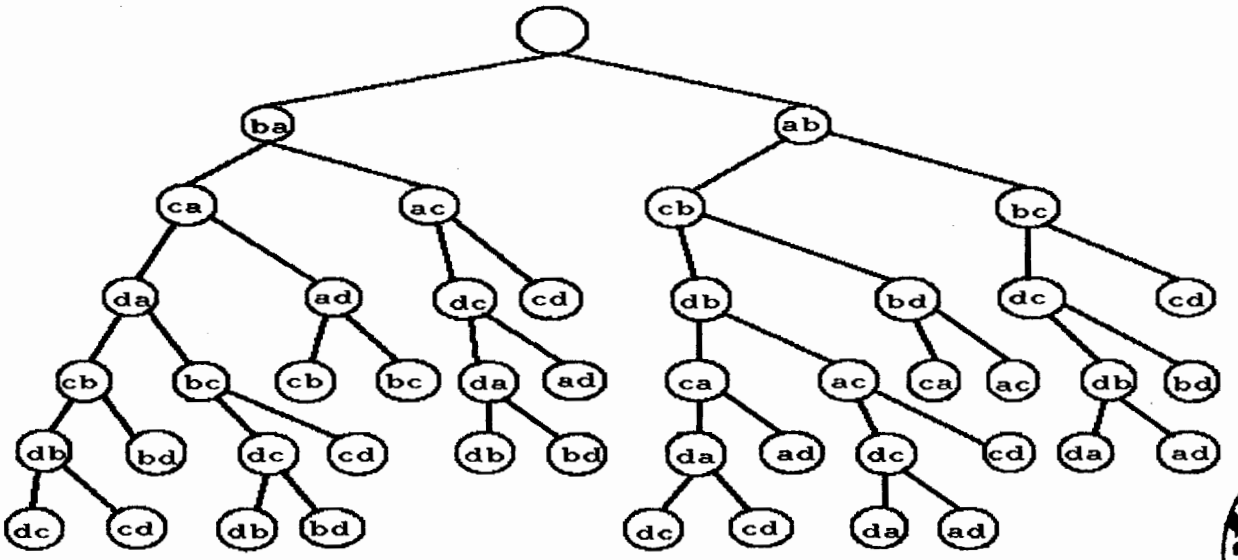


Figura 4.4

ACOTAMIENTO

Un árbol de búsqueda puede formarse con una pequeña parte de un árbol de enumeración completa (en el cual no se descarta ningún nodo, se consideran todos los posibles), debido al tamaño tan grande de éste. Para poder hacer esto escogemos una regla de acotamiento adecuada. Una función de acotamiento es una función $g:2^T \rightarrow R$ que asigna un número real g(x) a cada subconjunto T tal que si X está contenido en T entonces la cota g(X) satisface

$$g(X) \leq f(x) \text{ para toda } x \in X \cap F$$

(Note que las cotas son todas cotas inferiores ya que estamos suponiendo un problema de minimización). Podemos suponer también que

$$g(\{x\})=f(x) \text{ para toda } x \in F$$

Cotas para un problema dado P pueden obtenerse relajando alguna de las restricciones. Esto da el problema relajado Q cuyo conjunto factible F_Q contiene al conjunto factible F_P del problema original P. Sea X un subconjunto de T entonces

$$g(X) = \min_{y \in X \cap F_Q} f(y) \leq \min_{y \in X \cap F_P} f(y) \leq f(x) \quad \forall\, x \in X \cap F$$

Claramente g es una función de acotamiento inferior.

Una vez establecidas las reglas de ramificación y acotamiento, describiremos a continuación las principales estrategias de desarrollo del algoritmo de R-A, las más importantes son:

4.5 ESTRATEGIAS DE BÚSQUEDA EN RAMIFICACIÓN Y ACOTAMIENTO

BÚSQUEDA A PRIMER PROFUNDIDAD O REGLA DE LA COTA MÁS RECIENTE O TAMBIÉN LIFO

Es la estrategia de ramificación del nodo activo más reciente (un nodo Y es activo dependiendo de que f(S) > g(Y) donde S es la solución actual, Y es algún nodo del conjunto total de soluciones), esto es, se escoge el nodo de máxima profundidad entre aquellos nodos que aún no se ramifican. Si hay más de uno, entonces se selecciona aquél que corresponda al valor de cota inferior.

Esta regla logra que el árbol llegue a un nodo terminal rápidamente, aunque se podrían requerir más cálculos computacionales, se requiere un mínimo de memoria, ya que es suficiente para acumular problemas correspondientes a cada nodo en una ruta de la raíz del árbol al nodo terminal.

En la terminología de Ramificación y Acotamiento esta estrategia de desarrollo se conoce como búsqueda en la lista de nodos activos, bajo el criterio que el último en entrar es el primero en salir.

PRIMER ALCANCE O REGLA DE LA MENOR COTA O TAMBIÉN FIFO

Esta regla también se conoce como método o regla de separación progresiva y evaluación. Aquí se escoge sistemáticamente el nodo con el valor de la menor cota observando que es en éste en el que se tiene intuitivamente mayor oportunidad de que contenga una solución óptima entre sus sucesores. Sin embargo, con este método corremos el riesgo de tener que explorar una gran porción del árbol antes de encontrar una solución. Nuevamente, la primera solución que se encuentra es en general mejor (esto es, de menor costo) que la que se encuentra por la estrategia de búsqueda de "primer profundidad".

Esta regla se usa también en la programación dinámica y es la opuesta a la de primera profundidad en la estrategia de ramificar un "mínimo nodo activo recientemente creado". También se conoce como FIFO porque si se tiene una lista de nodos activos es el primero en entrar y el primero en salir.

COSTO UNIFORME

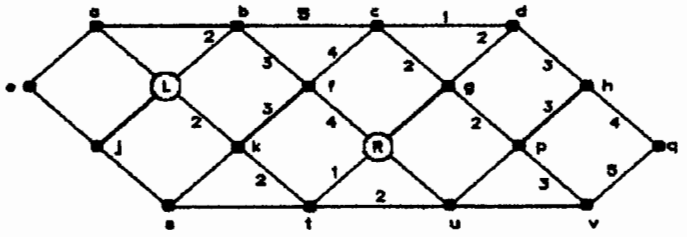
Esta regla consiste en escoger un nodo X del árbol parcialmente expandido, para el cual:

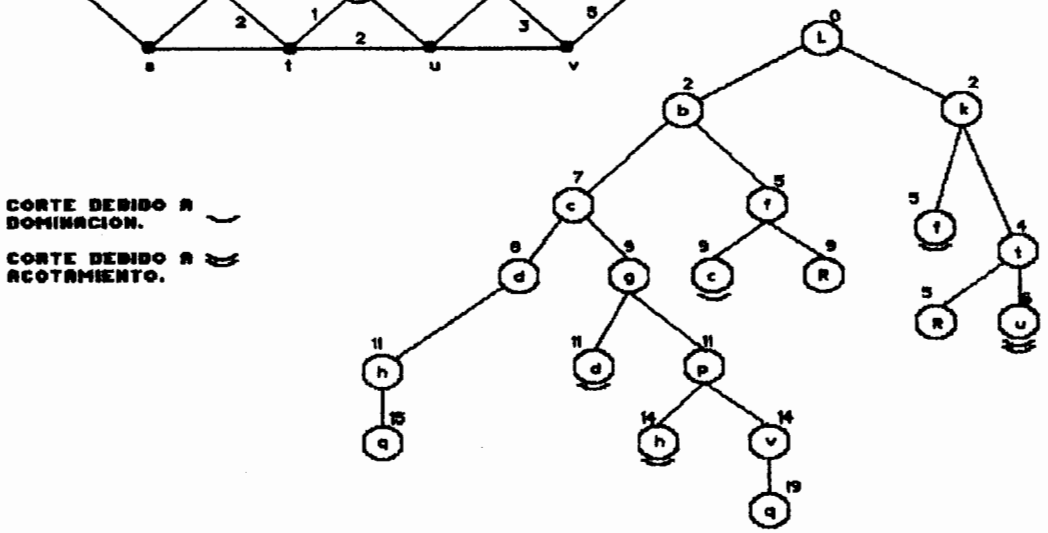
$$g(X) \leq g(Y) \quad \text{para todo nodo Y,}$$

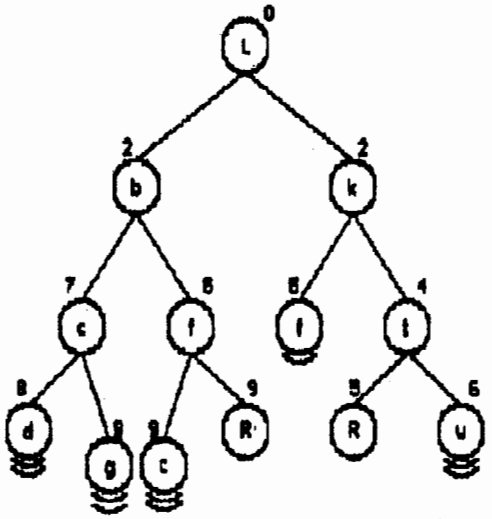
Los empates se resuelven por alguna regla subsidiaria. Esta es la ramificación de la menor cota o estrategia de costo uniforme (UC) y es computacionalmente óptima en el sentido de que efectivamente requiere un mínimo número de nodos a

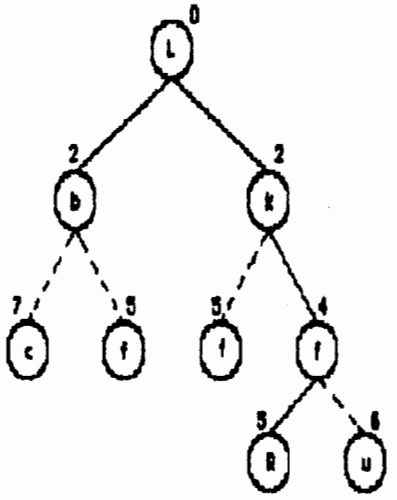
ser desarrollados. Aunque produce árboles pequeños, puede caer en dejar un mínimo tiempo de cómputo y también sufre el defecto de producir primero una solución factible en o cerca del fin de los cálculos.

Se describen en la figura 4.5 tres estrategias para el problema de búsqueda para encontrar la trayectoria mas corta de L a R. de la gráfica a). En b) se usa primer profundidad, en c) primer amplitud y en d) costo uniforme.









4.6 ALGORITMOS DE R-A PARA PROGRAMACIÓN ENTERA MIXTA.

El primer método que se presenta es el que desarrollaron A. H. Land y A.G. Doig, en 1960. Posteriormente R. J. Dakin propone un nuevo algoritmo, basado en el anterior pero con modificaciones que lo hacen más rápido y más tarde Norman J. Driebeek y Stanley Zionts presentan un nuevo método que plantea una alternativa distinta a las dos anteriores. Cabe señalar que aunque todos estos algoritmos se refieren a problemas de programación lineal mixta se pueden aplicar a problemas de programación entera pura.

EL ALGORITMO DE LAND Y DOIG.

Considere el problema:

maximizar $z = cx + dy$	(4.1)
sujeto a	
$Ax + Dy \leq b$	(4.2)
x vector columna con componentes en Z^+	(4.3)
$x \geq 0$	(4.3')
$y \geq 0$	(4.4)

Donde z es un escalar; b es un vector columna de m componentes c^t, x son vectores columna de n_1 renglones; d^t, y son vectores columna de $(n-n_1)$ componentes; A es una matriz de orden m x n_1; y D es una matriz de orden mx$(n-n_1)$. Una solución factible al problema es aquella que satisface (4.2), (4.3) y (4.4).

A continuación se desarrolla un ejemplo sencillo del Algoritmo de Land y Doig: Se aplica el algoritmo a un problema de programación lineal en dos variables sin restricción de variable discreta.

En la figura 4.6 se puede ver que la función (representada por la familia de rectas

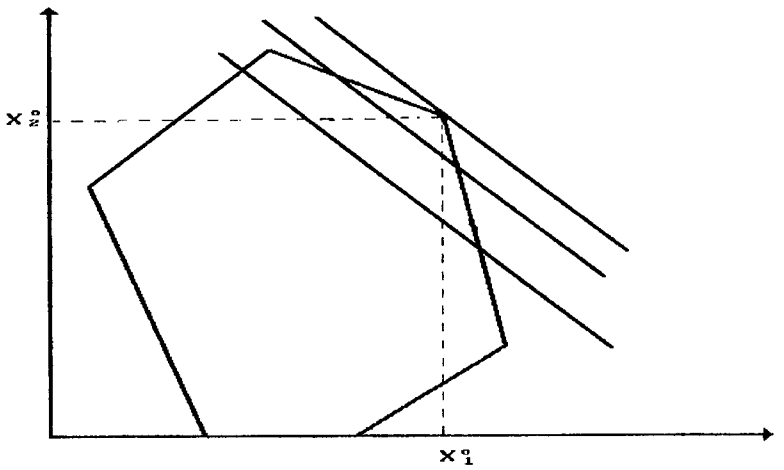


figura 4.6

paralelas) alcanza el máximo en (x_1^0, x_2^0).

Las restricciones de variable discreta limitan el conjunto de soluciones factibles a los puntos que están en el conjunto original para los cuales ambas coordenadas son enteras; en la figura 4.7 se muestra el conjunto completo de soluciones factibles para el problema de programación discreta el cual consta de 10 puntos, es fácil ver que en éste caso la solución máxima es $x_1 = 3$, $x_2 = 3$ (indicada por el punto E). El procedimiento para llegar a ella se puede describir como "mover hacia abajo la función lineal hasta encontrar un punto entero".

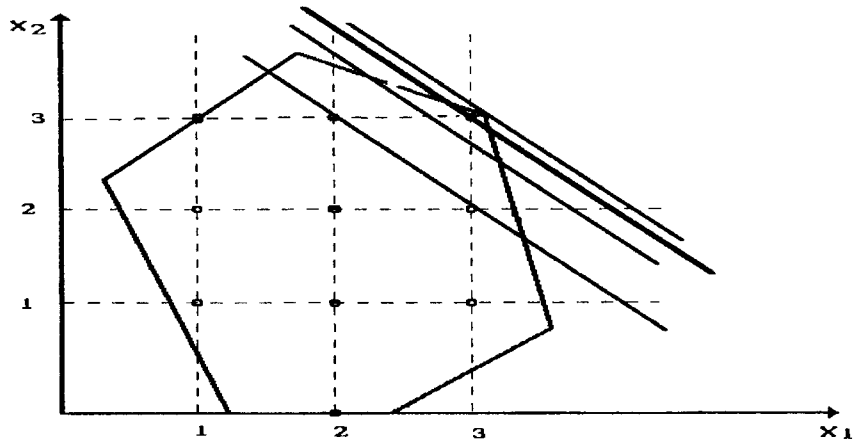


figura 4.7

DESCRIPCIÓN DEL MÉTODO

Este método usa sistemáticamente paralelas en el hiperplano en términos de reducir la función objetivo hasta encontrar un punto en el conjunto de soluciones factibles con coordenadas enteras, en las dimensiones que se especifican. Esto es en principio, ya que en la práctica no se tiene la facultad de "ver" un hiperplano en un espacio de n dimensiones, en términos de determinar si contiene un punto cuyas n coordenadas son enteras. Los métodos númericos sólo pueden examinar un punto a la vez, pero al pasar el hiperplano de una región a otra no se debe omitir ningún punto entero.

Si la cota superior de la función en cualquier etapa es z^K entonces se ha demostrado que hay una solución no discreta con un valor mas alto del maximando que z^K

Considere el conjunto convexo de soluciones de un problema de programación lineal y z^K cualquier valor factible de la función objetivo, que está únicamente asociado con una posición definida del hiperplano, que en general corta al conjunto convexo de n dimensiones y en el caso especial del valor óptimo toca al conjunto convexo de n-1 dimensiones.

Por ejemplo la figura 4.8 representa una intersección con un conjunto tridimensional.

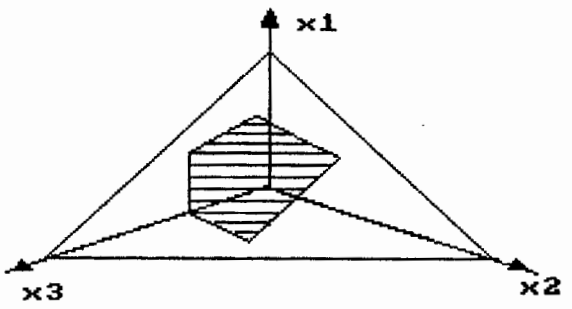


figura 4.8

Los puntos x_1, x_2 y x_3 son las intercepciones del hiperplano con los tres ejes respectivamente y el área sombreada representa el conjunto convexo que está dentro de las restricciones del problema de programación lineal ordinario. En tal caso el conjunto de n-1 dimensiones tendría un mínimo y un máximo para cada variable como se muestra en la figura 4.9

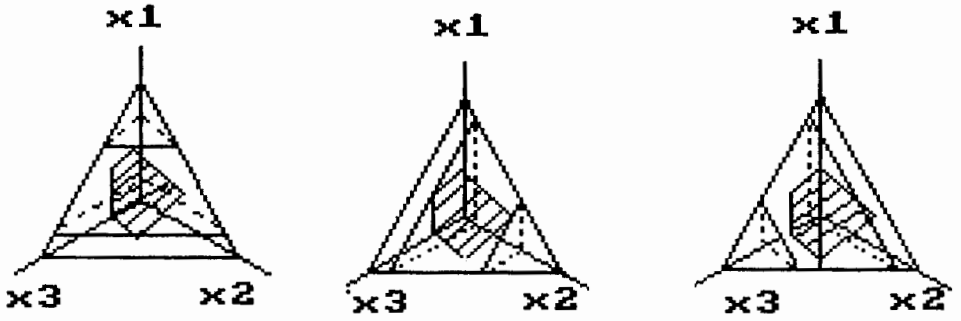


figura 4.9

Si $z^k = z^0$ esto es, el hiperplano es el asociado con la solución óptima del problema de programación lineal ordinario, este conjunto convexo consistiría normalmente de un único punto (a menos que tuviera soluciones óptimas múltiples). Es también cierto que, para cualquier valor del maximando (posición del hiperplano) hay un valor mínimo y uno máximo (que puede coincidir) para cada variable, consistente con las restricciones del problema de programación lineal ordinario.

Conociendo un valor particular de z, los valores mínimo y máximo para cada variable se conocen así como un posible valor entero de cada variable para la posición del hiperplano. Si no hay al menos un posible valor entero para cada variable entonces se puede decir inmediatamente que no hay solución al problema en el valor del maximando. Desafortunadamente lo inverso no es cierto, el hecho de que cada variable tome un valor entero en algún punto o puntos en el conjunto no es condición suficiente para que haya una intersección de esas coordenadas enteras en el conjunto y por lo tanto una solución factible.

ALGORITMO LAND Y DOIG

PROPÓSITO: Resolver el problema de programación entera mixta.

DESCRIPCIÓN

La solución se construirá en forma de gráfica de árbol cuyos nodos son elementos de uno de los conjuntos S_j j=0, 1,...,n_1 o del conjunto S^*. Los pasos de ésta construcción son

PASO 0 El primer nodo del árbol es la solución óptima de P(0), que está dada por la etiqueta z^0. Si ésta solución satisface también P(n_1), terminamos se tiene la solución óptima.

PASO 1 Si z^0 no es el valor óptimo se traza un arco a cada uno de los dos puntos en S_1 (o posiblemente en S^*). Se evalúa z en esos puntos, si están en S_1. Si están en S^* es inmediato que no necesitan calcularse.

PASO 2 Si los nodos z^0, z^1,...,z^{k-1} ya se han etiquetado, el nodo más alto no etiquetado (de acuerdo al valor de z que no está en S^*) se etiqueta z^k.

PASO 3 La solución final se ha alcanzado cuando por primera vez un nodo etiquetado es un elemento de S_{n1}, esto ocurre tan pronto como todas las variables de una solución son enteros no negativos. Si z^k no es tal solución se supone que es un elemento de S_j. Se dibuja un nuevo arco originando así el nodo que está inmediatamente abajo de z^k (etiquetado, que no necesariamente z^{k-1}) y terminando en un punto en el mismo subconjunto de soluciones (S_j) como z^k o en S^*. Si este nuevo nodo está en S_j, su valor z es necesariamente menor o igual que z^k.

PASO 4 Se dibujan dos arcos de z^k a los puntos en S_{j+1} o en S^*. Nuevamente, si esos puntos están en S_{j+1} sus valores z son menores o iguales a z^k. Los últimos dos pasos aumentan tres nuevos arcos y nodos al árbol y regresamos al Paso 2.

Los nodos etiquetados forman una sucesión de cotas superiores no crecientes en el valor z de la solución final. Además el valor z de un nodo no etiquetado del conjunto S_j no puede exceder al del nodo etiquetado y entonces a la vez que z^k está etiquetado, es la mejor cota superior actual en el valor óptimo de z. Así, el método usado para aumentar arcos y nodos a la gráfica cubre todas las posibilidades, el algoritmo debe alcanzar la solución óptima de P(n_1), siempre que tal solución exista.

Como se podrá observar los algoritmos enumerativos son usualmente más sencillos de entender si se ilustran geométricamente a través de un árbol compuesto de nodos y arcos. Un nodo también se puede hacer corresponder a un

punto x^k y un arco une a ese nodo x^k con otro nodo x^{k+1}, donde definimos el último punto del anterior haciendo una de sus n-k variables libres igual a un entero no negativo. Como una variable libre x_r puede tener usualmente varios valores, es posible tener muchos arcos que emanen desde el nodo x^k. Como se puede ver en la figura 4.10 los nodos numerados 8, 9 y 10 se crearon haciendo x_1, una variable libre en el nodo 5 igual a 3, 4 y 2 respectivamente. Los nodos que no se han usado para crear otros nodos, o que no tienen arcos se llaman activos. Por ejemplo, el nodo 5 no es un nodo activo. También el número de nivel l se refiere al número de variables fijas.

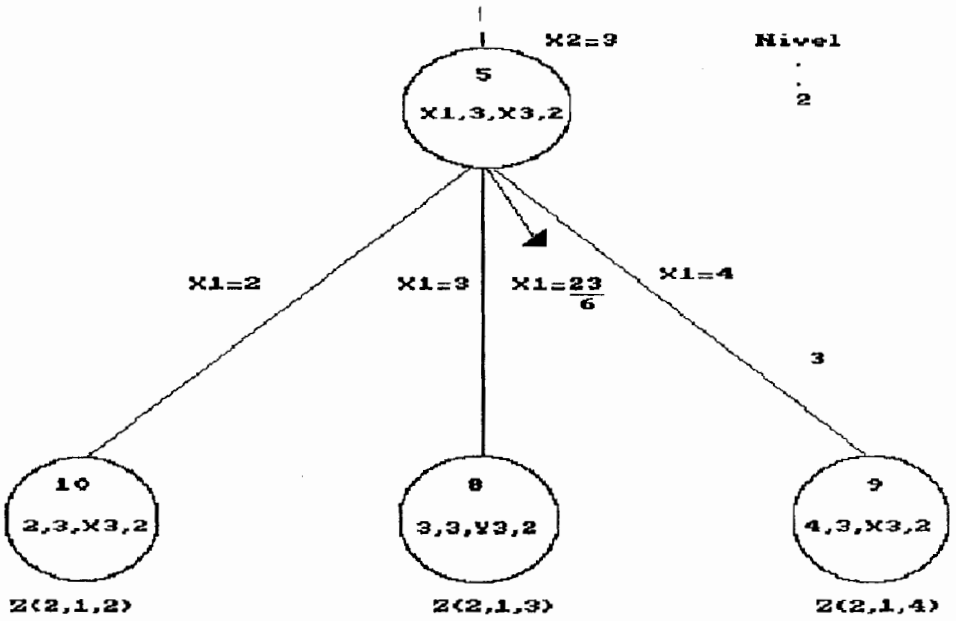


Figura 4.10

EJEMPLO 4.1
Considere el problema

Max $z = 77.9\ x_1 + 76.8\ x_2 + 89.6\ x_3 + 97.1\ y_1 + 31.3\ y_2$

Sujeto a:
$10.9\ x_1 + 3.6\ x_2 - 40.8\ x_3 + 43.9\ y_1 + 7.1\ y_2 + y_3 = 82.3$
$-86.8\ x_1 + 32.7\ x_2 + 24.3\ x_3 + 13.8\ y_1 - 12.6\ y_2 + y_4 = 77.3$
$60.9\ x_1 + 68.9\ x_2 + 69\ x_3 - 56.9\ y_1 + 22.5\ y_2 = 86.5$

con $x \geq 0$, $y \geq 0$, x entero

Resuelva usando el método de Land-Doig.

ITERACION 1

PASO 0

Resolvemos el problema original sin incluir las restricciones de variables enteras, aplicando el método simplex y obtenemos el valor de la función objetivo:

$$z^0 = 1165.50$$

Con valores en las variables de decisión:

$x_1 = 1.49$
$x_2 = 0$
$x_3 = 5.02$
$y_1 = 6.16$
$y_2 = y_3 = y_4 = 0$

PASO 1

Las variables no enteras, son x_1 y x_3. Elegimos como variable de ramificación la que tenga el valor más apartado al entero más cercano; x_1 es la más lejana a un valor entero. Calculamos el primer valor entero en las funciones Min x_1 y Max x_1, es inmediato que 1 = Min x_1 y 2 = Max x_1

1.1 Min x_1. Sea $x_1 = 1$, el valor óptimo de la función objetivo es entonces

$z = 1051.89$

con los siguientes valores en las variables

$x_1 = 1$
$x_2 = 0$
$x_3 = 4.40$
$y_1 = 5.48$
$y_2 = 1.48$
$y_3 = y_4 = 0$

1.2 Max x_1 Sea X1 = 2. El valor de la función objetivo es

$z = 770.69$

con los siguientes valores en las variables

$x_1 = 2$
$x_2 = 0$
$x_3 = 2.67$
$y_1 = 3.86$
$y_2 = y_3 = 0$
$y_4 = 132.57$

PASO 2

El mayor de los dos valores objetivo es z = 1051.89 con $x_1 = 1$, y se le asigna la etiqueta z. Para completar ésta etapa es necesario saber los valores de z para los enteros "cercanos".

PASO 3

Se calcula el mínimo de x_1 hasta su segundo valor entero: $x_1 = 0$ y se tiene que el valor de la función objetivo es:

$$z = 822.84$$

Con valores en las variables

$x_1 = 0$
$x_2 = 0$
$x_3 = 3.17$
$y_1 = 4.10$
$y_2 = 4.48$
$y_3 = y_4 = 0$

Estos tres valores de z se guardan en una lista en orden decreciente, para cada paso, así al inicio de cada Paso 2, se tomará un valor de la lista y al final de los pasos 3 y 4 se aumentarán 3 valores (inferiores). Los cálculos se terminan cuando todos los valores restantes de la lista son menores que un valor z asociado a una solución entera. En ésta etapa la lista está dada por:

Z		obtenida en el paso
1051.89	z^1	1
770.69		1
822.84		3

El valor más alto es 1051.89 que se ha etiquetado como z^1 (Paso 2). La solución y restricciones en este punto son $z^1 = 1051.89$, $x_1 = 1$, $x_2 = 0$, $x_3 = 4.40$, $y_1 = 5.48$, $y_2 = 1.48$, $y_3 = 0$

PASO 4

4.1 Se calcula Min x_3. Se hace $x_3 = 4$ entonces

$$z = 990.63$$

con valores en las variables:

$x_1 = 1$
$x_2 = 0.27$
$x_3 = 4$
$y_1 = 5.15$
$y_2 = 1.05$
$y_3 = y_4 = 0$

4.2 Se calcula Máx x_3. Entonces $x_3 = 5$, y $x_1 = 1$. Sin embargo se obtiene una solución no factible.

Nota: Este resultado aparentemente anormal se produce porque max x_3 es una

función fallida de Z. Lo que implica que el punto para el cual x_1=1, x_3=5 cae en S*, ya que el problema con tales restricciones no es factible.

La lista de valores de Z es:

z		obtenida en el paso
1051.89	z^1	1
770.69		1
822.84		3
990.63		4

ITERACION 2

PASO 2

El mejor valor no etiquetado es 990.63, que será z^2

PASO 3

Llevamos a min x_3 un paso mas adelante. Sea x_3 = 3 y se obtiene el siguiente valor de la función objetivo:

$$z = 840.65$$

Con valores en las variables

$x_1 = 1$
$x_2 = 0.94$
$x_3 = 3$
$y_1 = 4.33$
$y_2 = y_3 = 0$
$y_4 = 0.33$

Como en x_3 = 3 z = 840.82, en éste punto la mejor solución no etiquetada es 990.63 por lo cual la solución y restricciones son ahora: z^2 = 990.63, x_1 = 1, x_2 = 0.27, x_3 = 4, y_1 = 5.15, y_2 = 1.05

PASO 4

4.1 Se calcula Mín x_2 y se hace x_2 = 0, con valor en la función objetivo:

$$z = 981.60$$

y valores de las variables:

$x_1 = 1$
$x_2 = 0$
$x_3 = 4$
$y_1 = 5.07$
$y_2 = 1.69$
$y_3 = 0$
$y_4 = 18.26$

4.2 Se calcula Máx x_2 y se hace $x_2 = 1$ Aqui se agrega otro punto a S^*, puesto que se tiene una solución no factible.

Entonces se tiene una solución en

$z = 981.60$
$x_1 = 1$
$x_2 = 0$
$x_3 = 4$
$y_1 = 5.0702$
$y_2 = 1.0930$

Como éste valor de z es el mayor de los restantes en la lista, ésta es la solución óptima

El árbol de expansión que ilustra los cálculos anteriores se presenta a continuación

Algunas características importantes de éste algoritmo son las siguientes:

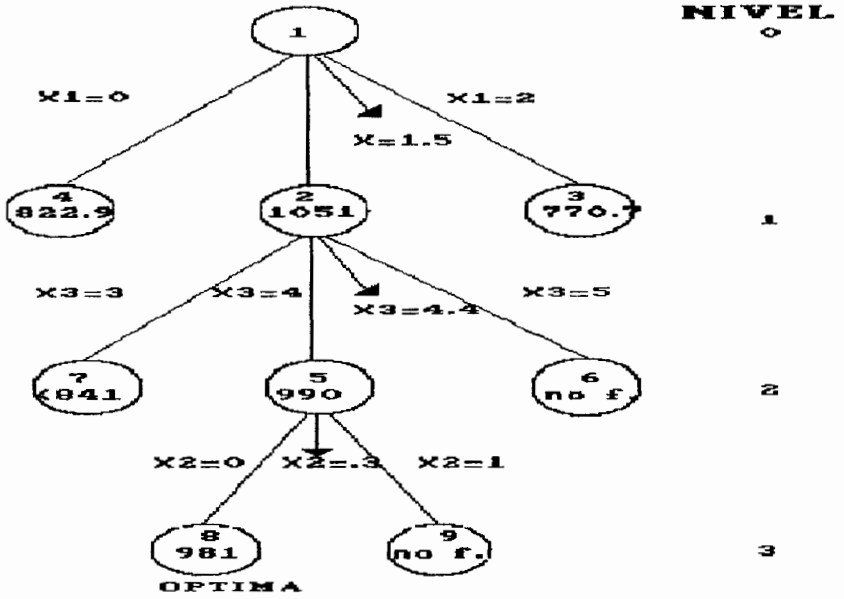


Figura 4.11

Cuando un sub-problema en el nivel n es factible, se produce una solución al Problema Entero Mixto. Por lo tanto el algoritmo en el peor de los casos se enumeran explícitamente todos los del árbol, y los nodos en el último renglón representan todas las combinaciones posibles de enteros.

Para reducir el esfuerzo computacional es probable que valga la pena introducir explícitamente la desigualdad $cx^1 + dy \geq z^0 + \varepsilon$ para cada problema lineal PL^1, donde ε es un número pequeño positivo. Haciendo esto un problema lineal factible siempre indica un nodo activo y se tienen entonces, mejores soluciones del

Problema Entero Mixto.
En este algoritmo se selecciona para ramificar el nodo con la solución lineal mayor. Con ésta estrategia se pretende encontrar rápidamente una buena solución entera mixta, sin embargo éste procedimiento de selección puede requerir mucho almacenamiento computacional.

La variable que se selecciona para ramificar es aquella con componente más fraccionario

En este algoritmo, la mayoría (si no todos) los problemas lineales se resuelven completamente. Esto resulta costoso desde el punto de vista computacional, especialmente cuando la única información necesaria en un nodo puede ser el valor de la función objetivo.

El proceso continuo de ramificar y acotar, sugiere la designación de Ramificación y Acotamiento al algoritmo.

Algunos inconvenientes de éste algoritmo son:

Es posible que un número grande de arcos provenga del mismo nodo. El problema es que esto no se puede predecir por anticipado, lo que complica el árbol de búsqueda y puede proporcionar un aumento severo en la memoria de la computadora.

Si el algoritmo se toma literalmente, será necesario resolver un problema lineal para cada ramificación en aras de determinar la cota superior propia. Esto lo hace aparecer como un método costoso, especialmente porque algunos de los nodos resultantes probablemente nunca se ramifiquen.

La determinación de una cota inferior sucede sólo como resultado de resolver el programa lineal en los diferentes nodos. Lo que significa que no obstante la importancia de cálculos truncados, no hay un método sistemático que asegure una cota inferior "estrecha" en una etapa temprana del procedimiento.

ALGORITMO DE DAKIN

Un algoritmo similar pero más sencillo de instrumentar que el algoritmo de Land y Doig, fue propuesto por Dakin. El algoritmo es aplicable a problemas tanto lineales como no lineales, sin embargo sólo se tiene experiencia computacional para el caso de programación lineal. Para explicar las bases de tal algoritmo considere nuevamente el problema:

Maximizar $z = cx + dy$
sujeto a
$Ax + Dy = b$
x vector entero
$x \geq 0, y \geq 0$

El problema sin la condición de que x sea entero se denomina el problema continuo. Una solución al problema continuo que también satisfaga x entero se llama una solución entera; en caso contrario se dice que es una solución no entera.

Suponga que existe un algoritmo (que llamaremos sub-algoritmo) para encontrar soluciones a los problemas resultantes del problema continuo con la adición de cotas superiores o inferiores en alguna de las variables enteras. La existencia de tal algoritmo limita nuestro problema donde z es cóncava y (4.2) y (4.4) se especifican como una región convexa.

Se demostrará que si el sub-algoritmo es finito, entendiendo por esto que resuelve cada sub-problema en tiempo finito, y las variables enteras acotadas, entonces se alcanzará una solución óptima entera después de un número finito de cálculos.

Suponga que x_j es una variable entera y k es un entero. Dado que el rango $k<x_j<k+1$ es inaceptable, podemos dividir todas las soluciones que cumplan las restricciones en dos grupos separados

i) Soluciones en que $x_j \leq k$ (4.6)
ii) Soluciones en que $x_j \geq k + 1$ (4.7)

Al igual que el algoritmo de Land y Doig, en éste algoritmo se comienza con una solución al problema continuo. Si ésta solución es entera entonces es la solución del problema, de otra manera al menos una variable entera -digamos x_j- es no entera y toma un valor b_j en ésta solución. Se divide b_j en dos partes, entera y fraccional $[b_j] + f_j$ respectivamente, definida por:

$$b_j = [b_j] + f_j \tag{4.8}$$

donde $[b_j]$ es un entero y $0 < f_j < 1$ Sustituimos

$$k = [b_j] \tag{4.9}$$

En (4.6) y (4.7). Aparentemente no se satisface ninguna de esas relaciones con la actual solución no entera, por lo que aumentamos ésta restricción al problema continuo y resolvemos los problemas resultantes, Se repite el procedimiento para cada una de las dos soluciones obtenidas. La estructura lógica de éste proceso es

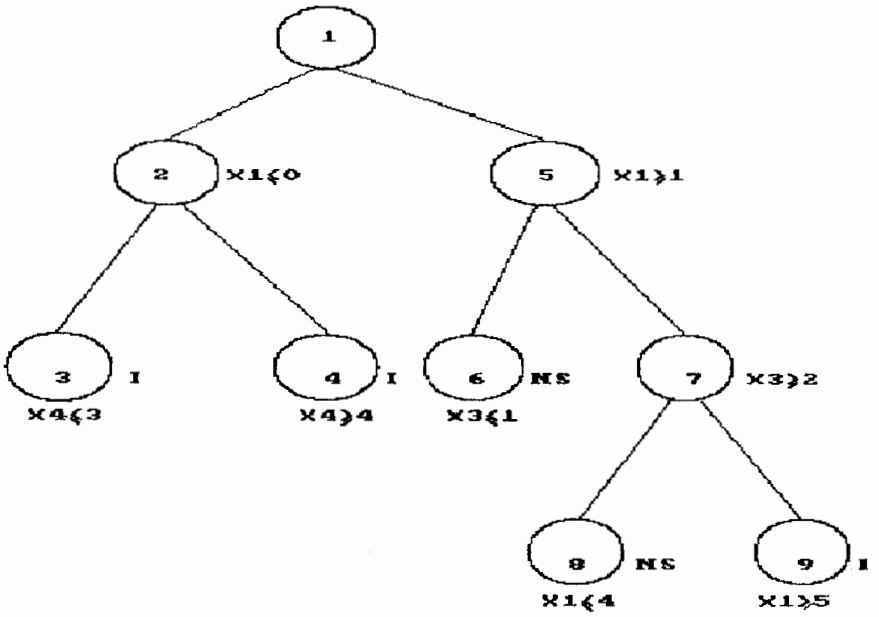


figura 4.12

la de un árbol, como el que se muestra en la figura 4.12
El árbol siempre terminará en uno de dos caminos, puede alcanzar una solución entera (denotada por "I" en la figura) o se puede encontrar que el conjunto actual de restricciones no tiene solución (denotado por "NS"). La solución al problema completo será la mejor solución entera alcanzada de ésta manera.

El árbol -en general- no es único para un problema dado, ya que en cualquier etapa se tiene libertad de formar la siguiente restricción usando cualquier xi que no sea entera; la selección de diferentes xj producirá diferentes árboles. Como se podrá ver la selección puede tener un gran efecto en la cantidad de cálculos requeridos.

Nótese que sólo hay dos ramas y por lo tanto dos subárboles que salen de cada nodo -en contraste con el algoritmo de Land-Doig, donde puede salir cualquier número de ramas.

Se puede operar mas de una restricción en una variable al mismo tiempo. Por ejemplo, en el nodo 9 de la figura 2.4 se fijan las restricciones $x_1 \geq 1$ y $x_1 \geq 5$ a la vez. El número de restricciones que pueden operar al mismo tiempo en una variable está limitado por el rango de valores que puede tomar la variable. Así, si la restricción original (2) limita a x_j a un rango de $0 \leq x_j \leq v$ entonces el entero v acota a x_j en éste rango y es consistente con $x_j = k$; por ejemplo

$$x_j \geq 1,\ x_j \geq 2,\ldots,x_j \geq k,\ x_j \leq k,\ x_j \leq k+1,\ldots,x_j \leq v-1.$$

ALGORITMO DE DAKIN

PROPÓSITO: Resolver el problema de programación entera mixta

DESCRIPCIÓN

PASO 1 Se resuelve el problema entero por medio del método simplex de programación lineal. Si la solución es entera, pare, se ha encontrado la solución óptima.

PASO 2 Se escoge arbitrariamente una variable entera x_j cuyo resultado en el Paso 1 sea fraccionario e igual a f_j.

PASO 3 Se resuelven dos nuevos problemas similares al problema original, uno con la restricción adicional $x_{jk} \leq k$ y el otro con la restricción adicional $x_j \geq k+1$.

PASO 4 De los programas lineales resueltos en el paso 3 se incluyen en el análisis a seguir sólo aquellos programas cuya solución (entera o fraccional) sea mejor a cualquiera de las soluciones enteras conocidas.

PASO 5 Se selecciona aquél programa lineal que tenga el mínimo valor de la función objetivo. Si las variables enteras tienen valor entero, se ha encontrado la solución óptima. Si no, regrese al paso 2 con la estructura del problema lineal resuelto en éste paso.

En los programas de cómputo que se han escrito para programación lineal, usando el método simplex como sub-algoritmo y guardando el tableau completo, el requerimiento total de almacenamiento de datos es

$$\Sigma v_j + (m + 2)(n'+ 2) \text{ palabras}$$

Donde m es el número de restricciones en el problema continuo (excluyendo restricciones de cota superior si se usa el procedimiento de variables acotadas), y n'es el número de variables no básicas, v_j es el rango de valores que puede tomar la variable x_j, de donde la suma de estos posibles valores acotan los requerimientos de almacenamiento.

EJEMPLO 4.2
Suponga que se desea resolver el problema:

$$\text{Max } z = 5x_1 + 2x_2$$
sujeto a
$$2x_1 + 2x_2 + x_3 = 9$$
$$3x_1 + x_2 + x_4 = 11$$

Usando el algoritmo de Dakin

ITERACIÓN 1

PASO 1

La solución óptima del problema continuo correspondiente tiene como valor de la función objetivo:

$$z = 18.75$$

Y los valores de las variables son:

$$x_1 = 3.25$$
$$x_2 = 1.25$$
$$x_3 = x_4 = 0$$

PASOS 2 -3

Se escoge arbitrariamente $x_2 = 1.25$ y se resuelven dos problemas lineales distintos, uno con la restricción adicional $x_2 \leq [1.25] = 1$ y el otro con la restricción adicional $x_2 \geq [1.25] + 1 = 2$ Es decir:

Problema (1) : Consiste en resolver

$$\text{Max } z = 5x_1 + 2x_2$$
sujeto a
$$2x_1 + 2x_2 + x_3 = 9$$
$$3x_1 + x_2 + x_4 = 11$$
$$x_2 + x_5 = 1$$

Cuyo valor óptimo de la función objetivo es

$$z = 18.67$$

Y los valores de las variables son:

$$x_1 = 3.33$$
$$x_2 = 1$$
$$x_3 = .33$$
$$x_4 = x_5 = 0$$

Problema (2) : Consiste en resolver

$$\text{Max } z = 5x_1 + 2x_2$$
sujeto a
$$2x_1 + 2x_2 + x_3 = 9$$
$$3x_1 + x_2 + x_4 = 11$$
$$x_2 - x_5 = 2$$

Cuyo valor óptimo de la función objetivo es igual a: $z = 16.5$ y los valores de las variables son:

$x_1 = 2.5$
$x_2 = 2$
$x_3 = x_5 = 0$
$x_4 = 1.5$

PASO 4
Como no ha habido ninguna solución entera en todo el proceso, se incluyen ambos problemas en el análisis.

PASO 5
Como la mejor función objetivo hasta el momento corresponde a una solución no entera (z = 18.67), se regresa al paso 2 con el problema (1) como candidato

ITERACIÓN 2

PASO 2
En el Problema (1) se escoge arbitrariamente la variable $x_1 = 3.33$ y se resuelven dos nuevos problemas. Uno que es igual al problema (1) más la restricción $x_1 \leq [3.33] = 3$. El otro que es igual al problema (1) más la restricción $x_1 \geq [3.33] + 1 = 4$. Específicamente:

Problema (3) : Consiste en resolver

$$\text{Max } z = 5x_1 + 2x_2$$
sujeto a
$$2x_1 + 2x_2 + x_3 = 9$$
$$3x_1 + x_2 + x_4 = 11$$
$$x_2 + x_5 = 1$$
$$x_1 + x_6 = 3$$

PASO 3
Se obtienen las soluciones óptimas al problema lineal (3). El valor óptimo de la función objetivo es:

$z = 17$

Y los valores de las variables son:
$x_1 = 3$
$x_2 = x_3 = x_4 = 1$
$x_5 = x_6 = 0$

El problema (4) no tiene solución factible y no se incluye en el listado.

PASO 4
Por tener una solución entera se incluye en el análisis.

PASO 5

Por ser el mejor valor de la función objetivo y además, ser entero, es la solución óptima.

El árbol que muestra el desarrollo del método se describe en la figura 4.13

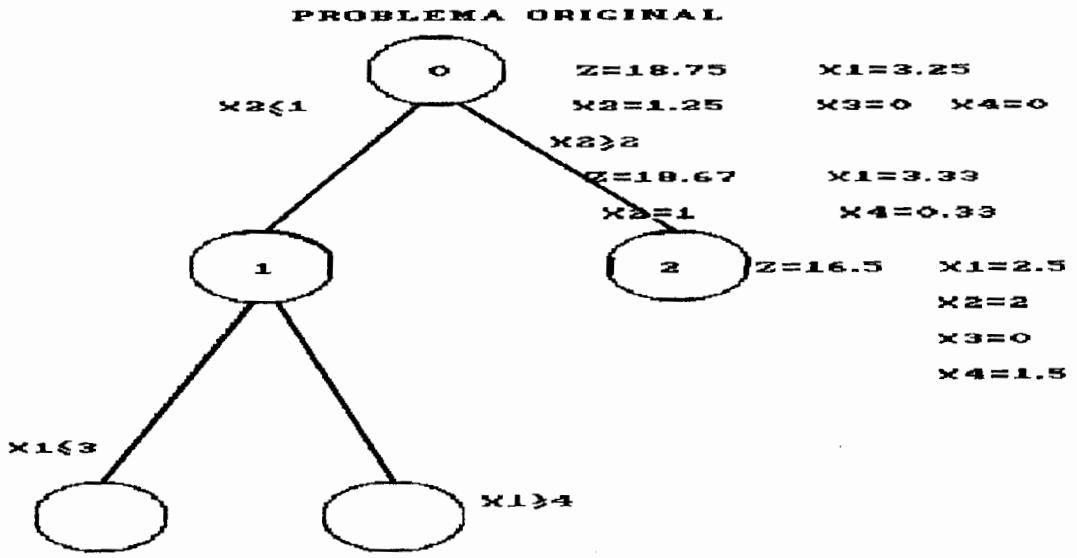


figura 4.13

COMPARACIÓN CON EL MÉTODO DE LAND Y DOIG

La cercana conexión entre el algoritmo descrito y el propuesto por Land-Doig (1960) es evidente. La diferencia básica es que el algoritmo de L-D fuerza a las variables enteras a tomar valores enteros en lugar de aplicar cotas, de tal forma que es necesario buscar sobre el rango de valores enteros para los cuales z es mayor que el valor que toma en la mejor solución entera conocida. Si todas las variables enteras se restringen a los valores 0 y 1 los dos métodos son iguales y la única contribución del método presentado consiste en proporcionar un procedimiento computacional conveniente para llevarlo a cabo.

La convergencia de ambos métodos depende de la selección de la variable entera que se maneja en cada etapa. Sin embargo hay indicios que hacen suponer una convergencia más rápida para éste algoritmo en muchos casos, ya que hará una búsqueda sobre un rango más pequeño de valores para una variable entera que el algoritmo de Land y Doig, pues podemos detenernos tan pronto como se alcance una solución entera. Más aún, este método en algunos casos se saltará valores que alguna variable entera pueda tomar con el algoritmo de Land y Doig, y como se ha visto, se hace la ramificación en solo dos ramas y no en mas como el método anterior.

A continuación se desarrolla un ejemplo usando tanto el algoritmo de Land y Doig, como el de Dakin para observar claramente la diferencia entre ambos

EJEMPLO 4.3
Considere el problema entero

$$\text{Max } z = -4x_1 - 5x_2 \qquad (4.10)$$

Sujeto a

$$-x_1 - 4x_2 \leq -5 \qquad (4.11)$$
$$-3x_1 - 2x_2 \leq -7 \qquad (4.12)$$
x_1, x_2 enteros no negativos (4.12')
$$x_1 \geq 0, \quad x_2 \geq 0 \qquad (4.13)$$

Aplique el algoritmo de Land-Doig:

PASO O
La solución al problema definido por (4.10), (4.11),(4.12) y (4.13) tiene como valor óptimo para su función objetivo:

$$z = -11.20$$

Cuyos valores en las variables son :

$$x_1 = 1.80$$
$$x_2 = 0.80$$

PASO 1
La cota superior de z y la solución actual al iniciar este paso es

$$z^0 = -11.2$$
$$x_1 = 1.8$$
$$x_2 = 0.8$$

Se escoge cualquiera de las dos variables, pues ambas están igual de lejanas al entero más próximo, sea $x_1 = 1.8$ y se hace max. x_1 y min x_1

1.1 min x_1 implica $x_1=1$, cuyo valor óptimo de la función objetivo es igual a: $z = -14$. Y los valores en las variables:$x_1 = 1, x_2 = 2$

1.2 Máx x_1 implica $x_1=2$, con valor óptimo de la función objetivo igual a:$z = -11.75$, y los valores de las variables son: $x_1 = 2$, $x_2 = .75$

PASO 2
El mayor de los dos es $z=-11.75$ con $x_1=2$, y se le asigna la etiqueta z^1. Para completar ésta etapa es necesario saber los valores de z para los enteros que se pueden alcanzar "más cercanos".

PASO 3

Se continúa con max x_1 hasta su segundo valor entero; x_1=3 y se tiene que el valor óptimo de la función objetivo es: z = -14.5, y los valores de las variables: x_1 = 3, x_2 = 0.5

Estos tres valores de z se guardan en una lista y se tiene

Z		obtenida en el paso
-11.75	z^1	1
-14		1
-14.5		3

El valor más alto es -11.75 que se ha etiquetado como z^1. La solución y restricciones en éste punto son:

restricciones $x_1 = 2$
solución $z^1 = -11.75$
$x_1 = 2$
$x_2 = .75$

PASO 4

4.1 Min x_2 implica x_2=0, no se tiene solución factible

4.2 Max x_2 implica x_2=1, cuyo valor óptimo de la función objetivo es: z = -13, y los valores de las variables son: x_1 = 2, x_2 = 1

La figura 4.14 ilustra el árbol de búsqueda asociado a éste problema

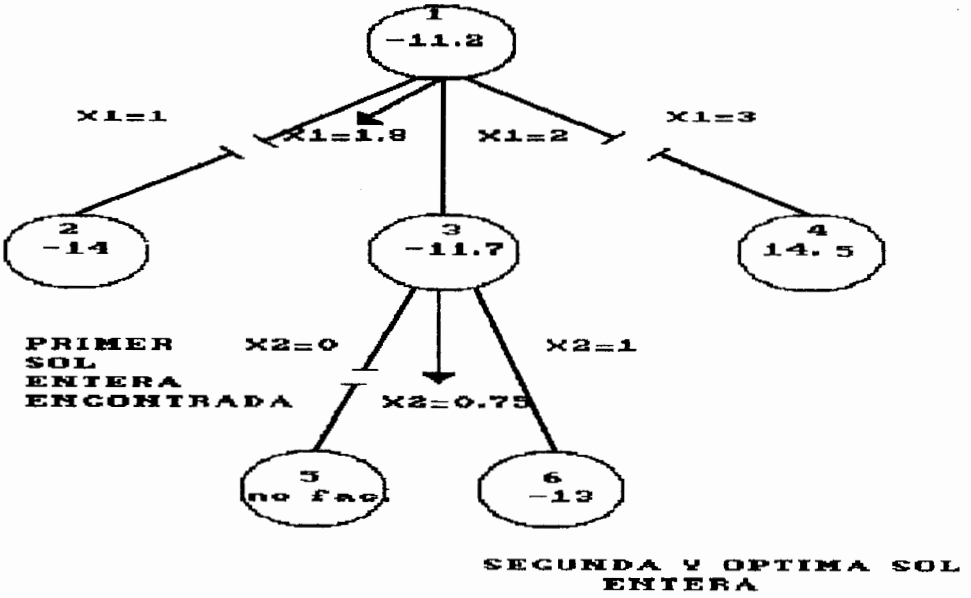


figura 4.14

Ahora desarrollamos el mismo ejemplo usando el algoritmo de Dakin

$$\max z = -4x_1 - 5x_2$$
sujeto a
$$-x_1 - 4x_2 \leq -5$$
$$-3x_1 - 2x_2 \leq -7$$

ITERACION 1

PASO 1
La solución óptima del problema lineal correspondiente tiene como valor óptimo de la función objetivo: $z = -11.2$ y con valores en las variables :$x_1 = 1.8$, $x_2 = .80$

PASO 2
Se escoge arbitrariamente $x_1=1.8$, y se resuelven los dos problemas lineales distintos uno con la restricción adicional $x_1 \leq [1.8] = 1$ y el otro con la restricción adicional $x_1 \geq [1.8] + 1=2$.

PASO 3
Se tienen los tableaus óptimos

Problema (1)
Con $x_1 \leq 1$, se tiene que el óptimo en la función objetivo es $z = -14$. Y con valores en las variables: $x_1 = 1$, $x_2 = 2$

Problema (2)
Con $x_1 \geq 2$, se tiene que el óptimo en la función objetivo es $z = -11.75$. Y los valores de las variables: $x_1 = 2$, $x_2 = .75$

PASO 4
Hay una solución entera sin embargo la mejor función objetivo corresponde a una solución no entera, y nos regresamos al Paso 2

ITERACION 2

PASO 2
Escogemos $x_2 = 75$ y se resuelven los dos nuevos problemas. Uno igual al problema 1 más la restricción $x_2 \leq [.75] = 0$ y el otro que es igual al problema 1 más la restricción $x_2 \geq [.75]+1 = 1$ es decir:

Problema (3)
$x_2 \leq 0$, se tiene que no hay solución factible.

Problema (4)
Con $x_2 \geq 1$, se tiene que el valor óptimo de la función objetivo es igual a: $z = -13$. Cuyos valores en las variables son: $x_1 = 2$, $x_2 = 1$.

La figura 4.15 ilustra el árbol de búsqueda correspondiente a éste problema.

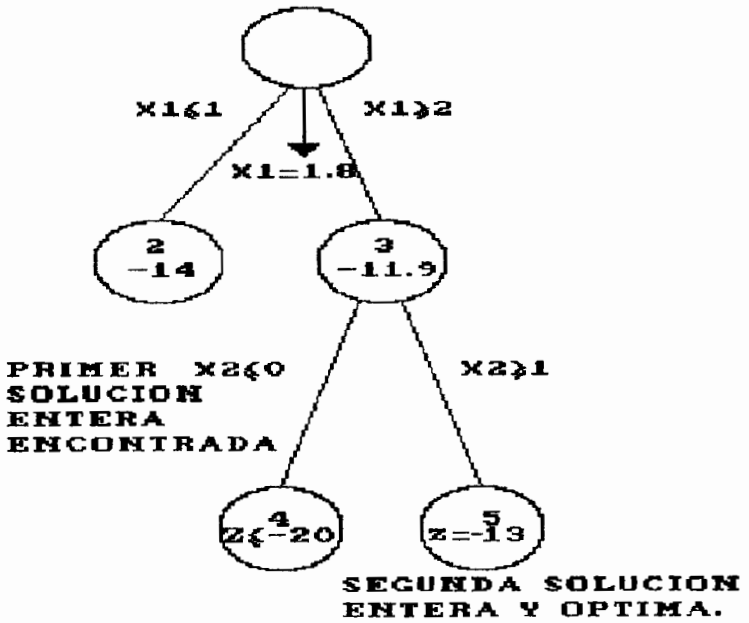


figura 4.15

ALGORITMO PARA LA SOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN ENTERA MIXTA DE DRIEBEEK

Este algoritmo, basado en el método de Land-Doig convierte a todas las variables enteras en binarias (cero-uno). Una vez realizado esto se sustituyen en el problema original y se encuentra la solución óptima utilizando programación lineal con ciertos ajustes, que se denominan costos penalizados. Dichos costos sirven para seleccionar la mejor variable entera que genera una bifurcación, y por lo tanto aceleran al método de R-A, es decir, lo hacen converger mas rápidamente a la solución óptima. El algoritmo fue propuesto por Norman J. Driebeek en marzo de 1966 y posteriormente en marzo de 1968 Stanley Zionts propuso algunas modificaciones al algoritmo que permiten considerar una restricción y n+1 variables para cada variable entera en lugar de n+1 restricciones y 2n+1 variables como se hacía originalmente. A continuación se presenta el algoritmo modificado por Zionts:

ALGORITMO DE DRIEBEEK

PROPÓSITO: Resolver el problema entero mixto de programación lineal.

DESCRIPCIÓN

PASO 1 Se convierte a todas las variables enteras en binarias de la siguiente manera: Sea x_i una variable entera, entonces

$$x_i = \sum_{k=0}^{n} k\alpha_{ik}$$

donde $\alpha_{ik} = 0$ o 1, para toda i, k

$$\sum_{K=0}^{n} \alpha_{ik} = 1$$

Si $\alpha_{ik} = 1$, esto implica que la variable entera es $x_i = k$ y se resuelve el problema no entero. Si se encuentra una solución óptima entera, parar.

PASO 2 Se calculan las penalizaciones de la siguiente manera:

a) Por cada variable α_{ik} no básica, el costo penalizado por alcanzar un valor $\alpha_{ik} = 1$, es decir $x_i = k$, es precisamente el "precio sombra"de α_{ik} es decir el valor z_{ik}-c_{ik} del tableau correspondiente.
b) Por cada variable α_{ik} básica, el costo penalizado por incrementar α_{ik} al valor 1, es decir, hacer $x_i = k$, es el mínimo incremento en la función objetivo por hacer $\alpha_{ik} = 1$ (o sea, hay que usar análisis de sensibilidad al hacer el término X_{Bi} correspondiente al vector básico α_{ik} igual a uno).

El cambio mínimo en el valor de la función objetivo es la cantidad en la que cambia la función objetivo en la primer iteración del método dual simplex, al hacer el cambio correspondiente en el lado derecho del tableau. Este cambio es igual a:

$$(x_{Bi})\text{máx}\left\{\frac{z_j - c_j}{Y_{rj}} \ni Y_{rj} \le 0\right\}$$

donde x_{Br} es el valor del vector básico a_{ik} y Y_{rj} son los valores del tableau correspondiente

PASO 3 De los costos penalizados calculados en a) se selecciona el mínimo. Si hay un empate, se escogen todos. Se combinan entonces con el costo mínimo penalizado calculado en b) y se selecciona aquella solución factible cuya suma de costos penalizados (los calculados en a) y en b)) sea mínima.

CONVERGENCIA

La experiencia computacional con el algoritmo ha sido buena, pues el algoritmo busca esencialmente una reducción mínima en el valor de la función objetivo cuando se activan las restricciones enteras. Porque las soluciones prueba se inician en la base de las penalizaciones, y porque aquellos puntos que tienen menores penalizaciones se prueban primero, se espera que la mejor o al menos una muy buena solución entera, se encuentre en una de las primeras pruebas; de los problemas que se han resuelto se ha encontrado la solución óptima en las primeras pruebas. En un problema que contiene 21 variables cero-uno o (2) 21 puntos, la solución óptima se encontró en la segunda prueba. Después de que una prueba se inicia, ésta sólo termina sí:

a) En una iteración dual, el valor de la función objetivo disminuye por debajo de la solución encontrada.
b) Se encuentra una condición dual no acotado (primal no factible).
c) Una solución entera válida, mejor que la que se encontró previamente.

El tiempo total de corrida del algoritmo depende principalmente del número de puntos de la red en el problema y menos del significado de las variables enteras. Cuando las variables se usan para representar costos iniciales, la solución se encuentra con mayor facilidad que cuando las variables enteras se usen para decidir entre parejas de variables muy similares. La mayoría de los problemas enteros se han resuelto de k a 3k iteraciones, donde k es el número de iteraciones requeridas para alcanzar una solución continua.

EJEMPLO 4.4

Considere el problema entero-mixto

$$\text{Max } z = 5x_1+2x_2$$

sujeto a

$$2x_1+2x_2+x_3 = 9$$
$$3x_1+ x_2 +x_4 = 11$$
$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0;\ x_1,\ x_2 \text{ enteros}$$

Resuelva usando el método de Driebeek.

Empezamos por hacer:

$$x_1 = 0\alpha_{10} + 1\alpha_{11} + 2\alpha_{12} + 3\alpha_{13} + \ldots$$
$$x_2 = 0\alpha_{20} + 1\alpha_{21} + 2\alpha_{22} + 3\alpha_{23} + \ldots$$

con

$$\alpha_{ij} = 0 \text{ o } 1,$$
$$\alpha_{10} + \alpha_{11} + \alpha_{12} + \ldots = 1$$
$$\alpha_{20} + \alpha_{21} + \alpha_{22} + \ldots = 1$$

Se tiene que como x_1 puede alcanzar un valor máximo factible de 3, y x_2 de 4, se toman los primeros cuatro términos de α_{1j} y los primeros cinco términos de α_{2j}, y el problema original se convierte en:

$$\max z = 5\alpha_{11} + 10\alpha_{12} + 15\alpha_{13} + 2\alpha_{21} + 4\alpha_{22} + 6\alpha_{23} + 8\alpha_{24}$$

sujeto a

$$2\alpha_{11} + 4\alpha_{12} + 6\alpha_{13} + 2\alpha_{21} + 4\alpha_{22} + 6\alpha_{23} + 8\alpha_{24} + x_3 = 9$$
$$3\alpha_{11} + 6\alpha_{12} + 9\alpha_{13} + \alpha_{21} + 2\alpha_{22} + 3\alpha_{23} + 4\alpha_{24} + x_4 = 11$$
$$\alpha_{11} + \alpha_{12} + \alpha_{13} + \ldots\ldots\ldots\ldots + \alpha_{10} = 1$$
$$\alpha_{21} + \alpha_{22} + \alpha_{23} + \alpha_{24} + \ldots\ldots\ldots\ldots + \alpha_{20} = 1$$

La primera tabla es:

	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}	α_{24}	x_3	x_4	α_{10}	α_{20}	x_B
z	-5	-10	-15	-2	-4	-6	-8	0	0	0	0	0
x_3	2	4	6	2	4	6	8	1	0	0	0	9
x_4	3	6	9	1	2	3	4	0	1	0	0	11
α_{10}	1	1	1	0	0	0	0	0	0	1	0	1
α_{20}	0	0	0	1	1	1	1	0	0	0	1	1

Después de dos iteraciones del método simplex, se obtiene la siguiente solución óptima no entera:

	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}	α_{24}	x_3	x_4	α_{10}	α_{20}	x_B
z	6	3	0	0	0	0	0	1	0	9	0	18
α_{24}	-0.5	-0.25	0	0.25	0.5	0.75	1	.125	0	-0.75	0	.375
x_4	-4	-2	0	0	0	0	0	-0.5	1	-6	0	0.5
α_{13}	1	1	1	0	0	0	0	0	0	1	0	1
α_{20}	0.5	0.25	0	0.75	0.5	0.25	0	-.125	0	0.75	1	.625

El cálculo de costos penalizados al pasar de α_{ik} distinto de 1 a $\alpha_{ik} = 1$ es el siguiente: α_{ij} no básicas

α_{ij}	***$x_i = j$***	***costo penalizado $= (z_{ij}-c_{ij})$***
α_{10}	$x_1 = 0$	9
α_{11}	$x_1 = 1$	6
α_{12}	$x_1 = 2$	3
α_{21}	$x_2 = 1$	0
α_{22}	$x_2 = 2$	0
α_{23}	$x_3 = 3$	0

α_{ij} básicas:

α_{ij}	$x_i = j$	costo penalizado (X_{Br}-1) Máx $\{z_j-c_j/Y_{rj} \mid Y_{rj} \leq 0\}$
α_{13}	$x_1 = 3$	(1-1) Máx {no hay candidatos} = 0
α_{20}	$x_2 = 0$	(5/8 - 1) Máx {1/(-1/8)} = (-3/8)(-8) = 3
α_{24}	$x_2 = 4$	(3/8 - 1) Máx {9/(-3/4),6/(-1.2),3/(-1/4)} = 15/2

Los costos penalizados mínimos se obtienen cuando:

$x_1 = 3$ y $x_2 = 1$ (solución factible)
con costo penalizado total 0+0 = 0

$x_1 = 3$ y $x_2 = 2$ (solución no factible)
con costo penalizado total 0+0 = 0

$x_1 = 3$ y $x_2 = 3$ (solución no factible)
con costo penalizado total 0+0 = 0

Como las dos últimas soluciones no son factibles, pues violan alguna de las restricciones y la primera solución es factible, es también óptima por lo que la solución queda:

$$z = 17,\ x_1 = 3,\ x_2 = 1$$

Como se podrá observar el algoritmo determina una cota superior (inferior) de la función objetivo en el caso de maximización (minimización), y por medio de los costos penalizados se ajusta con el costo penalizado mínimo, la solución óptima del problema entero. Este algoritmo puede dejar de iterar cuando la solución factible que se tiene se aproxima en un 80%-90% a la solución óptima.

Como veremos en el siguiente ejemplo, los costos penalizados sirven para seleccionar la mejor variable entera. Dada una variable básica $x_r=x_{Br}$ cuyo resultado final debe ser entero, y por el momento todavía es fraccionario, se tiene que el costo penalizado de hacer $x_r=[x_{Br}]$ (donde [x] es el máximo entero z menor o igual a x, y <x> el número entero z más pequeño, mayor o igual a x) es:

$$(x_{Br} - [x_{Br}]) \min_{j=1\ldots n} \left\{ \frac{z_j - c_j}{Y_{rj}} \ni Y_{rj} \geq 0 \right\}$$

mientras que el costo penalizado de hacer $x_r = \langle x_{Br} \rangle$ es:

$$(1 - x_{Br} + [x_{Br}]) \min_{j=1\ldots n} \left\{ \frac{z_j - c_j}{-Y_{rj}} \ni Y_{rj} \geq 0 \right\}$$

Entonces, asociada a cada variable básica x_r, que aún es fraccionaria, pero que en la solución óptima debe ser entera, se tienen dos costos penalizados, uno asociado con el cambio $x_r = [x_{Br}]$ y el otro con $x_r = \langle x_{Br} \rangle$. Para identificación, denótese al primer costo penalizado por $[CP]_r$ y al segundo por $\langle CP \rangle_r$. La variable que se selecciona para ramificarse es aquella cuyo $[CP]_r$ ó $\langle CP \rangle_r$ es el máximo entre todas las variables básicas x_r, descartando por supuesto a las de holgura, que siendo aún fraccionarias, deben ser enteras en la solución óptima. Este proceso acelera al método, pues al elegir una variable básica con costo penalizado alto, se evita aumentar el número de iteraciones al eliminar implícitamente soluciones peores a las actuales.

EJEMPLO 4.5

Resuelva el siguiente problema de programación entera usando el procedimiento descrito anteriormente para acelerar la convergencia

$$\max z = 5x_1 + 2x_2$$
sujeto a
$$2x_1 + 2x_2 + x_3 = 9$$
$$3x_1 + x_2 + x_4 = 11$$
$$20x_1 - 10x_2 - x_5 = 51$$
$$x_i \geq 0 \text{ enteros, } i = 1,...,5$$

La solución óptima del problema lineal asociado tiene como valor óptimo de la función objetivo

$$z = 18.75$$

Y los valores de las variables son:

$$x_1 = 3.25$$
$$x_2 = 1.25$$
$$x_3 = x_4 = 0$$
$$x_5 = 1.5$$

El cálculo de los costos penalizados $[CP]_i$, $<CP>_i$, i = 1,2 para las variables básicas (x_1, x_2) es:

$[CP]_1 = (3.25\text{-}3) \min \{0/1, 1.5/.5\} = 0$
$<CP>_1 = (1\text{-} 3.25 + 3 \text{ mín } \{-.25/-.25\} = .75$
$[CP]_2 = (1.25 \text{ -}1) \min \{0/1,.25/.75\} = 0$
$<CP>_2 = (1 \text{ -}1.25+1) \min \{1.5/-(-.5)\} = 0.75(3) = 2.25$

La variable que se selecciona es la x_2 y las dos nuevas ramificaciones estarán dadas por las nuevas restricciones

$$x_2 \leq 1 \text{ y } x_2 \geq 2.$$

1. Sea $x_2 \leq 1$

La solución óptima del problema lineal asociado es

	x_1	x_2	x_3	x_4	x_5	x_6	x_B
z	0	0	0	1.67	0	0.34	18.67
α_{24}	0	0	1	-.67	0	-1.34	.34
x_4	0	0	0	6.67	1	-16.6	5.67
α_{13}	1	0	0	0.34	0	-.34	3.34
α_{20}	0	1	0	0	0	1	1

Y el valor óptimo de la función objetivo es:

$$Z^* = 18.67$$

Con valores en las variables

$x_1 = 3.3$
$x_2 = 1$
$x_3 = .3$
$x_4 = x_6 = 0$
$x_5 = 5.67$

Se calculan nuevamente los costos penalizados $[CP]_i$, $<CP>_i$ para las variables básicas x_1 y x_2

$[CP]_1 = (3.34 - 3) \min \{0, 1.67/0.34\} = 0$
$<CP>_1 = (1 - 3.34 + 3) \min \{-0.34/-0.34\} = 0.67$
$[CP]_2 = (1-1) \min \{0, 0.34/1\} = 0$
$<CP>_2 = (1 - 1 + 1) \min \{\text{No hay disponibles}\} = 0$

Ahora vemos si hay solución para $x_2 \geq 2$

2. Con $x_2 \geq 2$ no se tiene solución factible, entonces escogemos del problema con $x_2 \leq 1$ la variable, con el mayor costo de penalización que es x_1, entonces planteamos los dos nuevos problemas: con $x_1 \leq 3$ y $x_1 \geq 4$ y se tiene 3. Sea $x_1 \leq 3$ cuyo valor óptimo de la función objetivo es :

$$z^* = 16.8$$

Y los valores de las variables:

$x_1 = 3$
$x_2 = 0.90$
$x_3 = 1.20$
$x_4 = 1.10$
$x_5 = x_7 = 0$
$x_6 = 0.10$

3. Con $x_1 \geq 4$, no hay solución factible
4.

Como en éste problema no hay solución factible entonces se tendría que ramificar sobre el problema 3, sin embargo los costos penalizados asociados son iguales a cero por lo que la solución óptima está dada por $z = 17$, $x_1 = 3$, $x_2 = 1$ En la figura 4.16 se muestra el árbol de búsqueda asociado a este problema.

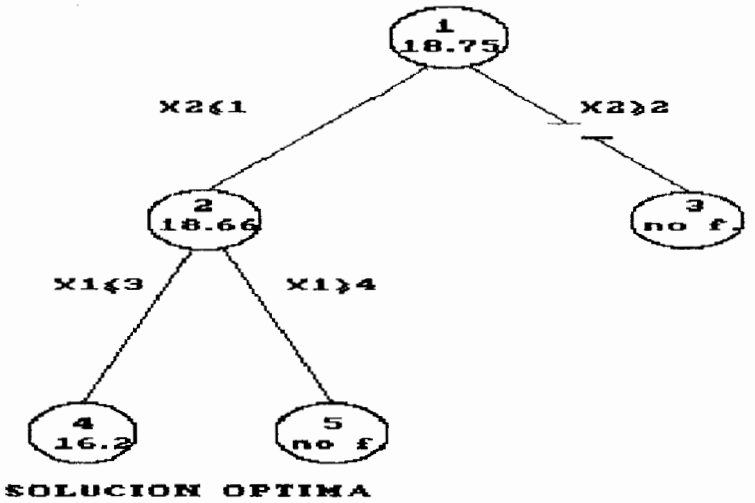


figura 4.16

Estos costos penalizados tienen la grandísima ventaja que permiten en cualquier instante del algoritmo de bifurcación y acotación (aunque no se haya obtenido aún la solución óptima), calcular una cota superior en el caso de maximización y una cota inferior en el caso de minimización entre la diferencia del valor de la función objetivo de la mejor solución entera obtenida hasta ese momento y el valor óptimo de la función objetivo. Esta cota se calcula de

$$\max_{r\in I, j\in N}(\min)\begin{cases}\dfrac{z_j - c_j}{Y_{rj}}(x_{Br} - [x_{Br}]) \text{ para } Y_{rj} > 0, \forall\ r, j \\ \dfrac{z_j - c_j}{Y_{rj}}(1 - x_{Br} + [x_{Br}]) \text{ para } Y_{rj} < 0\end{cases}$$

Donde I es el conjunto de todos los vectores básicos en la iteración en cuestión (cuando se para el algoritmo) y N es el conjunto de todos los vectores no-básicos.

EJEMPLO 4.6

Considere el siguiente problema entero

Mín $z = 7x_1 + 3x_2 + 4x_3$
sujeto a
$x_1 + 2x_2 + 3x_3 - x_4 = 8$
$3x_1 + x_2 + x_3 - x_5 = 5$
$x_1, x_2, x_3, x_4, x_5 \geq 0$, enteros

Se supone que después de ramificarse como lo indica la figura 4.17 se ha obtenido una solución entera y se para el algoritmo, se desea conocer cuan alejada se encuentra esa solución de la solución óptima. La tabla asociada al nodo 2 es:

	x_1	x_2	x_3	x_4	x_5	x_B
z	5	1	0	0	4	-17
x_3	3	1	1	0	-1	2
x_4	8	1	0	1	-3	4

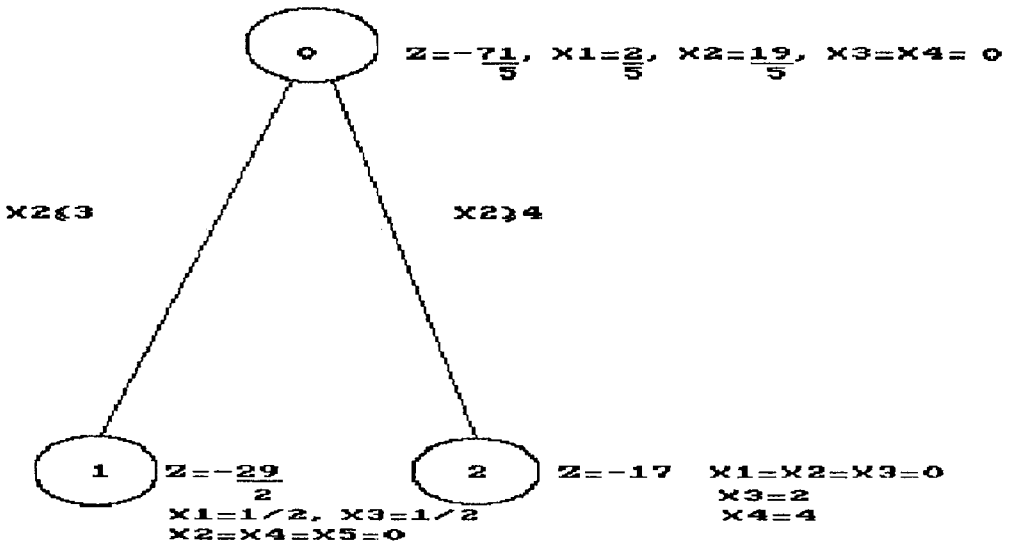


figura 4.17

Calculando los costos penalizados $[CP]_r$ y $<CP>_r$, para toda variable básica (sin incluir a las de holgura) (x_3) se tiene

max {min {(2-2)(5/3), (2-2)(1/1) }, min{(1-2+2)(4/-(-1))}} = max{min (0,0), Min{4}} = max{0,4} = 4

Lo que quiere decir que la función objetivo óptima no puede tener un valor mayor a -17+4 = -13 (puesto que se está minimizando se obtiene una cota inferior, en el caso de maximización se obtiene una cota superior, dada por el valor de la función objetivo al momento de parar, menos el valor del costo penalizado). Es decir 13 es una cota inferior del valor óptimo de la función objetivo del problema entero. De hecho el valor óptimo es:

$z = 15$

con $x_2 = 5$

$x_4 = 2$

$x_1 = x_3 = x_5 = 0.$

4.7 NOTAS HISTÓRICAS

El algoritmo clásico de enumeración de Land y Doig publicado en 1960 tiene variaciones como el método para el problema del agente viajero presentado por

Little, Murty, Sweeney y Karel en 1965 fue el primero en nombrarlo como de Ramificación y Acotamiento. En 1964 Thompson presentó un algoritmo para programación entera "El Método Simplex de Paro" como consecuencia del trabajo de Land y Doig. Un año más tarde Dakin propuso una variación que aunque simple era muy interesante. Beale y Small extendieron el método de Dakin que incluía procedimientos de post-optimización sugeridos por Driebeek. Más tarde Tomlin describió una versión más refinada de dicho algoritmo.

Aunque el procedimiento de ramificación y acotamiento se ha aplicado a varias situaciones particulares, el desarrollo de un principio general se produjo como un sucesor de tales aplicaciones. Bertier y Roy (1965) fueron los primeros en presentar una teoría general para Ramificación y Acotamiento.

Balas (1968) reescribió esta teoría en forma más simple y más tarde Mitten (1970) generalizó y extendió un poco más el trabajo de Balas.

CAPÍTULO 5
ALGUNAS APLICACIONES DE RAMIFICACIÓN Y ACOTAMIENTO

"Tiresias me ordenó que recorriera muchísimas ciudades, llevando en la mano un manejable remo, hasta llegar a aquellos hombres que nunca vieron el mar, ni comen manjares sazonados con sal, ni conocen las naves de purpúreos flancos, ni tienen noticia de los manejables remos que son como las alas de los bajeles."
Homero. *La Odisea*.

5.1 INTRODUCCIÓN

En el pasado reciente, los algoritmos de ramificación y acotamiento se han formulado para resolver una amplia variedad de problemas de optimización combinatoria. Entendiendo por problema combinatorio a aquél que asigna valores numéricos discretos a algún conjunto finito de variables X, de tal forma que satisfaga un conjunto de restricciones y minimice alguna función objetivo. Dos de tales problemas, tales como el del agente viajero y el de la mochila los proporcionamos como representativos no del rango de todas los posibles problemas combinatorios, pero si del tipo de problemas a donde se puede aplicar ramificación y acotamiento. La función objetivo que puede ser no lineal, discontinua o no necesariamente definida matemáticamente, se puede resolver con R-A.

Problemas como el del agente viajero o el de la mochila se conocen como no polinomiales completos o NP-completos. Un problema se dice polinomial si existe un algoritmo para el cual el tiempo requerido para su solución, está acotado por una función polinomial del tamaño del problema (donde entendemos el tamaño del problema como la longitud de un código, por ejemplo binario de los datos del problema). Se tiene así por ejemplo que en una gráfica G = [X, A] con N=X nodos y M = A arcos, una ruta más corta se encuentra a lo mas en un tiempo O(MN), un flujo máximo en un tiempo $O(N^3)$, un árbol de peso mínimo en un tiempo $O(N^2)$.

Un algoritmo se dice no determinístico si contiene afirmaciones que permiten una selección además de las afirmaciones usuales (determinísticas). La clase de problemas que se pueden resolver en tiempo polinomial por algoritmos no determinísticos se conocen como clase NP, en otras palabras, si para cada instrucción seleccionada, la selección es correcta, entonces el tiempo de computadora es polinomial. Por otro lado si se enumeran las posibles selecciones el algoritmo se vuelve determinístico pero con tiempo de computadora exponencial. Un problema NP-completo es aquel para el cual las únicas soluciones conocidas consisten, en esencia, en intentar todas las posibilidades. Tanto en el caso del problema del agente viajero como en el de la mochila los algoritmos que se basan en el método de R-A tienen un tiempo de computadora exponencial.

Este capítulo se desarrolla como sigue: en la segunda sección se describe el problema del agente viajero, se desarrolla el algoritmo de ramificación y acotamiento que se utiliza en la solución del problema del viajero y se presenta un ejemplo resuelto. También se presenta un algoritmo para el problema del agente viajero basado en el problema de asignación. En la tercera sección se describe el problema de la mochila, se expone el algoritmo de R-A desarrollado por Kolesar y se resuelve un ejemplo. En la cuarta sección se describen algunos juegos como el problema de las cuatro reinas, el juego del caballo y la torre de Hanoi cuyas soluciones se basan en la metodología de ramificación y acotamiento en general. Y finalmente en la última sección se dan algunas Notas Históricas.

5.2 EL PROBLEMA DEL AGENTE VIAJERO

El problema del agente viajero es un problema clásico en optimización combinatoria, y uno de los problemas más viejos en la optimización de redes que ha atraído mucho la atención en los últimos cuarenta años. Cuenta con numerosas aplicaciones, por ejemplo, en problemas de distribución donde se tiene que el almacén central de una compañía desea distribuir materia prima a cada una de sus sucursales a un costo mínimo o problemas de carteros, ¿ cómo debe el cartero atravesar la ciudad para repartir la correspondencia minimizando el tiempo de viaje?. O considere otra situación: una fábrica produce n artículos y se debe cambiar el montaje siempre que cambia la producción de artículos, conociendo el costo de montaje entre los artículos se desea encontrar una secuencia de producción óptima.

Plantear el problema es muy sencillo y lo haremos a continuación:

Un agente viajero debe visitar cada ciudad de su territorio exactamente una vez y regresar a su punto de partida. Dado el costo del viaje entre cada par de ciudades, ¿Cómo debe planear su itinerario de tal forma que visite cada ciudad una sola vez y el costo total del circuito sea mínimo?

En términos de la teoría de redes, el problema consiste en encontrar en una gráfica completa de n nodos, un circuito de peso mínimo de longitud n. Recuérdese que una gráfica completa es aquella en la cual entre cualquier par de nodos existe un arco que los une. Considere por ejemplo, un problema con cinco ciudades, como se muestra en la figura 5.1

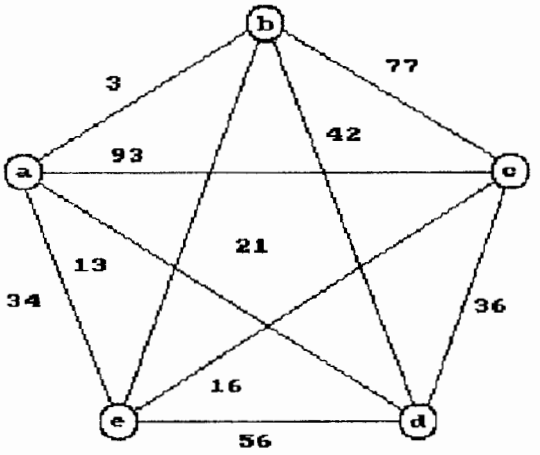


figura 5.1

El peso del circuito o costo de la ruta (a, b, c, d, e, a) es 206, mientras que el circuito de peso mínimo es (a, b, e, c, d, a) con peso 89.

Hay varias variantes a éste problema, pero primero comenzaremos por precisar conceptos:

Sea $W = [w_{ij}]$ el peso de la matriz de la gráfica, entonces se puede tener que los pesos en los arcos pueden no ser simétricos $w_{ij} = w_{ji}$, en cuyo caso estamos tratando con una gráfica dirigida y resolviendo un problema del agente viajero asimétrico. O algunas veces la gráfica dada no es completa, lo que significa que existen parejas de nodos que no están directamente conectados por arcos. No obstante, que toda gráfica completa siempre tiene circuitos de longitud n, una que no lo es puede no tener circuitos de longitud n. Por ejemplo la gráfica en la figura 5.2 no tiene circuitos de longitud 5, en tal caso el problema del agente viajero no tiene solución.

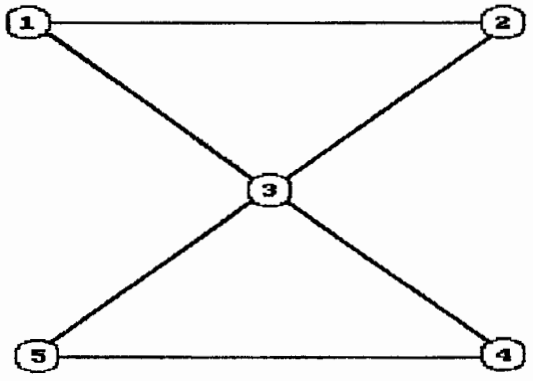


figura 5.2

El problema de determinar si una gráfica dada con n nodos tiene un circuito de longitud n se conoce como un problema de circuito hamiltoniano, originado en un juego inventado por el matemático irlandés Sir William Rowan Hamilton en 1859. Trataba sobre un viaje alrededor del mundo, el cual se representaba en forma simplificada por un dodecaedro (poliedro de 12 caras pentagonales con 20 vértices), y se requiere que se pase una sola vez por cada vértice o ciudad, usando solamente las caras del dodecaedro y se regrese al punto inicial. Este juego es equivalente a buscar un circuito hamiltoniano en la figura 5.3

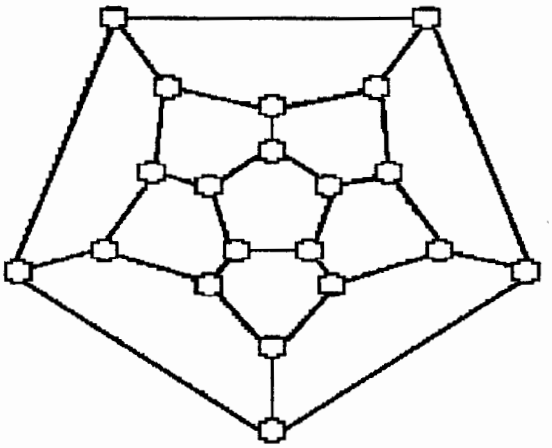


figura 5.3

Un circuito que pasa por cada nodo en la gráfica es llamado un circuito hamiltoniano. Si el agente viajero no quiere regresar al nodo inicial, después de visitar todos los nodos en la gráfica exactamente una vez, el problema se convierte entonces en encontrar la trayectoria de peso mínimo de longitud (n-1) y no un circuito de longitud n.

Se podrá notar un cambio muy drástico si al resolver éste problema se permite que el agente viajero visite un nodo mas de una vez (si esto es más barato). Por ejemplo en la figura 11 se tiene que si se permite que visite un nodo mas de una vez la ruta óptima (a, b, c, d, e, c, a) que además no es un circuito tiene un peso de 60, pero si no se permite que visite un nodo mas de una vez, la ruta (a, b, c, d, e, a) es la única solución posible, con un peso de 140.

La condición bajo la cual al menos uno de los circuitos hamiltonianos tenga un circuito de peso mínimo es que los pesos satisfagan la desigualdad del triángulo $w_{ij} \leq w_{ik} + w_{kj}$.

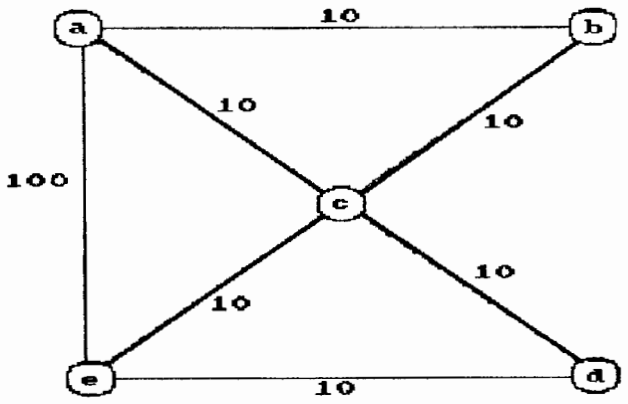


figura 5.4

Una forma de resolver el problema del agente viajero es haciendo una enumeración exhaustiva de todas las soluciones posibles y entre ellas escoger la mejor. Hay n! permutaciones de los n nodos, pero solo (n-1)! de ellas son circuitos hamiltonianos distintos (en una gráfica dirigida completa), porque podemos fijar alguno de los n nodos, y posteriormente hay (n-1)! formas de acomodar los (n-1) nodos restantes en el circuito. En caso de que la red dada no sea una red dirigida entonces hay (n-1)!/2 circuitos distintos.

Teóricamente, el problema se puede resolver generando los (n-1)! circuitos y comparando sus pesos, sin embargo como método éste es muy ineficiente. Por ejemplo, dada una gráfica de 20 nodos se tienen $19! > 10^{17}$ circuitos, por lo que se requieren años de cálculos continuos para resolver el problema.

Como se podrá ver el problema del agente viajero es uno de los problemas de optimización combinatoria para los cuales no se conoce un algoritmo eficiente de tiempo polinomial. Todos los algoritmos conocidos requieren de un tiempo de computadora exponencial en el número n de ciudades.

Para salvar esta dificultad computacional hay dos formas de resolverla:

1. Usando técnicas refinadas tales como ramificación y acotamiento o programación dinámica, que reducen drásticamente el efecto que se da en enumeración exhaustiva. Tales técnicas refinadas enumerativas son buenas para encontrar una solución óptima, pero en el peor de los casos si se tiene un problema muy grande pueden requerir un número exponencial de cálculos que se hacen prohibitivamente grandes.
2. Empleando métodos de solución aproximados pero rápidos (en tiempo polinomial), que no producen una solución óptima, pero si soluciones subóptimas que estén aceptablemente cercanas a la óptima.

Ambos métodos son útiles, sin embargo para nuestros propósitos sólo discutiremos el caso de ramificación y acotamiento.

ALGORITMO DE RAMIFICACIÓN Y ACOTAMIENTO

Un método de ramificación y acotamiento para un problema en particular queda definido al especificar sus operaciones de R-A. En el caso del problema del viajero, existen tres métodos de ramificación y acotamiento

1. Construcción del circuito según la matriz reducida.
2. Eliminación de subcircuitos a partir de la solución de problemas de asignación.
3. Construir la trayectoria basándose en el árbol de expansión mínima.

Usaremos el primer método para resolver el problema. El funcionamiento de éste método se basa en la construcción del circuito hamiltoniano según la matriz reducida.

ALGORITMO LITTLE-MURTY

PROPÓSITO: Determinar el circuito hamiltoniano de costo mínimo dada una matriz de costos. El método utiliza la propiedad de la matriz de costos reducida para probar la inclusión o exclusión de un arco en el circuito.

DESCRIPCIÓN

Paso 1 Dada la matriz de costos D, se efectúan sustracciones en los renglones y las columnas de la matriz D, sin permitir que aparezcan valores negativos. Con esto obtenemos una cota inferior del problema del viajero al sumar los elementos que se restaron a los renglones y a las columnas. La matriz D' es la matriz reducida de D.

Paso 2 Sea S un nodo del árbol y ev(S) la cota inferior de éste nodo S, con cada arco (i, j) con $d_{ij}' = 0$, debemos usar algún arco comenzando en i y penalizarlo por α_i. También debemos usar algún arco en j, y podríamos penalizarlo por $ß_j$, con lo cual $\Theta_{ij} = \alpha_i + ß_j$ es la penalización de no escoger (i,j). Se selecciona el arco que tiene el máximo de los Θ_{ij}

Paso 3 Si el arco (i, j) no se selecciona, ev (S) + Θ_{ij} es una cota inferior. Si el arco (i, j) se seleccionó, entonces la matriz se reduce omitiendo el renglón i y la columna j. Buscar la condición adicional sobre D' para excluir subcircuitos posibles.

Paso 4 Seleccione el vértice que tiene menor costo y vaya al paso 1.

En el ejemplo siguiente se ilustra este algoritmo.

EJEMPLO 5.1

Sea la matriz de tiempos D que se presenta en la Tabla 5.1, entonces teniendo el tiempo de duración de cada uno de los viajes entre las distintas paradas, se desea encontrar el circuito hamiltoniano de menor duración.

Tabla 5.1

	A	B	C	D	E	F
A	∞	27	43	16	30	26
B	7	∞	16	1	30	25
C	20	13	∞	35	5	0
D	21	16	25	∞	18	18
E	12	46	27	28	∞	5
F	23	5	5	9	5	∞

PASO 1

Si T es la duración de un recorrido hamiltoniano asociado a la matriz de tiempos de la tabla 5.1 entonces la duración de ese mismo circuito con la matriz de tiempos obtenida de restar un escalar h_i al renglón i está dado por $T-h_i$, porque cada circuito contiene uno y solo un elemento de éste renglón. Lo mismo sucede si restamos un escalar h^j de una columna j (j = A, B, C, D, E, F). Una cota inferior del circuito hamiltoniano óptimo es sencilla de obtener si restamos de cada renglón de la matriz de tiempos D, el mínimo elemento correspondiente. O bien

$$h_i = \min d_{ij} \geq 0$$

Tabla 5.2

	A	B	C	D	E	F
A	∞	11	27	0	14	10
B	6	∞	15	0	29	24
C	20	13	∞	35	5	0
D	5	0	9	∞	2	2
E	7	41	22	23	∞	0
F	18	0	0	4	0	∞

Y entonces, en la matriz D' asi reducida, restamos de cada columna j la constante

$$h^j = \min_j d_{ij}^1 \geq 0$$

La nueva matriz D^1, tiene elementos no-negativos, y contiene al menos un elemento cero en cada renglón y cada columna. Como se muestra en la tabla 5.3

Tabla 5.3

	A	B	C	D	E	F
A	∞	11	27	0	14	10
B	1	∞	15	0	29	24
C	15	13	∞	35	5	0
D	0	0	9	∞	2	2
E	2	41	22	23	∞	0
F	13	0	0	4	0	∞

Tomamos:

$$h = \sum_i h_i + \sum_j h_j$$

La duración z(t) de un circuito hamiltoniano t de la matriz D es entonces igual a z(t) = h + z'(t), donde z'(t) es la duración del circuito t de la matríz D'. Ya que z'(t) (porque $d_{ij}^1 \geq 0$), entonces tenemos $z(t) \geq h$ donde h es una evaluación por cota inferior del conjunto Ω de circuitos hamiltonianos y se denota ev(Ω).

De la matriz dada D^1 obtenemos

$h_1 = 16$, $h_2 = 1$, $h_3 = 0$, $h_4 = 16$, $h_5 = 5$, $h_6 = 5$
$h^1 = 5$, $h^2 = h^3 = h^4 = h^5 = h^6 = 0$

Un aspecto importante es que cada circuito hamiltoniano asociado a la Tabla 5.3 tiene una duración no negativa y que difiere en tiempo del circuito original en 48 unidades de tiempo. De donde una cota inferior de la duración del circuito mínimo es 48 o bien:

$$ev(\Omega) = \Sigma h_i + \Sigma h^j = 16 + 1 + 16 + 5 + 5 + 5 = 48$$

PASO 2

Considerando todas las entradas de la matriz de la tabla 5.3 iguales a cero, calculamos los retardos o penalizaciones por no usar alguno de esos arcos que prometen un circuito de menor costo, ya que fueron los que al restar los mínimos en renglones y columnas quedaron igual a cero.

Calculamos el retardo Θ_{ij} correspondiente a $d_{ij}' = 0$

$\Theta_{AD} = \alpha_A + ß_D = 10 + 0 = 10$
$\Theta_{BD} = \alpha_B + ß_D = 1 + 0 = 1$
$\Theta_{CF} = \alpha_C + ß_F = 5 + 0 = 5$
$\Theta_{DA} = \alpha_D + ß_A = 0 + 1 = 1$
$\Theta_{DB} = \alpha_D + ß_B = 0 + 0 = 0$
$\Theta_{EF} = \alpha_E + ß_F = 2 + 0 = 2$
$\Theta_{FB} = \alpha_F + ß_B = 0 + 0 = 0$
$\Theta_{FC} = \alpha_F + ß_C = 0 + 9 = 9$
$\Theta_{FE} = \alpha_F + ß_E = 0 + 2 = 2$

En general un arco (i,j) con d_{ij}'= 0 no se escoge. Ya que debemos dejar el punto i, debemos usar algún arco comenzando en i y penalizarlo por α_i. También debemos usar algún arco en j, y podríamos penalizarlo por $ß_j$, con lo cual $\Theta_{ij} = \alpha_i + ß_j$ es la penalización de no escoger (i, j). Dicha penalización Θ_{ij} se puede interpretar en éste caso como un retardo.

Si ev (S) es la evaluación obtenida reduciendo la matriz por el nodo S, entonces ev (S) + Θ_{ij} es una primer evaluación por cota inferior del nodo ij, obtenido de S por no escoger el arco (i, j)

PASO 3
Separamos la pareja (i, j) que maximiza el retardo Θ_{ij} para garantizar una mayor cota inferior de la duración de los circuitos hamiltonianos:$\Theta_{AD} = 10$

Entonces el nodo AD tiene una evaluación por cota inferior igual a 48+10=58

Una manera de proceder a la determinación del circuito hamiltoniano de mínima duración consiste en particionar el conjunto de circuitos hamiltonianos como sigue:

a) Circuitos hamiltonianos que usan el arco AD
b) Circuitos hamiltonianos que no usan el arco AD

En el primer caso la matriz de tiempos entre localidades se reduce a una nueva matriz en donde se elimina al renglón A y la columna D. Asimismo, el tiempo entre la localidad D a la A se hace igual a infinito o a un número muy grande para evitar usar el arco DA; pues sabemos que no forma parte del circuito hamiltoniano mínimo. Si no hacemos éste tiempo infinito, existe la posibilidad de la aparición de subcircuitos. La tabla 6.4 muestra la matriz reducida donde se han eliminado el renglón A y la columna D, el arco DA se hace igual a infinito

Tabla 5.4

	A	B	C	E	F
B	1	∞	15	29	24
C	15	13	∞	5	0
D	∞	0	9	2	2
E	2	41	22	∞	0
F	13	0	0	0	∞

Una cota inferior de la duración de los circuitos hamiltonianos asociados con esta tabla es sencilla de obtener si restamos una unidad a cada elemento del renglón uno como se muestra en la tabla 5.5

Tabla 5.5

	A	B	C	E	F
B	0	∞	14	28	23
C	15	13	∞	5	0
D	∞	0	9	2	2
E	2	41	22	∞	0
F	13	0	0	0	∞

PASO 4

Cabe hacer notar que como resultado de las manipulaciones anteriores podemos decir que los circuitos hamiltonianos asociados a la tabla 5.1, que usen el arco AD tienen una duración no menor de 49 unidades de tiempo, 48 acumuladas hasta la obtención de la tabla 5.3 y una unidad de tiempo al pasar de la tabla 5.4 a la tabla 5.5, por lo que 49 unidades representan una cota inferior a la duración de los circuitos hamiltonianos que usan el arco AD.

PASO 2

El siguiente paso consiste en calcular los retardos correspondientes a la tabla 5.5

$$\Theta_{BA} = \alpha_B + \beta_A = 14 + 2 = 16$$
$$\Theta_{CF} = \alpha_C + \beta_F = 5 + 0 = 5$$
$$\Theta_{DB} = \alpha_D + \beta_B = 2 + 0 = 2$$
$$\Theta_{EF} = \alpha_E + \beta_F = 2 + 0 = 2$$
$$\Theta_{FB} = \alpha_F + \beta_B = 0 + 0 = 0$$
$$\Theta_{FC} = \alpha_F + \beta_C = 0 + 9 = 9$$
$$\Theta_{FE} = \alpha_F + \beta_E = 0 + 2 = 2$$

Se escoge el arco BA para efectuar la ramificación como sigue:

a) Circuitos hamiltonianos que usan el arco BA
b) Circuitos hamiltonianos que no usan el arco BA

PASO 3

En el primer caso partimos de la tabla 5.5 eliminando el renglón B y la columna A y haciendo el tiempo de A a B igual a infinito para evitar circuitos innecesarios. En éste momento, como AD y BA se han seleccionado para formar el circuito hamiltoniano hacemos que el tiempo de DB sea infinito, la tabla 5.6 exhibe ésta situación

Tabla 5.6

	B	C	E	F
C	13	∞	5	0
D	∞	9	2	2
E	41	22	∞	0
F	0	0	0	∞

Con el propósito de tener un elemento cero en cada renglón y columna así como mejorar la cota inferior de los circuitos hamiltonianos procedemos a restar dos unidades del renglón dos en la tabla 5.6 para obtener:

Tabla 5.7

	B	C	E	F
C	13	∞	5	0
D	0	7	0	2
E	41	22	∞	0
F	0	0	0	∞

Y se puede observar que una cota inferior de los circuitos hamiltonianos que usan BA es 51 unidades de tiempo.

PASO 2

Nuevamente calculamos las penalizaciones para saber que arco es el que se va a usar:

$\Theta_{CF} = \alpha_C + ß_F = 5 + 0 = 5$
$\Theta_{DE} = \alpha_D + ß_E = 0 + 0 = 0$
$\Theta_{DF} = \alpha_D + ß_F = 0 + 0 = 0$
$\Theta_{EF} = \alpha_E + ß_F = 22 + 0 = 22$
$\Theta_{FB} = \alpha_F + ß_B = 0 + 13 = 13$
$\Theta_{FC} = \alpha_F + ß_C = 0 + 9 = 9$
$\Theta_{FE} = \alpha_F + ß_E = 0 + 2 = 2$

PASO 3

Como se tiene 22 para EF entonces se incluye al arco EF y si no se incluye se tiene una penalización de 51 + 22 = 73 y la tabla 5.8 exhibe el resultado

Tabla 5.8

	B	C	E
C	13	∞	5
D	∞	7	0
F	0	0	0

Nos fijamos en el primer renglón y restamos 5 para tener cero quedando

Tabla 5.9

	B	C	E
C	8	∞	0
D	∞	7	0
F	0	0	0

PASO 2

Lo que nos da un costo de 56 unidades nuevamente calculamos las penalizaciones:

$\Theta_{CE} = \alpha_C + \beta_E = 8 + 0 = 8$
$\Theta_{DE} = \alpha_D + \beta_E = 7 + 0 = 7$
$\Theta_{FB} = \alpha_F + \beta_B = 0 + 8 = 8$
$\Theta_{FC} = \alpha_F + \beta_C = 0 + 7 = 7$

PASO 3

De acuerdo a esto podemos elegir como arco a usar el CE o el FB, escogemos FB y obtenemos la tabla

Tabla 5.10

	C	E
C	∞	0
D	7	0

Restamos siete en el renglón de D lo que nos da un costo de 63 unidades por usar el arco FB y obtenemos:

Tabla 5.11

	C	E
C	∞	0
D	0	0

De donde los únicos arcos que quedan que se pueden usar son CE, DC y DE de donde escogemos CE y DC y asignamos infinito a DE para evitar posibles subcircuitos con lo que nos queda la tabla

Tabla 5.12

	C	E
C	∞	0
D	0	∞

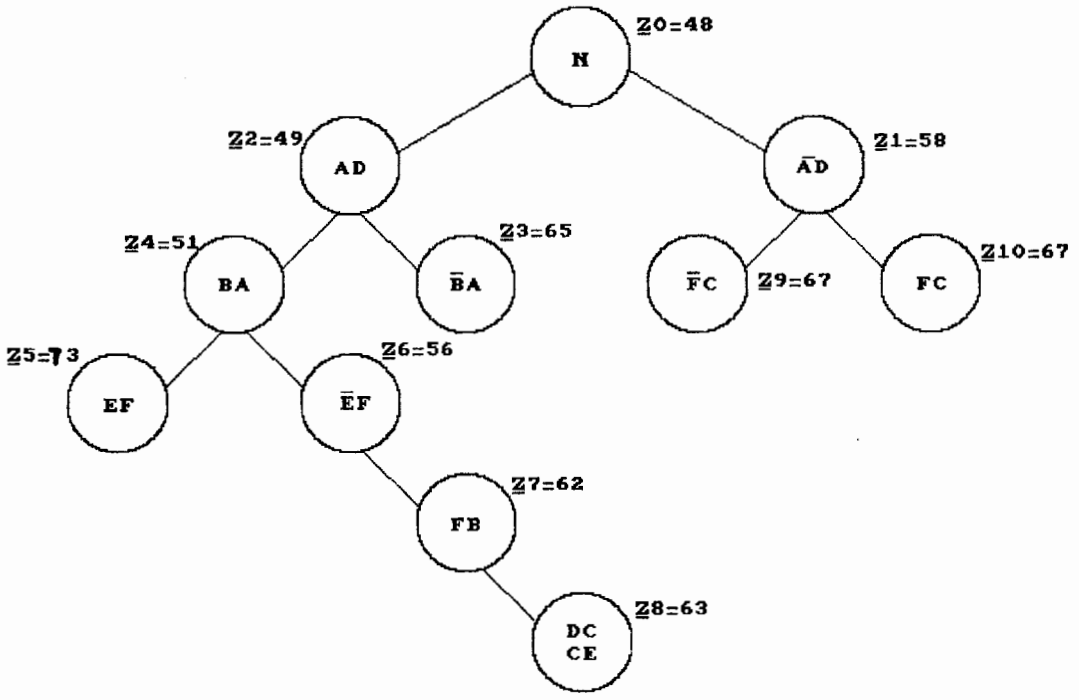


figura 5.5

De donde se concluye que el circuito hamiltoniano óptimo es A-D-C-E-F-B-A con una duración de 63 unidades de tiempo.

Para verificar que ésta solución es óptima, debemos examinar el vértice AD cuya evaluación por cota inferior es 58 (< 63). La separación del vértice AD produce: usando FC una cota de 63 y sin usar FC una cota de 67, por lo cual el circuito propuesto es óptimo.

ALGORITMO BASADO EN EL PROBLEMA DE ASIGNACION

El problema del agente viajero se puede escribir como un modelo de asignación de la siguiente manera:

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

Sujeto a

$$\sum_{j=1}^{n} xij = 1 \quad i = 1,\ldots, n$$

$$\sum_{i=1}^{n} xij = 1 \quad j = 1,\ldots, n$$

$$xij = 0 \text{ o } 1$$

Donde $c_{ij} = \infty$ para i = j, y $x_{ij} = 1$ si el agente viajero va de la ciudad i a la ciudad j.

Excepto por el requerimiento de que la solución sea un circuito hamiltoniano, la formulación es un modelo de asignación. Desafortunadamente no hay una garantía de que la solución óptima del modelo de asignación será un circuito hamiltoniano. Más bien la solución nos presentará una serie de subcircuitos a partir de los cuales podremos construir dicho circuito. Esto se logra a través de un algoritmo de ramificación y acotamiento, aunque note que el problema se convierte en resolver un problema de asignación en cada nodo del árbol de búsqueda.

El algoritmo que se describe a continuación está basado en el principio general de R-A, los únicos detalles para completar el algoritmo son los siguientes:

1. Se determinan cotas superiores e inferiores en la función objetivo óptima del problema del agente viajero, es decir, dado z^* el valor óptimo asociado al circuito, entonces z_L y z_U se determinan tales que $z_L \leq z^* \leq z_U$ y

2. Se especifica el procedimiento exacto para ramificar en cada nodo.

Inicialmente se hace $z_U = \infty$. Sin embargo, si el circuito es factible con $c_{12} + c_{23} + \ldots + c_{n1} < \infty$, entonces z_U es igual a este valor. El valor inicial de z_U se determina resolviendo el problema de asignación del problema original. Si z^0 es el valor óptimo de la función objetivo, entonces $z_L = z^0$ es una cota inferior inicial.

Naturalmente, si la solución asociada a z^0 es un circuito, el algoritmo termina. De otra manera la solución dada consiste en al menos dos subcircuitos. Se selecciona el subcircuito con el menor número de ciudades. Si ese subcircuito incluye las ciudades $i_1, i_2, \ldots, i_k$, entonces $x_{i1,i2} = x_{i2i3} = \ldots = x_{iki1} = 1$. La ramificación se diseña de tal forma que el problema de asignación asociado con los nodos subsecuentes que emanen del nodo actual se eliminarán del subcircuito. Esto se puede lograr haciendo una de las variables igual a cero. El algoritmo se describe a continuación:

ALGORITMO ASIGNACIÓN

PROPÓSITO: Resolver el problema del agente viajero con base en el problema de asignación.

DESCRIPCIÓN

En la iteración r-ésima, se tiene que z^r es la función objetivo óptima, como la cota inferior z* cambia con el nodo, z^r se definirá automáticamente como z_L

PASO 0. Determine z^0. Si la solución asociada es el circuito completo, pare. De otra manera, guarde $z_U = c_{12} + c_{23} + ... + c_{n1}$ y (1, 2,...,n,1) como el circuito asociado. Haga r = 0 y vaya al paso 1.

PASO 1 Seleccione un subcircuito asociado a z^r con el menor número de ciudades e inicialice tantas ramas como número de variables x_{ij} que son igual a uno en el subcircuito. Para la rama (i,j) defina una nueva matriz de costos que difiere de la primera en que $c_{ij} = \infty$. Haga r = r+1 y vaya al Paso 2.

PASO 2 Seleccione uno de los nodos que no se han ramificado. Si no queda ninguno, deténgase el recorrido asociado con zU es el óptimo. De otra manera vaya al paso 3.

PASO 3 Resuelva el problema de asignación asociado con el nodo seleccionado. De aquí pueden resultar tres casos:

i) Si $z^r \geq z_U$, , entonces el nodo actual se dice que se ha examinado ya que no puede dar un mejor circuito que el asociado con z_U. Haga r = r+1 y vaya al Paso 2.

ii) Si $z^r < z_U$ y la solución asociada es un circuito, entonces haga $z_U = z^r$ y guarde el recorrido o circuito asociado como el mejor disponible hasta este momento.

iii) Si $z^r < z_U$ pero la solución asociada no es un recorrido, entonces haga r =r+1 y vaya al Paso 1.

EJEMPLO 5.2

Considere el problema del agente viajero asociado a la siguiente matriz de costos.

	A	B	C	D	E
A	∞	2	0	6	1
B	1	∞	4	4	2
C	5	3	∞	1	5
D	4	7	2	∞	1
E	2	6	3	6	∞

El valor inicial es:

$z_U = c_{AB}+c_{BC}+c_{CD}+c_{DE}+c_{EA} = 10$ y el circuito asociado (A,B,C,D,E,A)

Como se puede ver en la figura 5.6

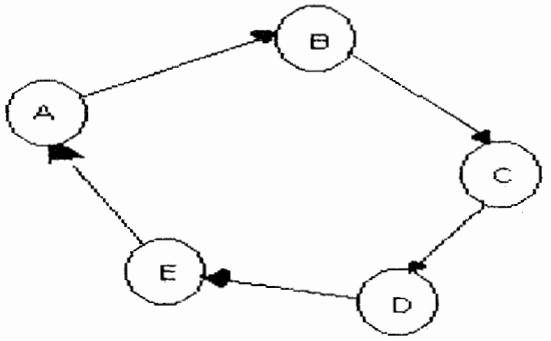


figura 5.6

Si se resuelve el problema de asignación usando LINDO se tiene:

```
MIN  2XAB + 6XAD + XAE +XBA + 4XBC + 4XBD + 2XBE + 5XCA + 3XCB + XCD + 5XCE + 4XDA
+ 7XDB + 2XDC + XDE + 2XEA + 6XEB + 3XEC + 6XED
ST
   XAB + XAC + XAD + XAE = 1
   XBA + XBC + XBD + XBE = 1
   XCA + XCB + XCD + XCE = 1
   XDA + XDB + XDC + XDE = 1
   XEA + XEB + XEC + XED  = 1
   XBA + XCA + XDA + XEA = 1
   XAB + XCB + XDB + XEB = 1
   XAC + XCB + XDC + XEC = 1
   XAD + XBD + XCD + XED = 1
   XAE + XBE + XCE + XDE  = 1
END
INT 20
```

Cuya solución está dada en el reporte siguiente:

OBJECTIVE FUNCTION VALUE

1) 8.000000

VARIABLE	VALUE	REDUCED COST
XAB	1.000000	2.000000
XAD	0.000000	6.000000
XAE	0.000000	1.000000
XBA	1.000000	1.000000
XBC	0.000000	4.000000
XBD	0.000000	4.000000
XBE	0.000000	2.000000
XCA	0.000000	5.000000
XCB	0.000000	3.000000
XCD	1.000000	1.000000
XCE	0.000000	5.000000
XDA	0.000000	4.000000
XDB	0.000000	7.000000
XDC	0.000000	2.000000
XDE	1.000000	1.000000
XEA	0.000000	2.000000
XEB	0.000000	6.000000
XEC	1.000000	3.000000
XED	0.000000	6.000000
XAC	0.000000	0.000000

Lo que da $z^0 = 8$ y la solución $x_{AB} = x_{BA} = 1$, $x_{CD} = x_{DE} = x_{EC} = 1$ que consiste de 2 sub-recorridos, como se puede ver en la figura 5.7

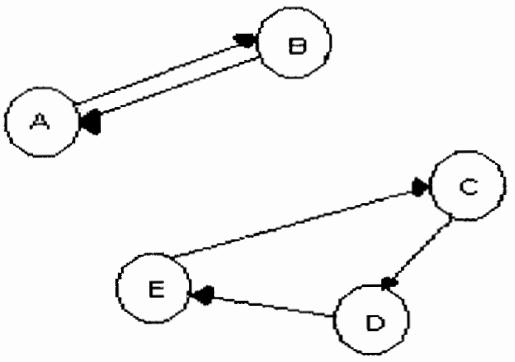


figura 5.7

Ya que el recorrido (A,B,A) tiene el menor número de ciudades, este se usa como ramificación. Se crean entonces los dos nodos correspondientes a $x_{AB}=0$ (o equivalentemente, $c_{AB} = \infty$, en la matriz C^0), y $x_{BA}=0$ (o equivalentemente $c_{BA} = \infty$, en la matriz C^0)

Seleccionamos el nodo asociado con $x_{AB} = 0$, la asignación nos da $z^1 = 9$ con un circuito (B, A, C, D, E, B). Así $z^U = 9$ y se guarda el circuito. Esto se puede ver en la figura 5.8.

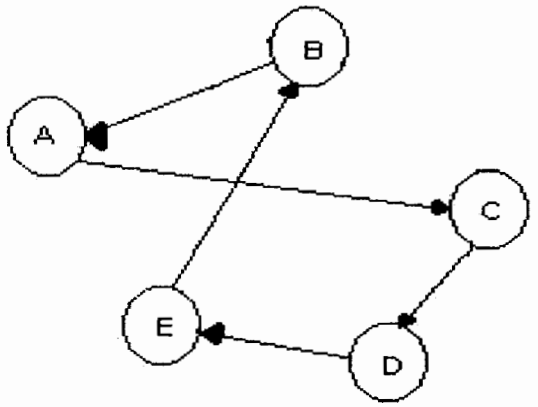


figura 5.8

Si se resuelve el problema de asignación usando LINDO obtenemos:

```
MIN  2XAB + 6XAD + XAE +XBA + 4XBC + 4XBD + 2XBE + 5XCA + 3XCB + XCD + 5XCE + 4XDA
+ 7XDB + 2XDC + XDE + 2XEA + 6XEB + 3XEC + 6XED
ST
   XAB + XAC + XAD + XAE = 1
   XBA +  XBC + XBD + XBE = 1
   XCA + XCB + XCD + XCE = 1
   XDA + XDB + XDC +  XDE = 1
   XEA + XEB + XEC + XED  = 1
   XBA + XCA + XDA + XEA = 1
   XAB + XCB + XDB + XEB = 1
   XAC + XCB + XDC + XEC = 1
   XAD + XBD + XCD + XED = 1
   XAE + XBE + XCE + XDE  = 1
   xAB = 0
END
INT 20
```

Y la solución:

```
     OBJECTIVE FUNCTION VALUE

     1)     9.000000

 VARIABLE        VALUE          REDUCED COST
     XAB        0.000000          2.000000
     XAD        0.000000          6.000000
     XAE        0.000000          1.000000
     XBA        1.000000          1.000000
     XBC        0.000000          4.000000
     XBD        0.000000          4.000000
     XBE        0.000000          2.000000
     XCA        0.000000          5.000000
     XCB        0.000000          3.000000
     XCD        1.000000          1.000000
     XCE        0.000000          5.000000
     XDA        0.000000          4.000000
     XDB        0.000000          7.000000
     XDC        0.000000          2.000000
     XDE        1.000000          1.000000
     XEA        0.000000          2.000000
     XEB        1.000000          6.000000
     XEC        0.000000          3.000000
     XED        0.000000          6.000000
     XAC        1.000000          0.000000
```

El nodo remanente $x_{BA} = 0$ donde se obtiene C^2 de C^0 haciendo $c^{21} = \infty$, nos da $z^2=9$. Como se puede ver de LINDO:

```
MIN  2XAB + 6XAD + XAE +XBA + 4XBC + 4X24 + 2XBE + 5XCA + 3XCB + XCD + 5XCE + 4XDA
+ 7XDB + 2XDC + XDE + 2XEA + 6XEB + 3XEC + 6XED
ST
   XAB + XAC + XAD + XAE = 1
   XBA +  XBC + XBD + XBE = 1
   XCA + XCB + XCD + XCE = 1
   XDA + XDB + XDC +  XDE = 1
   XEA + XEB + XEC + XED  = 1
   XBA + XCA + XDA + XEA = 1
   XAB + XCB + XDB + XEB = 1
   XAC + XCB + XDC + XEC = 1
   XAD + XBD + XCD + XED = 1
   XAE + XBE + XCE + XDE  = 1
   xBA = 0
END
INT 20
```

Y la solución:

```
     OBJECTIVE FUNCTION VALUE

     1)    9.000000

 VARIABLE      VALUE        REDUCED COST
    XAB       1.000000       2.000000
    XAD       0.000000       6.000000
    XAE       0.000000       1.000000
    XBA       0.000000       1.000000
    XBC       0.000000       4.000000
    XBD       0.000000       4.000000
    XBE       1.000000       2.000000
    XCA       0.000000       5.000000
    XCB       0.000000       3.000000
    XCD       1.000000       1.000000
    XCE       0.000000       5.000000
    XDA       0.000000       4.000000
    XDB       0.000000       7.000000
    XDC       1.000000       2.000000
    XDE       0.000000       1.000000
    XEA       1.000000       2.000000
    XEB       0.000000       6.000000
    XEC       0.000000       3.000000
    XED       0.000000       6.000000
    XAC       0.000000       0.000000
```

Con dos subcircuitos (A,B,E,A) y (C,D,C). Como $z_2 = z^U$, el nodo 2 no puede producir una mejor solución y la búsqueda termina con la solución óptima (B,A,C,D,E,B) con $z^*=9$. Como se puede ver en la figura 5.9

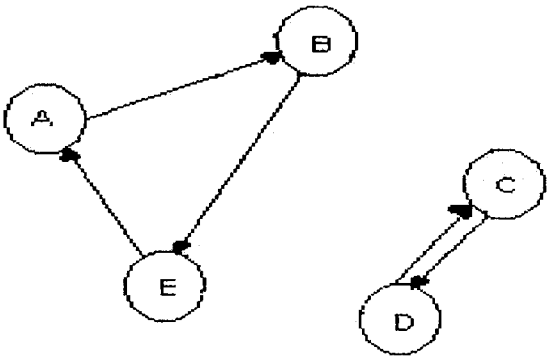


figura 5.9

El árbol de soluciones está en la figura 5.10

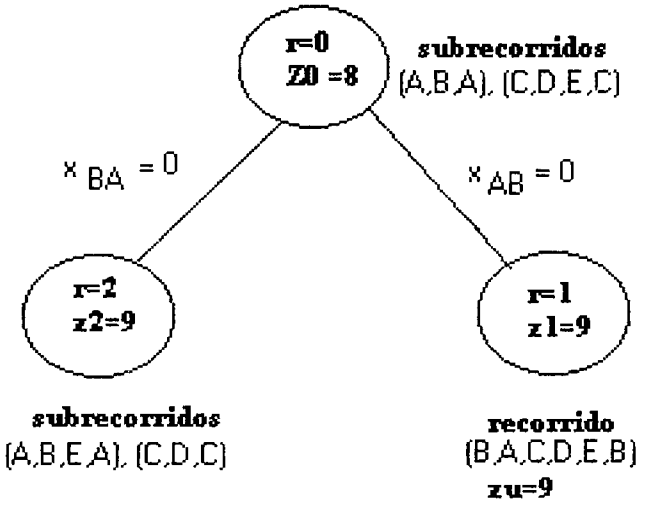


figura 5.10

5.3 EL PROBLEMA DE LA MOCHILA

Muchos problemas industriales se pueden formular como problemas de tipo mochila, por ejemplo problemas de cargo fijo, selección de proyectos, corte en inventarios, control de presupuestos, etc. La versión más popular del problema contiene sólo una restricción lineal, pero casi cualquier problema lineal entero y muchos otros problemas combinatorios se pueden reducir a él. El problema de la mochila se presenta también como un subproblema en varios algoritmos de programación lineal pura y entera.

Hay muchas versiones distintas del problema de la mochila, en nuestro caso consideraremos el problema de la mochila 0-1 que se expresa de la siguiente manera:

$$\max \quad z = \sum_{i=1}^{n} p_i x_i$$

sujeto a

$$\sum_{i=1}^{n} w_i x_i \geq W$$

$$x_i = 0,1$$

$$i = 1,2,..., n$$

Donde p_i, wi (i=1,2,...,n) y W son números enteros. En otros términos, suponga que se tiene que llenar una mochila con diferentes objetos con un beneficio p_i y peso w_i sin exceder un peso total dado W. El problema consiste en encontrar una asignación factible de objetos para que el valor total de los objetos en la mochila sea el máximo.

El problema de la mochila 0-1 es un caso especial del problema de la mochila acotado, que se define igual que el anterior y solo difiere en la restricción (5.3) ya que en éste caso se tiene $0 \leq x_i \leq b_i$, donde x_i es entero, i=1,2,...,n

En el problema de la mochila acotado, la mochila se puede llenar con a lo más b_i objetos del tipo i. En el problema general de la mochila, que a veces se denomina no acotado, la restricción (5.3) se relaja a $x_i \geq 0$, x_i entero, i=1,2,...,n

Sin pérdida de generalidad podemos suponer que los parámetros p_i, w_i y W en los problemas anteriores satisfacen las condiciones:

p_i y w_i son enteros positivos, con $w_i \leq W$ $\quad i = 1, 2, ..., n$, y

$$\sum_{i=1}^{n} w_i x_i \geq W$$

Los problemas de la mochila 0-1, acotado y generalizado a veces se conocen como problemas unidimensionales, donde el uno se refiere al número de restricciones lineales en el problema. Los más populares son los de valor independiente (cuando $w_i=p_i$) y el problema de hacer cambios, éste problema consiste en encontrar el menor número de monedas de tipos o valores especificados w_i que constituyen exactamente un cambio dado V. Suponemos que se dispone de cada tipo de moneda en una cantidad ilimitada, formalmente el problema es

$$\min \quad z = \sum_{i=1}^{n} x_i$$

sujeto a

$$\sum_{i=1}^{n} x_i w_i = W$$

$$x_i \geq 0$$

$$x_i \text{ entero} \qquad i = 1,\ldots, \quad n$$

Observe que como la restricción en éste problema es de igualdad no siempre existe solución, a menos que alguna de las monedas disponibles valga 1.

Los problemas tipo mochila unidimensionales se pueden generalizar de muchas maneras, la generalización más natural es aquella en que los objetos que tenemos que guardar pueden ponerse en m mochilas, cada una con capacidad W_j (j=1,2,...,m). Sea x_{ij} una variable 0-1 tal que $x_{ij}=1$ si el i-ésimo objeto se asigna a la j-ésima mochila. El problema 0-1 multimochila, se expresa como:

$$\max z = \sum_{i=1}^{n}\sum_{j=1}^{m} p_i x_{ij}$$

sujeto a

$$\sum_{i=1}^{n} w_i x_{ij} \leq W_j \quad j = 1,\ldots, m$$

$$\sum_{j=1}^{m} xij \leq 1 \quad i = 1,\ldots, n$$

$$x_{ij} = 0,1 \quad i = 1,\ldots, n; \quad j = 1,\ldots, m$$

La primer restricción significa que en una restricción factible de objetos no se sobrecarga ninguna mochila y la segunda, que cada objeto puede asignarse a lo más a una mochila, pueden formularse de aquí las versiones acotada y no acotada de éste problema.

Si antes de poner los objetos en una mochila se tienen que comprar, a un costo ci para el i-ésimo objeto, y se tiene una cantidad limitada de dinero C, se tiene

entonces el problema de asignar objetos a la mochila que no pesen mas de W y no cuesten mas de C, entonces el problema se convierte en

$$\max z = \sum_{i=1}^{n} p_i x_{ij}$$

sujeto a

$$\sum_{i=1}^{n} w_i x_{ij} \leq W_j \quad j = 1,..., m \quad i = 1,..., n$$

$$\sum_{i=1}^{n} c_i x_i \leq C$$

$$x_{ij} = 0,1 \quad i = 1,..., n; \quad j = 1,..., m$$

En general, se pueden introducir muchas restricciones al asignar objetos a una mochila, el problema se convierte entonces en un problema de la mochila multidimensional, donde evidentemente se pueden considerar los casos acotado y no acotado.

Los problemas de tipo mochila a menudo se refieren como problemas de cargo, pero de hecho, el problema de cargo estándar consiste en asignar objetos dados con volúmenes conocidos a cajas, que tienen restricciones de capacidad, con el objeto de minimizar el número de cajas usadas. Sea k_j la capacidad de la j-ésima caja y w_i el volumen del i-ésimo objeto. El problema de cargo se define como sigue:

$$\min z = \sum_{j=1}^{m} y_j$$

sujeto a

$$\sum_{i=1}^{n} w_i x_{ij} \leq k_j y_j \quad j = 1,..., m$$

$$\sum_{j=1}^{m} xij = 1 \quad i = 1,..., n$$

$$x_{ij}, y_j = 0,1 \quad i = 1,..., n; \quad j = 1,..., m$$

En un cargo factible, tenemos x_{ij}=1 si el i-ésimo objeto se pone en la j-ésima caja, y y_j=1 si se usa la j-ésima caja.

El problema de la mochila acotado o no acotado puede también representar el problema de cortar objetos unidimensionales (por ejemplo la longitud de un papel, vidrio y acero) en piezas pequeñas de valores y tamaños dados para maximizar el valor total de las piezas o minimizar el material que sobra.

Este pequeño panorama de las posibles generalizaciones y modificaciones del problema de la mochila 0-1 indica la variedad de problemas del mundo real que se pueden modelar por problemas que provienen de la familia de problemas de tipo mochila.

REDUCCION DEL PROBLEMA LINEAL ENTERO AL PROBLEMA DE LA MOCHILA.

Cada sistema de ecuaciones lineales con coeficientes enteros se puede transformar en una sola ecuación lineal que tiene el mismo conjunto de soluciones enteras no negativas que el correspondiente sistema lineal entero. Entonces, las restricciones de un problema lineal entero se pueden primero transformar en una única restricción y así resolverlo como un problema de tipo mochila.

El paso básico del proceso de transformación agrega dos ecuaciones de tal forma que el conjunto de soluciones enteras no negativas no se altera. Describiremos un método que supone todas las variables acotadas.

Sean las dos ecuaciones enteras en variables acotadas

$$g_j(x) = b_j - \sum_{j=1}^{n} a_{ji}x_i = 0 \qquad j = 1,2... \tag{5.4}$$

$$0 \le x_i \le u_i,\ x_i \text{ es entero} \quad i = 1,2,...,n$$

Determinaremos ahora los multiplicadores válidos α_1 y α_2 tales que son distintos de cero y la ecuación

$$\alpha_1 g_1(x) + \alpha_2 g_2(x) = 0 \tag{5.5}$$

implica que:

$$g_1(x) = g_2(x) = 0$$

Ya que (5.5) es equivalente a $g_1(x) + (\alpha_2 / \alpha_1) g_2(x) = 0$, podemos preguntar por sólo un multiplicador α tal que:

$$g_1(x) + \alpha g_2(x) = 0 \tag{5.6}$$

implica que:

$$g_1(x) = g_2(x) = 0$$

Usando cotas superiores en las variables, $g_j(x)$ (j=1,2) se pueden acotar de la siguiente manera:

$$b_j - \sum_{j=1}^{n} a_{ji}^{+} u_i \le g_j(x) \le b_j - \sum_{j=1}^{n} a_{ji}^{-} u_i$$

donde $a_{ji}^{+} = \max(a_{ji}, 0)$ y $a_{ji}^{-} = \min(a_{ji},0)$.

TEOREMA 5.1

El vector entero x que satisface $0 \leq x \leq u$ es una solución de (5.4) si y sólo si es una solución de (5.6) con α que satisface.

$$\alpha > \max \{b_1 - a_{1i}^{-} u_i, -b_1 + a_{1i}^{+} u_i\}$$

EJEMPLO 5.3

Aplicaremos el Teorema 5.1 al siguiente sistema de ecuaciones:

$$\begin{aligned} 4x_1 - x_2 + 2x_3 + x_4 + x_5 &= 0 \\ x_1 - x_3 - x_4 - x_6 &= -1 \\ x_i = 0 \text{ o } 1 \qquad i &= 1,2,...,6 \end{aligned}$$

Por la enumeración de todos los posibles vectores 0-1 $(x_1,x_2,x_3,x_4,x_5,x_6)$ encontramos las únicas soluciones de éste sistema (0,1,0,1,0,0),(0,1,0,0,1,1) y (0,0,0,0,0,1). Primero tratamos con dos multiplicadores arbitrarios $\alpha_1 = 1$ y $\alpha_2 = 1$. La ecuación agregada es de la forma

$$5x_1 - x_2 + x_3 + x_5 - x_6 = -1$$

y se tienen las tres soluciones anteriores mas:

(0,1,0,0,0,0),(0,0,0,1,0,1),(0,1,1,0,0,1),(0,1,1,1,0,1) y (0,1,0,1,1,1).

Sin embargo aplicando el Teorema 5.1 se encuentra que α debe ser mayor que 8. Para $\alpha = 9$, la ecuación válida agregada es de la forma

$$13x_1 - x_2 - 7x_3 - 8x_4 + x_5 - 9x_6 = -9$$

Una vez que sabemos como agregar dos ecuaciones en una, podemos aplicar éste proceso iterativamente a un sistema de ecuaciones arbitrario. Este método es de uso limitado para programas lineales enteros debido al rápido crecimiento en los valores de los coeficientes conforme se van agregando restricciones.

MÉTODOS COMPUTACIONALES

Desde el punto de vista computacional, el problema de la mochila es un problema difícil. No se ha encontrado un algoritmo de tiempo polinomial para resolverlo, y es muy poco probable que tal algoritmo exista. Se han propuesto varias técnicas como métodos de solución para éstos problemas como las siguientes:

1. Métodos de Reducción y Aproximación (del tipo gradiente y Lagrangeano)
2. Métodos exactos
   a. Redes
   b. Programación dinámica
   c. Métodos de Enumeración (Ramificación y Acotamiento)

Los métodos del tipo gradiente generalizado son eficientes computacionalmente sólo cuando se requieren soluciones aproximadas.

Los algoritmos de programación dinámica son eficientes cuando el valor de W es pequeño, cuando éste crece tiende a ser ineficiente porque se requiere un almacenamiento excesivo, mientras que las técnicas de enumeración resultan menos afectadas por ésta desventaja. El caso de la solución a través de redes formulan el problema de la mochila como un problema de ruta más corta, resultan también ineficientes porque se producen redes de tamaño muy grande

En éste trabajo sólo analizaremos el problema a través de el método de ramificación y acotamiento como veremos a continuación.

MÉTODO DE RAMIFICACIÓN Y ACOTAMIENTO

El primer método de ramificación y acotamiento que se usó para resolver el problema de la mochila es debido a Peter J. Kolesar (1967), éste método usa la estrategia de búsqueda por primer amplitud; el gran uso de memoria de computadora y los requerimientos de tiempo de éste algoritmo se redujeron mucho con el método de Greenberg y Hegerich (1970) que usa la estrategia de primer profundidad. Posteriormente Horowitz y Sahni(1974) propusieron un procedimiento de ramificación y acotamiento altamente eficiente, basado en el esquema de Greenberg-Hegerich, este mismo algoritmo se obtuvo independientemente por Ahrens y Finke (1975), y se han presentado más perfeccionamientos por Barr y Ross (1975), Fayard y Plateau (1975), Zoltners (1978), Nauss (1976), Suhl (1977), Martello y Toth (1977).

ALGORITMO DE KOLESAR

El algoritmo que se propone es un algoritmo de ramificación y acotamiento en el sentido de Little, et al.

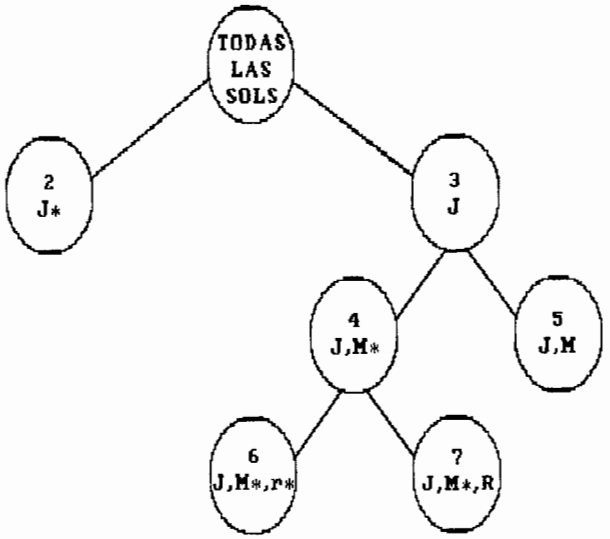


figura 5.11

Este método considera que un nodo lleva un índice i si el articulo se incluye y un índice i* si no se incluye. Un nodo con índice (i,j) significa que se incluye el artículo i primero y después el artículo j, mientras que el índice (i^*,j) significa que el artículo i no se incluye pero el j si como se muestra en la figura 5.11

Se denota por B(n) la cota superior asociada con el nodo n. Dado el nodo n tenemos tres categorías de artículos:

Los artículos incluidos In es el conjunto de artículos que están incluidos explícitamente en las soluciones contenidas en el nodo n; los artículos excluidos En como su nombre lo dice están explícitamente excluidos de las soluciones contenidas en el nodo n y finalmente los artículos no asignados son aquellos que no están incluidos en ninguna de las dos categorías anteriores, cuando el conjunto de artículos no asignados en un nodo es vacío, el nodo contiene una solución y ya no es posible ramificarlo más. En el mismo árbol de la figura 5.11, el nodo 1 representa la clase de todas las soluciones factibles, el nodo 2 representa todas las soluciones factibles que no incluyen al artículo j, mientras que el nodo 3 representa a las que si incluyen al artículo j, el nodo 7 representa a todas las que incluyen a j y r pero no a m. Los conjuntos In, En para todos los nodos de la figura 5.11 son:

$I_1 = \phi$	$E_1 = \phi$
$I_2 = \phi$	$E_2 = (j)$
$I_3 = (j)$	$E_3 = \phi$
$I_4 = (j)$	$E_4 = (m)$
$I_5 = (j,m)$	$E_5 = (m)$
$I_6 = (j)$	$E_6 = (m,r)$
$I_7 = (j,r)$	$E_7 = (m)$

El cálculo de cotas superiores se basa en dos observaciones. La primera es que la relajación de la restricción de indivisibilidad en uno o más de los artículos conlleva a un nuevo problema de optimización cuyo valor óptimo de la función objetivo no puede ser menor que el valor óptimo de la función objetivo para el problema original. La segunda observación es que para un problema de cargo en que todos los artículos son perfectamente divisibles, esto es:

$$\text{max imizar} \quad \sum_{j=1}^{m} p_i y_i \qquad (5.7)$$

Sujeto a

$$\sum_{j=1}^{m} y_i w_i \leq W \qquad (5.8)$$

$$0 \leq y_i \leq 1 \qquad (5.9)$$

Se obtiene una solución óptima arreglando los artículos en orden decreciente de magnitud de la razón p_i/w_i, y entonces, se comienza con el primer artículo de la lista, secuencialmente se va cargando cada artículo, hasta que se alcanza un peso límite W.

Aplicamos ahora éstos dos hechos al cálculo de cotas superiores. Supongamos que estamos en el nodo n, con los conjuntos In y En como se definieron anteriormente, antes de calcular una cota superior en el nodo n, probamos la factibilidad de la clase de soluciones contenidas en el nodo verificando si

$$\Sigma_{i \in In} w_i \leq W \qquad (5.10)$$

Los nodos que no satisfagan (5.10) no se incluyen en los cálculos. Para calcular la cota superior en el nodo n, decimos que es suficiente con resolver el siguiente problema:

$$\text{max imizar} \quad \sum_{i=1}^{n} p_i x_i$$

Sujeto a

$$\sum_{i=1}^{n} x_i w_i \leq W$$

$$x_i = 1 \quad \text{si} \quad i \in I_n$$
$$x_i = 0 \quad \text{si} \quad i \in E_n$$
$$0 \leq x_i \leq 1 \quad \text{si} \quad i \in (I_n \cup E_n) \qquad (5.11)$$

La solución de éste sistema es una cota superior en el nodo n aplicando la primer observación, ya que en (5.11) hemos relajado la restricción para los artículos no asignados en el nodo n, la segunda observación da una solución directa a éste sistema ya que por una simple transformación se puede poner en la forma (5.7),(5.8),(5.9). La solución está dada por:

a. Todos los artículos en $(I_n \cup E_n)'$ se ordenan decrecientemente p_i/w_i y se comienza con el primer artículo en la lista y se continua hasta que el peso total de los artículos sea igual a $W - \Sigma w_i$.
b. La cota superior en el nodo n, B(n) es igual al valor total cargado en a. más Σp_i.

Para desarrollar la operación de ramificación, se deben tomar dos decisiones. Primero se debe seleccionar el nodo terminal en el que se hace la siguiente ramificación, y segunda, se debe seleccionar el artículo que se deberá sumar a la lista de artículos incluidos y excluidos en los dos nodos resultantes, llamamos a éste al artículo "pivote". La selección del nodo a ramificar está dada por el método de ramificación y acotamiento, de asignar el nodo terminal al que tenga la mayor cota B(n). Por otro lado la selección del elemento pivote es arbitraria, se desea que la selección sea de tal forma que el algoritmo alcance una solución óptima

más rápidamente, resolviendo (5.7),(5.8),(5.9) se tiene una buena selección heurística. Suponiendo que los artículos originales están arreglados en orden decreciente de la razón p_i/w_i, seleccionamos el elemento pivote como la variable no asignada con el mayor p_i/w_i. En caso de haber empate se selecciona el primer artículo que aparece en la secuencia.

ALGORITMO KOLESAR

PROPÓSITO: Resolver el problema tipo mochila 0-1 usando ramificación y acotamiento.

DESCRIPCIÓN

PASO 1

1.1 Pruebe la factibilidad no trivial del problema verificando al menos uno de los índices i=1,2...,n.

$$w_i \leq W$$

si el problema es factible no trivial, siga a 1.2 si no, pare.

1.2 Si $\sum w_i \leq W$, entonces se pueden vaciar todos los artículos y el problema es trivial, si no vaya a 1.3

1.3 Ordene los artículos en orden decreciente de p_i/w_i, en lo que sigue se supondrá que los artículos están ordenados, vaya a 1.4

1.4 Para el nodo 1 haga B(1)=0, $I_i=\phi$, $E_i = \phi$ y vaya al paso 2

PASO 2 Selección del nodo para ramificar

2.1 Encuentre el nodo terminal con el mayor valor de B(n). Este es el nodo desde el cual se va a ramificar.

2.2 Pruebe si el nodo actual k, $(I_k \cup E_k)' = \phi$ si es cierto, hay una solución óptima dada por los índices contenidos en I_k. Si no seleccione el nuevo artículo pivote i^* por $i^* = i$ tal que p_i/w_i es máximo para $i \in (I_k \cup E_k)'$, y vaya al paso 3.

PASO 3 Cálculo de las cotas superiores.

3.1 Si n = n+1, $I_n = I_k$, $E_n = E_k \cup (i^*)$ vaya a 3.3

3.2 Si n = n+1, $I_n = I_k \cup (i^*)$, $E_n = E_k$ vaya a 3.3

3.3 Pruebe la factibilidad de las soluciones contenidas en el nodo n verificando si:

$$\sum_{i \in I_n} w_i \leq W,$$

si la prueba falla haga B(n) = -999, de otra forma proceda a calcular la cota superior metiendo todos los artículos en el conjunto In y procediendo entonces en forma ordenada i=1,2,...,n,cargando tantos artículos como sea posible del conjunto $(I_n \cup E_n)'$ hasta que el peso total cargado sea igual a W. El valor total es B(n). Pruebe si el nodo con índice n es par. Si lo es proceda con 3.2 si no vaya al Paso 2.

EJEMPLO 5.4

Considere el siguiente problema tipo mochila con siete artículos cuyos pesos y valores se dan a continuación:

ARTICULO No.	PESO	VALOR
1	40	40
2	50	60
3	30	10
4	10	10
5	10	3
6	40	20
7	30	60

El peso total permitido es W = 100. Se desea maximizar el valor de los artículos que se van a llevar en la mochila.

Una prueba preliminar revela que el problema posee una solución factible y no es trivial, $\sum w_i > 100$. Calculamos las razones p_i/w_i y reordenamos los artículos, como se muestra a continuación con un nuevo índice:

NUEVO INDICE	ARTICULO No.	PESO	VALOR	RAZON
1	7	30	60	2
2	2	50	60	6/5
3	1	40	40	1
4	4	10	10	1
5	6	40	20	1/2
6	3	30	10	1/3
7	5	10	3	3/10

El primer nodo es el que incluye todas las soluciones posibles, y la primer ramificación usa el índice 1 como primer pivote, y en el nodo 2 se excluye este índice de la solución, y la cota superior es igual a :

$$B(2)= p_2 + p_3 + p_4 = 110$$

Mientras en el nodo 3 donde éste índice se incluye tenemos:

$$B(3)= p_1 + p_2 + \tfrac{1}{2} p_3 = 140$$

A continuación se desarrollan los cálculos para cada nodo del árbol: cuando sobra peso se resta el peso sobrante, y en la columna del valor se resta el mismo peso multiplicado por la razón de valor/peso.

B(2) no incluye el artículo 1

Índice	peso	Valor
2	50	60
3	40	40
4	10	10
total	100	110

la mejor cota es de 140 por lo que se procede a ramificar desde ahí:

B(4) incluye (1,2*)

Índice	peso	Valor
1	30	60
3	40	40
4	10	10
5	40	20
Total	120	130
	-20	-10
	100	120

la mejor cota es 140 por lo que se procede a ramificar desde ahí:

B(6) incluye (1,2,3*)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	130	150
	-30	-15
	100	135

B(9) incluye (1,2,3*,4)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	130	150
	-30	-15
	100	135

B(3) incluye el artículo 1

Índice	peso	Valor
1	30	60
2	50	60
3	40	40
Total	120	160
	-20	-20
	100	140

B(5) incluye (1,2)

Índice	peso	Valor
1	30	60
2	50	60
3	40	40
Total	120	160
	-20	-20
	100	140

B(7) incluye 1,2,3)

Índice	peso	Valor
1	30	60
2	50	60
3	40	40
Total	120	160

NO FACTIBLE

B(8) incluye (1,2,3*,4*)

Índice	peso	Valor
1	30	60
2	50	60
5	40	20
Total	120	140
	-20	-10
	100	130

B(11) incluye (1,2,3*,4,5)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	130	150

NO FACTIBLE

B(13) incluye (1,2,3*,4,5*,6)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	120	140

NO FACTIBLE

B(14) incluye (1,2,3*,4,5*,6*,7*)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
Total	90	130

B(10) incluye (1,2,3*,4,5*)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
6	30	10
Total	120	140
	-20	-6.7
	100	133.3

B(12) incluye (1,2,3*,4,5*,6*)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
7	10	3
Total	100	133

B(15) incluye (1,2,3*,4,5*,6*,7)

Índice	peso	Valor
1	30	60
2	50	60
4	10	10
7	10	3
Total	100	133

B(2) no incluye el artículo 1*

Indice	peso	Valor
2	50	60
3	40	40
4	10	10
total	100	110

B(3) incluye el artículo 1

Indice	peso	Valor
1	30	60
2	50	60
3	40	40
Total	120	160
	-20	-20
	100	140

la mejor cota es de 140 por lo que se procede a ramificar desde ahi:

B(4) incluye (1,2*)

Indice	peso	Valor
1	30	60
3	40	40
4	10	10
5	40	20
Total	120	130
	-20	-10
	100	120

B(5) incluye (1,2)

Indice	peso	Valor
1	30	60
2	50	60
3	40	40
Total	120	160
	-20	-20
	100	140

la mejor cota es 140 por lo que se procede a ramificar desde ahi:

B(6) incluye (1,2,3*)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	130	150
	-30	-15
	100	135

B(7) incluye 1,2,3)

Indice	peso	Valor
1	30	60
2	50	60
3	40	40
Total	120	160

NO FACTIBLE

B(9) incluye (1,2,3*,4)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	130	150
	-30	-15
	100	135

B(8) incluye (1,2,3*,4*)

Indice	peso	Valor
1	30	60
2	50	60
5	40	20
Total	120	140
	-20	-10
	100	130

B(11) incluye (1,2,3*,4,5)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	130	150

NO FACTIBLE

B(10) incluye (1,2,3*,4,5*)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
6	30	10
Total	120	140
	-20	-6.7
	100	133.3

B(13) incluye (1,2,3*,4,5*,6)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
5	40	20
Total	120	140

NO FACTIBLE

B(12) incluye (1,2,3*,4,5*,6*)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
7	10	3
Total	100	133

B(14) incluye (1,2,3*,4,5*,6*,7*)

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
Total	90	130

B(15) incluye (1,2,3*,4,5*,6*,7

Indice	peso	Valor
1	30	60
2	50	60
4	10	10
7	10	3
Total	100	133

De donde se observa que el óptimo se obtiene en el nodo (15).

El valor total es 133 llevando los artículos cuyos índices son 1, 2, 4 y 7, es decir los artículos 7, 2, 4, y 5 y el árbol se ilustra en la figura 5.12

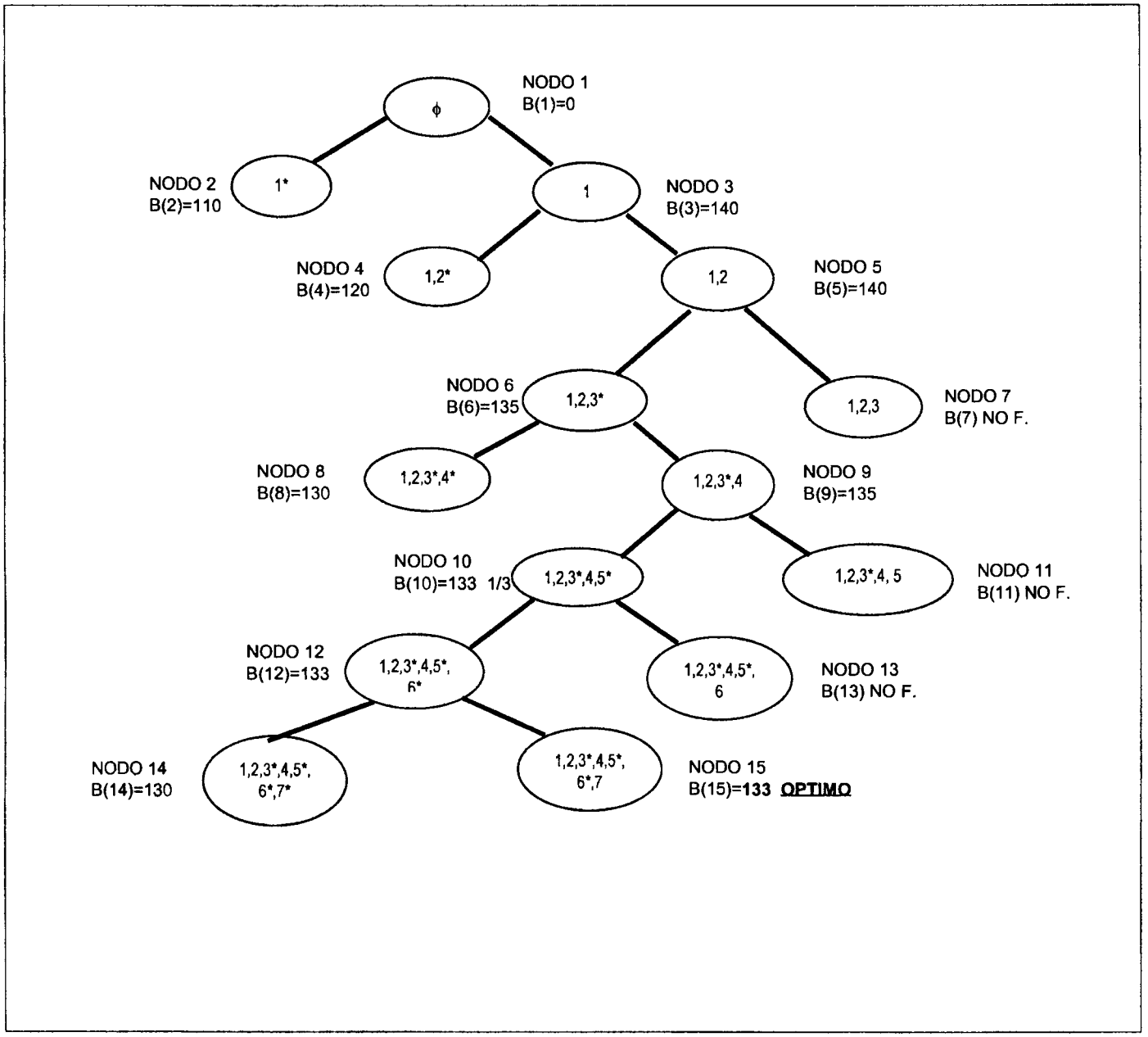


Figura 5.12

5.4. EL PROBLEMA DE LAS CUATRO REINAS.

-¿Sabes sumar?-le preguntó la reina blanca-
¿Cuánto es uno y uno y uno y uno y uno y uno y uno y uno?
-No sé -dijo Alicia- he perdido la cuenta...
-¡No tiene ni idea de matemáticas! –
sentenciaron enfáticamente ambas reinas a la vez.
Lewis Carroll. *Alicia a través del Espejo.*

Cuatro reinas se deben colocar en un tablero de cuatro por cuatro, de forma tal que ninguna reina pueda atacar a otra. En otras palabras, la colocación debe ser tal que en cada renglón, columna o diagonal del tablero, a lo mas contenga a una reina. El problema puede considerarse como una sucesión de problemas, en donde por ejemplo se coloca una reina en el primer cuadro del renglón de arriba, entonces se coloca otra reina en el segundo renglón de forma tal que no pueda ser atacada por la primera, y de igual forma se procede con la tercera y la cuarta reina.

Podemos asociar posiciones con nodos de un árbol, donde el nodo raíz T corresponde a la posición inicial sin reinas, los siguientes nodos corresponden a las posiciones posibles de una reina en el primer renglón (Es suficiente considerar como posiciones iniciales los primeros dos cuadros, ya que los otros dos dejan posiciones simétricas), conectados con el nodo T, por arcos de longitud cero, las siguientes ramas del árbol se construyen a partir de éstos nodos, considerando las posibles posiciones de una reina en el segundo renglón, sin que ésta ataque a la reina que se ha colocado con anterioridad, procediendo a unir éstos nodos con arcos de longitud cero, así se continúa hasta colocar a las cuatro reinas en sus posiciones respectivas.

Resolveremos primero éste problema usando la estrategia de búsqueda de primer profundidad. Numeramos los renglones y las columnas del tablero del 1 al 4, las reinas también pueden numerarse del 1 al 4. Ya que cada reina debe estar en diferente renglón, podemos suponer sin pérdida de generalidad que la reina i debe encontrarse en el renglón i. Todas las soluciones al problema deben presentarse entonces como cuartetas (x_1, x_2, x_3, x_4) donde x_i es la columna donde se pone la reina i. Las restricciones explícitas usando ésta formulación son $S_i = \{1,2,3,4\}$, $1 \leq i \leq n$. Entonces el espacio de soluciones consiste de 4^4 cuartetas. Las restricciones implícitas para éste problema es que no puede haber dos xi's que sean iguales(esto es, todas las reinas deben estar en columnas distintas) y dos reinas no pueden estar en la misma diagonal. La primera de éstas dos restricciones implica que todas las soluciones son permutaciones de las cuartetas (1,2,3,4). Esta realización reduce el tamaño del espacio de soluciones de 4^4 a 4!.

Mas adelante veremos como formular la segunda restricción en términos de las x_i. Expresada como una solución la cuarteta de la figura 5.13 se tiene (2,4,1,3).

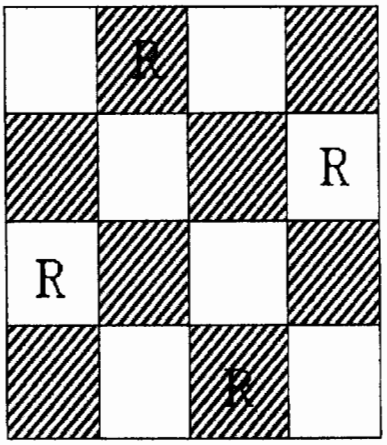


Figura 5.13

Como una función de acotamiento usamos el criterio obvio de que si $(x_1, x_2,...,x_i)$ es la trayectoria del T-nodo actual entonces todos los nodos hijos con etiquetas padre-hijo x_{i+1} tales que $(x_1,...,x_{i+1})$ representa una configuración en donde no se pueden atacar dos reinas.

Comenzamos con el nodo raíz como el único nodo vivo, éste es el T-nodo y la trayectoria es (0). Se genera un hijo. Suponemos que generamos hijos en orden ascendente, así el nodo numerado 2 de la figura 5.2 se genera y la trayectoria es ahora (1). Esto corresponde a poner la reina 1 en la columna 1. El nodo 2 se convierte en el T-nodo. El nodo 3 se genera e inmediatamente se muere. El siguiente nodo generado es el nodo 8 y la trayectoria se convierte en (1,3). El nodo 8 se convierte en el T-nodo, sin embargo se muere ya que sus hijos representan configuraciones que no producen una solución. Regresamos al nodo 2 y generamos otro hijo, el nodo 13. La trayectoria es ahora (1,4). La figura 2 muestra las configuraciones en el tablero como procedimientos recursivos, y muestra gráficamente los pasos que sigue el algoritmo al tratar de encontrar la solución, los puntos indican los lugares de una reina que se probaron y rechazaron porque otra reina estaba atacando.

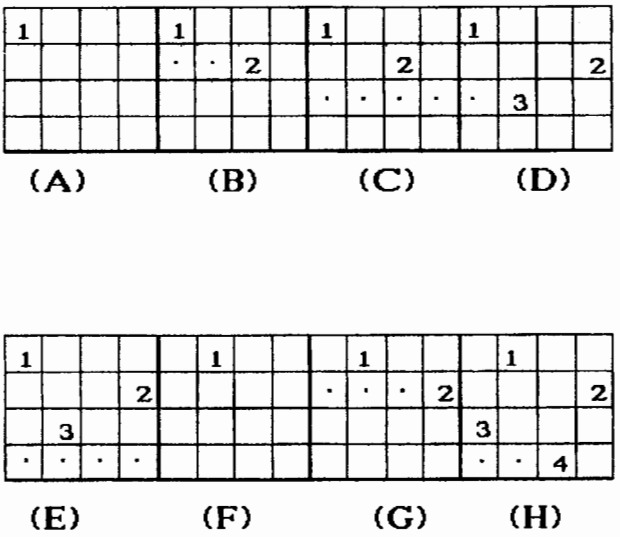


figura 5.14

En B) la segunda reina se pone en las columnas 1, 2 y finalmente se fija en la columna 3. En C) el algoritmo trata las cuatro columnas y no es posible colocar en ninguna a la próxima reina. Regresando se tiene en D) que la segunda reina se mueve a la próxima columna posible, la columna 4 y se pone la tercer reina en la columna 2. Estos tableros muestran los pasos que el algoritmo sigue hasta encontrar la solución.

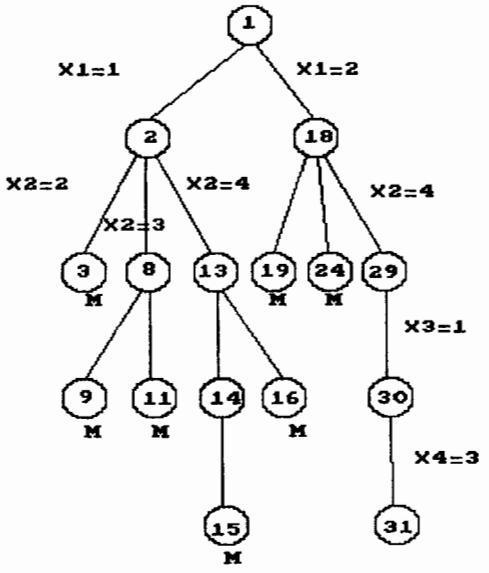


figura 5.15

La figura 5.15 muestra la parte del árbol que se ha generado usando la estrategia de primera profundidad. Los nodos que se han muerto como resultado de las funciones de acotamiento tienen una M debajo de ellos.

El árbol completo, usando búsqueda de primer profundidad y sin considerar las funciones de acotamiento se muestra en la figura 5.16

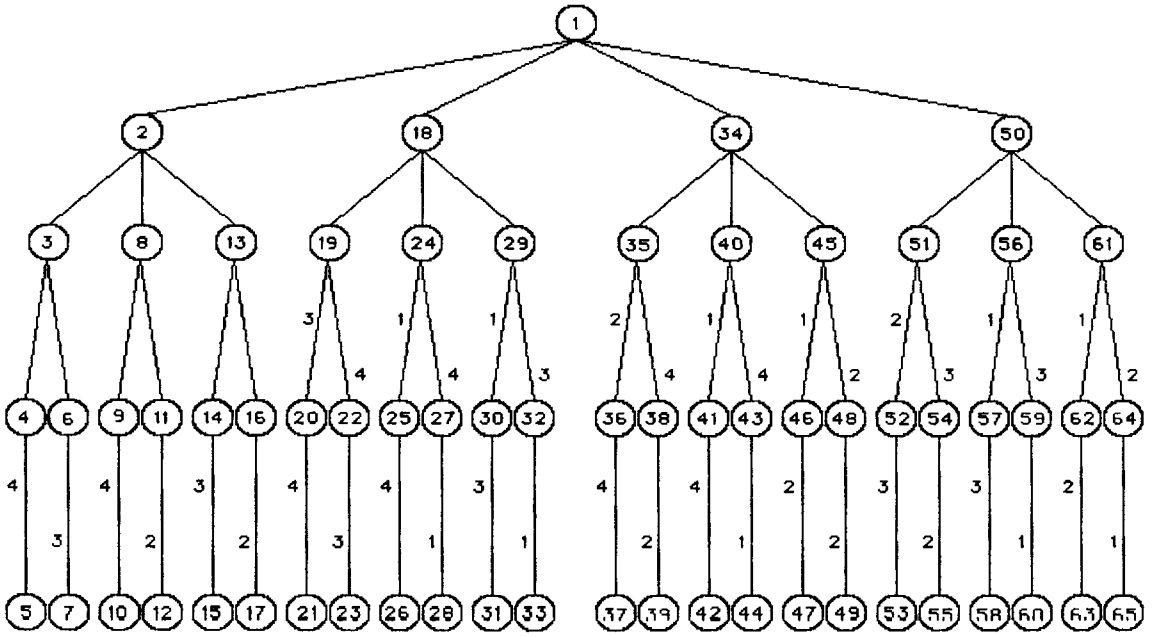


figura 5.16

Otra forma de resolverlo es usando la estrategia de primer amplitud o FIFO. Como ya se decía inicialmente hay un solo nodo vivo, el T-nodo y ahora le asignamos el número 1, (este es el caso en que no se tienen reinas en el tablero), Este nodo se particiona y sus hijos los nodos 2, 18, 34 y 50 se generan. estos nodos representan el tablero con una reina en el renglón 1 y columnas 1,2,3 y 4 respectivamente. Ahora los nodos vivos son los nodos 2,18,34 y 50. Si los nodos se generaron en éste orden el próximo T-nodo sería el nodo 2, éste se ramifica y los nodos 3, 8 y 13 se generan. El nodo 3 se mata inmediatamente usando la función de acotamiento. Los nodos 8 y 13 se aumentan a la cola de nodos vivos. El nodo 18 se convierte en el nuevo T-nodo. Los nodos 19, 24 y 29 se generan. Los nodos 19 y 24 se mueren como resultado de la función de acotamiento, que es la misma que se usó en la solución por primer profundidad. El nodo 29 se aumenta a la cola de nodos vivos. El próximo T-nodo es el nodo 34. La figura 5.17 muestra el árbol generado por la estrategia FIFO (primero en entrar primero en salir).

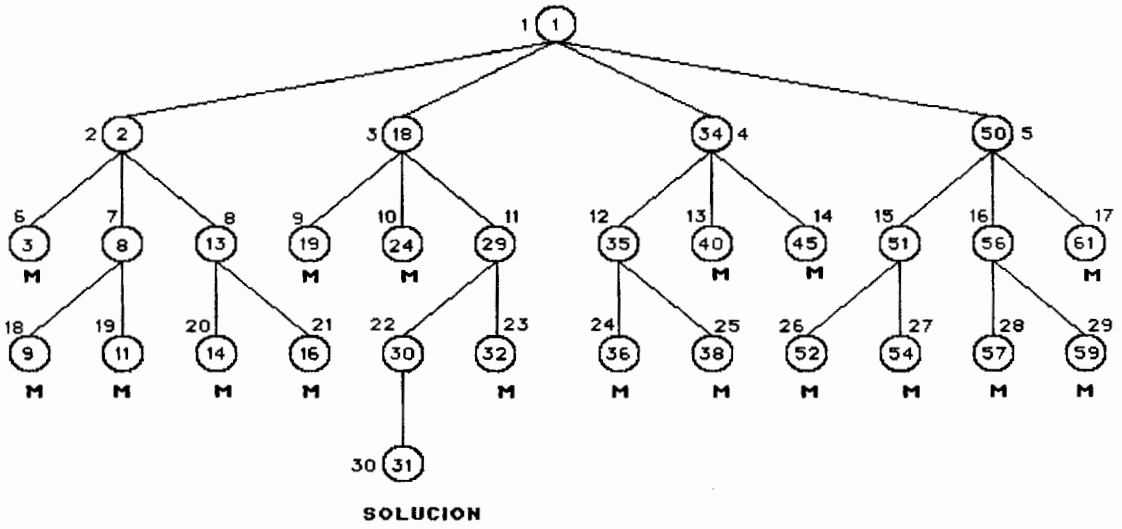


figura 5.17

Los números dentro del nodo son los mismos es la estrategia de primer profundidad y los números fuera del nodo corresponden al orden en que fueron generados por FIFO, en el momento en que se alcanza el nodo solución 31, los únicos nodos que permanecen vivos son los nodos 38 y 54. Una comparación de las figuras 5.16 y 5.17 indican que la estrategia de primer profundidad es un mejor método para éste problema.

EL JUEGO DEL CABALLO

Se tiene un tablero de nxn con n^2 campos. Un caballo (a quien se le permite moverse conforme a las reglas de ajedrez) se pone en el campo con las coordenadas iniciales x_0, y_0. El problema radica en encontrar una cobertura de todo el tablero (si es que existe), o sea calcular un circuito de $n^2 - 1$ movimientos (jugadas) tales que cada campo del tablero sea visitado exactamente una vez.

La manera obvia de reducir el problema es abarcar n^2 campos consiste en considerar el problema de realizar una jugada siguiente o descubrir que ninguna es posible.

Se tiene noticia de que Euler (1759) y Vandermonde (1771) discutieron el problema del circuito de un caballo. Este problema es un circuito hamiltoniano en una gráfica cuyos vértices son los 64 cuadros de un ajedrez, con dos vértices adyacentes si y sólo si el caballo se puede mover en un paso de un cuadrado al otro.

La solución a dicho problema dada por Leonhard Euler consiste en construir un cuadrado en el que la suma de cada hilera vertical u horizontal da 260; y a media hilera la suma da 130. Lo mas sorprendente es que además nos da como resultado adicional que si el caballo comienza moviéndose a partir del cuadro 1 en su movimiento de L puede recorrer los 64 cuadros en orden numérico, como se muestra en la figura 5.18

1	48	31	50	33	16	63	18
30	51	46	3	62	19	14	35
47	2	49	32	15	34	17	64
52	29	4	45	20	61	36	13
5	44	25	56	9	40	21	60
28	53	8	41	24	57	12	37
43	6	55	26	39	10	59	22
54	27	42	7	58	23	38	11

figura 5.18

Esto lo podemos ver también a través de la construcción de un árbol como en el problema de las reinas donde se ven las posibles jugadas iniciando desde un cuadro cualquiera

LA TORRE DE HANOI

La conocida Torre de Hanoi fue inventada por el matemático francés Edouard Lucas, y se vendió como juguete en 1883. Originalmente llevaba el nombre de "Profesor Claus" del Colegio de "Li- Sou - Stian" pero pronto se descubrió que eran anagramas de "Profesor Lucas" del Colegio de "Saint Louis". El problema consiste en transferir la torre de ocho discos a cualquiera de dos clavijas vacías en el menor número de movimientos, moviendo un disco cada vez y sin colocar un disco encima de otro más pequeño.

Considere los tres postes A, B y C (como se muestran en la figura 5.19)

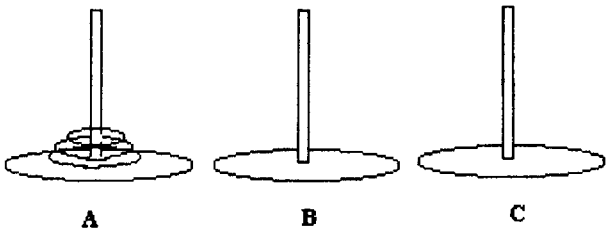


Figura 5.19

Inicialmente el poste A tiene cierta cantidad de discos de distintos tamaños, con el más grande en la parte inferior y otros sucesivamente más pequeños encima. El problema se puede resolver sencillamente con el siguiente algoritmo: se imaginan los postes dispuestos en un triángulo. Con un número de movimientos impar, se mueve el disco más pequeño hacia un poste en el sentido de las manecillas del reloj. Con un número de movimientos pares, se hace un único movimiento válido que no implique al disco más pequeño.

El algoritmo anterior es conciso y correcto, pero es difícil entender por qué funciona y de descubrir intuitivamente. En cambio considere el siguiente enfoque de ramificación y acotamiento: el problema de pasar los n discos más pequeños de A a B puede considerarse compuesto de dos problemas uno de ellos de tamaño n-1. Primero se mueven los n-1 discos más pequeños del poste A al C, dejando el n-ésimo disco mas grande en el poste A. Se mueve ese disco de A a B. Después se pasan los discos más pequeños de C a B. El movimiento de los n-1 discos más pequeños se efectuará por medio de la aplicación recursiva del método. Como los n discos implicados en los movimientos son más pequeños que los demás discos, no es necesario preocuparse de los que se encuentran debajo de ellos en los postes A, B o C. Aunque el movimiento real de los discos individuales no es obvio y la simulación a mano es difícil debido al apilamiento de llamadas recursivas, el algoritmo es conceptualmente fácil de entender, de probar que es correcto y de considerarlo como una buena opción.

No es difícil demostrar que hay una solución, sin importar cuántos discos hay en la torre, y que el número mínimo de movimientos necesarios está expresado por la fórmula $2^n - 1$ donde n es el número de discos. De éste modo, se pueden transferir tres discos en siete movimientos, cuatro en 15, cinco en 31 y así sucesivamente. Para los ocho discos se requieren 255 movimientos.

La descripción original del juguete decía que éste era una versión simplificada de una mítica "Torre de Brahma" de un templo de la ciudad india de Benarés.

La descripción dice que ésta torre está formada por 64 discos de oro, que están en el proceso de ser transferidos por los sacerdotes del templo. Antes de que completen su tarea, se decía, el templo se desmoronará hecho polvo y el mundo se desvanecerá el estampido de un trueno. Puede ponerse en duda la desaparición del mundo, pero no así el desmoronamiento del templo. La fórmula $2^{64} - 1$ da por resultado un número de 20 dígitos.

18, 446, 744, 073, 709, 551, 615

Suponiendo que los sacerdotes trabajaran día y noche, moviendo un disco por segundo, terminar el trabajo les llevaría varios miles de millones de años.

A propósito de éste número, es que no es primo, pero si aumentamos el número de discos a 89, 107 o 127, el número de movimientos necesario para transferirlos es primo, en cada caso. Son ejemplos de los llamados números Mersenne: primos que tienen la forma de $2^n - 1$. El propio Lucas fue el primer hombre que verificó que $2^{127} - 1$ era primo. Este monstruoso número de 39 dígitos era el primo mas grande que se conocía hasta 1952, cuando se utilizó una gran computadora electrónica para encontrar cinco números primos Mersenne mayores que ése. Se conocen treinta números primos Mersenne. El trigésimo y mayor $2^{216091} - 1$ fue descubierto en 1985 y tiene 65,050 dígitos.

EL AGENTE VIAJERO Y LA TORRE DE HANOI

¿ Cómo se relaciona este rompecabezas con el juego de Hamilton?

Para explicar la conexión debemos considerar primero una torre de tres discos solamente, etiquetando los discos, desde arriba hacia abajo A, B y C. Si seguimos el procedimiento mencionado para resolver el problema de la torre de Hanoi tendremos la solución moviendo los discos en el siguiente orden: ABACABA.

Llamemos ahora A, B y C a las tres coordenadas de un hexaedro regular, llamado comúnmente cubo, como se muestra en la figura 5.20

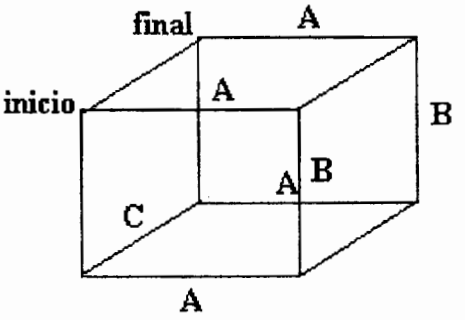


figura 5.20

Si trazamos una ruta a lo largo de los bordes del cubo eligiendo las coordenadas en el orden ABACABA la ruta formará un circuito hamiltoniano.

La generalización de esto la vio D. W. Crowe de la Universidad de Columbia Británica como sigue: el orden de la transferencia de n discos en la Torre de Hanoi se corresponde exactamente con el orden de las coordenadas al trazar un camino Hamiltoniano en un cubo de n dimensiones. Para el caso de n = 4 se proyecta la red de sus bordes en un modelo tridimensional como se muestra en la figura 5.21.

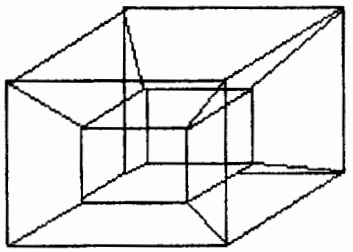


figura 5.21

Este hipercubo tiene las coordenadas A, B, C y D, el camino que se sigue es ABACABADABACABA. Lo mismo se hace para un hipercubo de 5 dimensiones, 6 hasta n.

5.5 NOTAS HISTÓRICAS

La primera vez que se usó el término "El Problema del Agente Viajero" en los círculos matemáticos pudo ser en los años 1931-32 por Karl Menger. Pero en 1832, un libro editado en Alemania titulado " El Agente Viajero, como debe ser y que debe hacer para cumplir sus comisiones y tener éxito en los negocios. Por un Agente Viajero Veterano". Aunque el libro trata de muchos otros temas en su último capítulo alcanza lo que es la esencia del problema: " Para la selección apropiada de un buen recorrido, se puede ganar mucho tiempo para lo cual hacemos algunas sugerencias...El aspecto más importante es cubrir la mayor cantidad de lugares sin visitar alguno dos veces..."

No sabemos con exactitud quién trajo el nombre del problema en los círculos matemáticos pero no hay duda de que Merrill Flood es responsable de su publicación en la comunidad de Investigación de Operaciones. Flood menciona que fue A.W. Tucker quien en 1937 se lo mencionó al hablar sobre el estudio de ruta de un autobús escolar. Pero Tucker dice a su vez que la historia la escuchó de Hassler Whitney, lo que ocurrió entre 1931 y 1932.

CAPÍTULO 6
MÉTODOS HEURÍSTICOS

La imaginación es más importante que el conocimiento
Albert Einstein

6.1 INTRODUCCIÓN

En Investigación de Operaciones el término "heurístico" se aplica a menudo a métodos (que pueden o no involucrar búsquedas) que están basados en argumentos intuitivos o plausibles que conducen a soluciones razonables pero que no garantizan el óptimo. Son métodos para el problema bajo estudio, basados en reglas de sentido común, o adaptaciones de métodos exactos para modelos simples.

Un problema con el que comenzamos este capítulo es el de las cuatro reinas mismo que ya habíamos considerado en el caso de ramificación y acotamiento, además abordaremos otros casos y algunos métodos heurísticos que se han convertido en clásicos dentro del área de optimización combinatoria. Tales métodos son:

- Recocido Simulado
- Algoritmos Genéticos
- Búsqueda Tabú
- Redes Neuronales
- GRASP (Greedy, Randomized, Adaptive, Search, Procedures, o procedimientos de búsqueda glotones, aleatorizados y adaptativos).

También ya se mencionaba en el primer capítulo de estos apuntes, la naturaleza de los problemas que se busca resolver en esta área lo que los sitúa en términos de complejidad computacional como NP-duros. Es por esto entre otros factores que se comenzó a buscar una forma diferente de resolverlos con algoritmos que no fueran exactos. A partir de la década de los setentas se comenzaron a conocer más sus resultados y se dieron a conocer ampliamente. Otros factores que contribuyeron a su desarrollo fueron:

1. Cuando no existe un método exacto de solución o éste requiere mucho tiempo de cálculo o memoria.
2. Cuando no se necesita la solución óptima.
3. Cuando los datos son poco fiables.
4. Cuando se requieran respuestas rápidas
5. Como auxiliar o paso intermedio en otro algoritmo.

Una ventaja importante al usar heurísticos es que permiten una mayor flexibilidad para el manejo de las características del problema. No suele resultar complejo diseñar algoritmos heurísticos que en lugar de considerar funciones lineales usen no lineales. Además, generalmente ofrecen más de una solución, lo que permite

ampliar las posibilidades de elección del que decide. Por otra parte, suele ser más fácil de entender por parte de personas que no son expertas en la formulación.

Por el contrario, el inconveniente del uso de métodos heurísticos, es que por lo general no es posible conocer la calidad de la solución, es decir, cuán cerca está del óptimo, la solución que nos ofrecen. Afortunadamente, existen métodos para realizar acotaciones que permiten tener una orientación de la calidad de la solución obtenida.

Este capítulo se desarrolla como sigue: En la segunda sección se presentan algunos ejemplos de usos de métodos heurísticos, como en el problema de las ocho reinas visto en el capítulo anterior y el rompecabezas del ocho. En la tercera sección se presentan algunos métodos heurísticos para resolver el problema del agente viajero. En la cuarta los diferentes tipos de heurísticos para en la quinta desarrollar el método de Recocido Simulado. En la sexta sección se desarrollan Algoritmos Genéticos y finalmente en la séptima las Notas Históricas.

6.2 USOS TÍPICOS DE HEURÍSITICOS PARA LA SOLUCIÓN DE PROBLEMAS.

Los heurísticos son criterios, métodos o principios para decidir cual de entre varios cursos de acción promete ser el más efectivo para alcanzar un objetivo. Representan compromisos entre dos requerimientos: la necesidad de hacer tal criterio simple y, al mismo tiempo, el deseo de poder discriminar correctamente entre las buenas y las malas elecciones.

Un heurístico puede ser una "regla de dedo" que se usa para guiar nuestra acción. Por ejemplo, un método popular para escoger un melón consiste en presionar el punto del melón en cuestión, que estaba unido a la planta y oler este punto. Si huele como si fuera el interior del melón, lo más probable es que este maduro. Esta regla de dedo no garantiza buenos resultados, pero es efectiva la mayoría de las veces.

Otro ejemplo del uso de heurísticos es el siguiente: considere a un gran maestro del ajedrez que está enfrentándose a varias jugadas posibles. Este gran maestro puede decidir que un movimiento particular es más efectivo porque el resultado en la posición del tablero "aparece" más fuerte que las posiciones resultantes de otros movimientos.

El criterio de "parece más fuerte" es más simple de aplicar para este gran maestro que el determinar rigurosamente que movimiento o movimientos fuerzan a un jaque mate. El hecho de que los grandes maestros no siempre ganen indican que sus heurísticos no garantizan la selección del movimiento más efectivo. Finalmente cuando se les pide describir sus heurísticos, los grandes maestros solo pueden dar descripciones parciales y rudimentarias de lo mismo que ellos aplican sin mayor esfuerzo.

Los problemas más complejos requieren de la evaluación de un gran número de posibilidades para determinar una solución exacta. El tiempo requerido para encontrar una solución exacta a menudo es mayor que toda una vida. Los heurísticos juegan un papel efectivo en tales problemas indicando un camino para reducir el número de evaluaciones y para obtener soluciones dentro de restricciones de tiempo razonables. Para ilustrar el papel de los heurísticos en la solución de problemas haremos uso de algunos juegos como el de las 8 reinas.

EL PROBLEMA DE LAS OCHO REINAS

Este problema es solo una ampliación del problema de las cuatro reinas así que las soluciones deben cumplir con las mismas restricciones que el anterior. Un método podría consistir en ir resolviendo el problema en forma **incremental** paso a paso, siguiendo una serie de decisiones locales cada una de ellas basada en nueva información reunida a lo largo del camino. Podemos empezar, por ejemplo, en un arreglo arbitrario de las ocho reinas en el tablero y transformar el arreglo inicial iterativamente, yendo de una configuración inicial del tablero a otra hasta que las ocho reinas se han dispuesto en forma adecuada. Debemos estar seguros que la sucesión de transformaciones no es aleatoria pero si es **sistemática** de tal forma que no se genere la misma configuración una y otra vez y tal que no perdamos la oportunidad de generar la configuración deseada. Una manera de sistematizar la búsqueda consiste en **construir** (más que transformar) configuraciones en el tablero paso a paso.

Comenzando con el tablero vacío podemos considerar el poner todas las reinas en el tablero al mismo tiempo y considerar la violación de las reglas hasta obtener una configuración satisfactoria con las ocho reinas arregladas.

Suponga que estamos en alguna etapa de la búsqueda, las primeras tres reinas se han posicionado en sus lugares marcadas con R`s como se muestra en la figura 6.1, y tenemos que decidir donde posicionar la cuarta reina en la celda A, B o C. El papel del heurístico en este contexto sería el de proporcionar un criterio de regla de dedo para decidir donde posicionar la cuarta reina en la celda A, B o C. El papel del heurístico en este contexto sería el de proporcionar un criterio de regla de dedo para decidir, al menos tentativamente cual de las tres posiciones parece tener mejor oportunidad de arrojar una solución satisfactoria.

Considerando tales heurísticos podemos razonar que para estar en condiciones de poner las ocho reinas, necesitamos dejar tantas opciones como sean posibles para futuras adiciones. Esto significa que necesitamos poner atención al número de celdas en los renglones no usados que permanecen no atacadas por las reinas que se colocaron previamente. Una celda candidata es preferida si deja el mayor número de **celdas no atacadas** en las posiciones restantes del tablero. Consecuentemente el número de celdas no atacadas dejadas por alguna alternativa constituye una medida f(.) de su mérito.

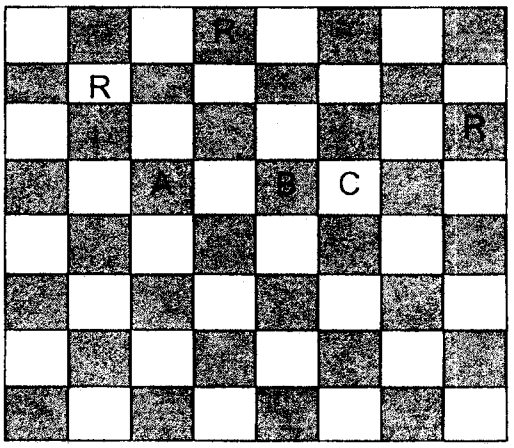


Figura 6.1

Si calculamos la función f(.) para las alternativas A, B y C en el ejemplo anterior tenemos

$$f(A) = 2 + 1 + 2 + 3 = 8$$
$$f(B) = 3 + 1 + 3 + 2 = 9$$
$$f(C) = 2 + 3 + 2 + 3 = 10$$

Así la alternativa C será la preferida. Podemos darle más valor a nuestra heurística mostrando la solución dada por las alternativas B y C (ver figura 6.2), mientras que la alternativa A no da ninguna solución al problema.

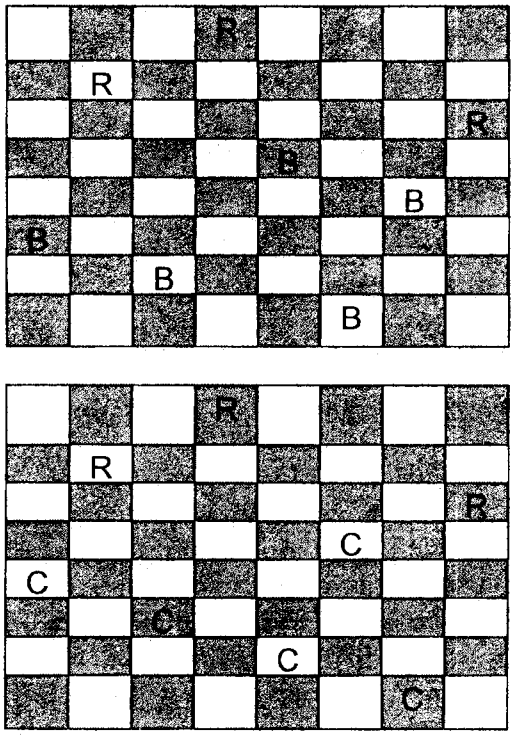


Figura 6.2

EL PROBLEMA DEL ROMPECABEZAS DEL OCHO

El objetivo de este rompecabezas es rearreglar una configuración inicial de ocho fichas numeradas y arregladas en un tablero de 3x3, de forma tal que queden ordenadas en una configuración llamada el estado meta. El rearreglo solo es posible si se puede deslizar una de las fichas en un cuadro vacío que esta colocado en una posición adyacente, como se muestra en la figura 6.3.

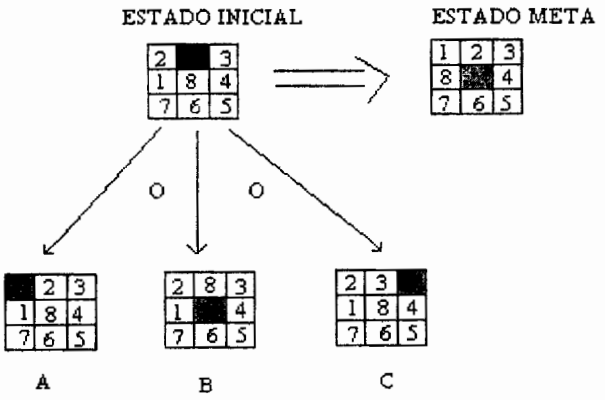


Figura 6.3

¿Cuál de las tres alternativas A, B o C parece más prometedora? La respuesta se puede obtener por supuesto buscando exhaustivamente movimientos subsecuentes en el rompecabezas para ver cual de los tres estados nos da la ruta más corta hacia el estado meta. La búsqueda exhaustiva, es sin embargo impráctica cuando el número de estados que se debe examinar es inmenso. Esta "explosión combinatoria" ocurre en tales rompecabezas cuando la longitud de la ruta del estado inicial al estado meta es muy larga, o cuando se involucran tableros de mayor tamaño (por ejemplo de 4x4).

Una manera de encontrar la trayectoria consiste en estimar que tan cerca esta un estado o configuración del estado meta. Para este problema hay dos heurísticos muy comunes que se usan para estimar la cercanía de un estado con otro. La primera es el número de fichas que no están bien ubicadas, aquellas para las cuales los dos estados difieren, y la llamaremos heurístico h_1. El segundo se refiere a la suma vertical y horizontal de las distancias de las fichas que no concuerdan entre los dos estados, y la llamaremos h_2. Este segundo heurístico recibe también el nombre de Manhattan o distancia de las calles de una ciudad. Para ver como funcionan estos heurísticos calculamos las estimaciones para los estados A, B y C de la figura 6.3.

$h_1(A) = 2$	$h_1(B) = 3$	$h_1(C) = 4$
$h_2(A) = 2$	$h_2(B) = 4$	$h_2(C) = 4$

Evidentemente ambos heurísticos proclaman que el estado A esta más cerca al estado meta y por lo tanto debe explorarse antes que a B o C.

6.3 ALGUNOS MÉTODOS HEURÍSTICOS PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO EUCLIDEANO.

Considere nuevamente el problema del agente viajero, dado en la figura 6.4

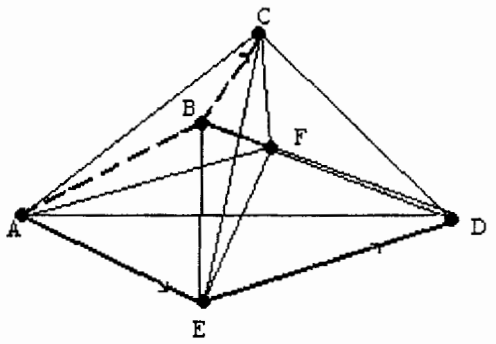


Figura 6.4

Las dos trayectorias marcadas ABC y AED representan dos recorridos parciales. La pregunta aquí es cuál de los dos al completarse forma un recorrido completo que sea óptimo. Los costos del recorrido se suman y se trata de encontrar el que sea más económico. Sin embargo el problema de encontrar el costo óptimo completo es más difícil que encontrar el costo de los recorridos parciales, por lo que se puede usar una heurística que estime este costo. Así lo que procede es considerar estos costos y aumentarle el costo estimado de tal forma que se tenga un costo total estimado.

Se tiene entonces que inventar una función que estime el costo total, se puede dar algunas que son simples y fáciles de calcular. Por ejemplo, se puede tomar el costo del nodo final del recorrido parcial hacia el nodo final, otro ejemplo sería tomar el recorrido más barato y tomamos los nodos inicial y final de ese recorrido para completar el recorrido y comparamos los costos.

Se han propuesto varias funciones en la literatura y las más usadas son:

1. La gráfica de segundo grado que va hacia los nodos remanentes
2. El árbol de expansión mínima hacia los nodos remanentes.

La primera se obtiene usando el problema de asignación óptimo que consta de N^3 pasos o una complejidad computacional de N^3. La segunda requiere N^2 pasos.

Que estas dos funciones proporcionen estimaciones óptimas para completar los costos es aparente ya que si consideramos que completar el recorrido requiere la selección de una trayectoria que vaya a través de todas las ciudades no visitadas. Una trayectoria es un caso especial de una gráfica de grado 2 y también un caso especial de un árbol de expansión. Entonces el conjunto de todas las trayectorias de complexión pertenece al conjunto de todos los objetos sobre los cuales la optimización tiene lugar, y así la solución encontrada debe tener un costo inferior que el obtenido al optimizar sobre el conjunto de trayectorias. La figura 6.5 a) muestra la solución encontrada al resolver un problema de asignación en lugar de completar el sub-recorrido AED. La figura 6.5 b) muestra una complexión de un árbol de expansión mínima del sub-recorrido AED. En éste caso lo que encontramos es un árbol que contiene un nodo de grado mayor que 2.

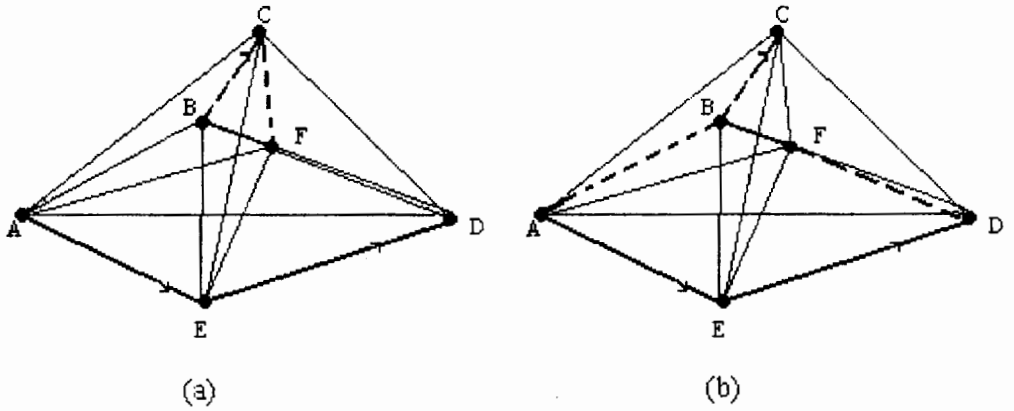


Figura 6.5

Estas soluciones pueden usarse como una buena estimación del recorrido óptimo, y como heurísticas para encontrar los candidatos que mejoren la eficiencia de la búsqueda. Otras heurísticas para este problema aunque se considera el caso euclideano son las siguientes.

Heurística de la vecindad más cercana

Esta heurística usa un procedimiento "glotón" para construir una ruta. Se comienza en una ruta parcial que consta de una ciudad elegida arbitrariamente. En cada etapa la heurística sólo se mueve un paso hacia adelante y decide ir desde la última ciudad en la ruta a la ciudad que este más cercana que no este en la ruta. Esta heurística tan simple puede dar soluciones poco satisfactorias. El último enlace del recorrido puede ser muy grande. Note que el recorrido construido por esta heurística depende de la ciudad que se seleccione para comenzar. Si se empieza en diferentes ciudades se tienen distintos recorridos.

Heurística de Inserción

Esta heurística construye una sucesión de sub-circuitos de tamaño creciente. Para iniciar el algoritmo, seleccione arbitrariamente una ciudad para iniciar el recorrido. En cada etapa se selecciona una ciudad no visitada de acuerdo a algún criterio y se agrega al actual sub-recorrido. Las heurísticas de inserción difieren en el paso de selección. La heurística más cercana (más lejana) selecciona el nodo no visitado que es más cercano (o más alejado de) cualquier ciudad del sub-recorrido actual. El paso de inserción es el mismo en ambas heurísticas. Suponga que se selecciona la ciudad K para aumentarla al actual sub-recorrido primero determine el arco (v, w) en el sub-recorrido actual con un costo de inserción C_{vk} + C_{kw} - C_{vw}. Que es un mínimo donde C_{ij} denota la distancia de la ciudad i a la ciudad j. Entonces se inserta la ciudad k entre las ciudades v y w. Experimentos han demostrado que la inserción más lejana dan mejores resultados que otras estrategias de inserción.

Heurística del mayor ángulo

A diferencia de los algoritmos anteriores, el algoritmo del mayor ángulo explota la geometría inherente a los problemas del agente viajero euclideano. La heurística del mayor ángulo toma la cubierta convexa de las ciudades como un sub-recorrido inicial. La cubierta convexa consiste del menor número de ciudades en orden en que la región abarcada por esas ciudades contiene a las demás ciudades como puntos interiores. Después de la construcción de este sub-recorrido inicial las ciudades remanentes se insertarán entre las ciudades consecutivas del sub-recorrido. El algoritmo consiste en nuevamente insertar ciudades una a la vez para formar un nuevo sub-recorrido hasta que todas las ciudades se han incluido.

El paso de inserción se basa en un argumento geométrico. Se miden todos los ángulos cuyos vértices están en puntos interiores y cuyos lados son rectos a través de ciudades consecutivas en el sub-recorrido. La ciudad con el mayor ángulo se escoge para insertarse entre dos ciudades consecutivas en el sub-recorrido el proceso de medir ángulos y seleccionar el mayor se repite en el nuevo sub-recorrido y las ciudades interiores restantes. Esta heurística usualmente da soluciones sorprendentemente buenas.

6.4 TIPOS DE HEURÍSTICOS

1. **MÉTODOS CONSTRUCTIVOS**. Consisten en ir paulatinamente añadiendo componentes individuales a la solución hasta que se obtiene una solución factible. El más popular de estos métodos lo constituyen los algoritmos glotones o devoradores (greedy), los cuales construyen paso a paso la solución buscando el máximo beneficio en cada paso. Un ejemplo de ellos es GRASP. Por ejemplo en el agente viajero se puede tener un algoritmo glotón si se escoge una ciudad al azar, y luego desplazarse sucesivamente a la ciudad más cercana hasta cerrar el circuito.

2. **MÉTODOS DE DESCOMPOSICIÓN**. Se trata de dividir el problema en subproblemas más pequeños, divide y vencerás. Un ejemplo de aplicación de este método a problemas de programación entera mixta, consiste en decidir de alguna forma (quizá mediante alguna otra heurística) una solución para las variables enteras, y luego resolver el problema LP obtenido al sustituir esas variables por su valor. En el caso del agente viajero consiste en dividir el plano en varias regiones pequeñas y resolver el TSP para las ciudades que haya en cada una de ellas, uniendo finalmente todas las soluciones para obtener la solución del problema global.

3. **MÉTODOS DE REDUCCIÓN**. Tratan de identificar alguna característica que presumiblemente deba poseer la solución óptima y de ese modo simplificar el problema. Así, puede identificarse que alguna variable siempre deba tomar el valor cero, que otras están correlacionadas, etc.

4. **MANIPULACIÓN DEL MODELO**. Estas heurísticas modifican la estructura del modelo al hacerlo más sencillo de resolver, deduciendo de su solución la solución del modelo original. Por ejemplo se pueden linearizar las funciones que no son lineales, agrupar variables para reducir su número, imponer nuevas restricciones basándose en el comportamiento esperado que debería tener la solución óptima, o incluso aumentarlo al eliminar restricciones del problema.

5. **MÉTODOS DE BÚSQUEDA POR ENTORNOS**. Dentro de esta última categoría es donde se encuadran la mayoría de las meta-heurísticas que analizaremos a continuación. Estos métodos parten de una solución factible inicial (obtenida quizá mediante otra heurística) y, mediante alteraciones de esa solución, van pasando de forma iterativa, y mientras no se cumpla un determinado criterio de parada, a otras factibles de su "entorno" almacenando como óptima la mejor de las soluciones visitadas. Este entorno se define como vecindad. Por ejemplo si consideramos el problema de la mochila con 5 artículos y tenemos una solución s=(1,0,0,1,1) donde xi=1 indica el artículo i seleccionado. Podríamos considerar en el entorno de s todas aquéllas soluciones consistentes en sacar uno de los artículos seleccionados y cambiarlo por otro. De este modo en el entorno de s podríamos considerar la solución s'=(0,0,1,1,1). Se sacó el artículo 1 y se metió el 3.

6.5 METAHEURÍSTICAS: RECOCIDO SIMULADO

El concepto de recocido (annealing) en optimización combinatoria fue introducido a inicios de la década de los 80 por Kirkpatrick, Gellat y Vecchi (1983) e independientemente por Cerny (1985). Este concepto está basado en una fuerte analogía entre el proceso físico de recocidos de sólidos y el problema de resolver problemas de optimización combinatoria grandes.

EL PROCESO DE RECOCIDO DE UN SÓLIDO

Recocido denota un proceso de calentamiento de un sólido a una temperatura en la que sus granos deformados recristalizan para producir nuevos granos. La temperatura de recocido o de recristalización, depende del tipo de material, del grado de deformación del mismo, además de su uso futuro. Seguida a la fase de calentamiento, viene un proceso de enfriamiento en donde la temperatura se baja poco a poco. De esta manera, cada vez que se baja la temperatura, las partículas se reacomodan en estados de más baja energía hasta que se obtiene un sólido con sus partículas acomodadas conforme a una estructura de cristal. Si se comienza con un valor máximo de temperatura, en la fase de enfriamiento del proceso de recocido, para cada valor de la temperatura T debe permitirse que se alcance su equilibrio térmico. Sin embargo, si el proceso de enfriamiento es demasiado rápido y no se alcance en cada etapa el equilibrio térmico, el sólido congelará en un estado cuya estructura será amorfa en lugar de la estructura cristalina de más baja energía. La estructura amorfa está caracterizada por una imperfecta cristalización del sólido.

El equilibrio térmico está caracterizado por la distribución de Boltzmann. De acuerdo a esta distribución, la probabilidad de que el sólido esté en un estado i con energía E_i a la temperatura T, viene dado por

$$P_T\{X = i\} = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{k_B T}\right),$$

Donde X es una variable aleatoria que denota el estado actual del sólido. Z(T) es una constante de normalización llamada la función partición, que está definida como

$$Z(T) = \sum_j \exp\left(\frac{-E_j}{k_B T}\right)$$

Donde la sumatoria se extiende sobre todos los estados posibles y k_B es una constante física conocida como la constante de Boltzmann. El factor exp $(-E_i/k_B T)$ se conoce como el factor de Boltzmann. Obviamente la función P_T es una función de densidad de probabilidad ya que siempre es mayor o igual a cero y la suma sobre todos los valores es igual a la unidad. Se puede observar en esta función que cuando el valor de T disminuye, la distribución de Boltzmann se concentra en los estados de menor energía mientras que si la temperatura se aproxima a cero, únicamente los estados con mínima energía tienen una probabilidad de ocurrencia diferente de cero.

Por lo dicho anteriormente, el proceso de recocido consta de dos pasos fundamentales que son:

- Incrementar la temperatura del baño térmico a un valor máximo.
- Decrementar cuidadosamente la temperatura del baño térmico hasta que las partículas se reacomoden por sí mismas en un estado de mínima energía, denominado el estado fundamental del sólido.

En el proceso de elevar la temperatura del sólido, todas las partículas se reacomodan aleatoriamente. En el estado fundamental, las partículas se acomodan en una retícula altamente estructurada y la energía del sistema es mínima. El estado fundamental del sólido se obtiene solamente si la temperatura máxima es suficientemente elevada y el proceso de enfriamiento es suficientemente bajo. De otra manera se obtendrá un estado amorfo denominado meta-estado.

El proceso físico de recocido puede modelarse exitosamente usando métodos de simulación, el algoritmo que se presenta para tal propósito se basa en técnicas Monte Carlo y genera una sucesión de estados del sólido de la siguiente manera. Dado un estado i del sólido con energía E_i, se genera un estado subsecuente j aplicando un mecanismo de perturbación que transforma el estado actual en el siguiente estado por medio de una pequeña distorsión, por ejemplo, por el desplazamiento de una partícula. La energía del siguiente estado es Ej. Si la diferencia de energía, $E_j - E_i$, es menor o igual a cero, el estado j se acepta como el estado actual. Si la diferencia de energía es mayor que cero, el estado j se acepta con una probabilidad que esta dada por:

$$\exp\left(\frac{E_i - E_j}{k_B T}\right),$$

Donde T denota la temperatura del baño térmico y k_B es la constante de Boltzmann. La regla de decisión descrita arriba se conoce como el criterio de Metrópolis y al algoritmo se le conoce como algoritmo de Metrópolis.

Si la temperatura se baja poco a poco, el sólido puede alcanzar su equilibrio térmico en cada temperatura. En el algoritmo de Metrópolis esto se lleva a cabo generando un número grande de transiciones para un valor dado de la temperatura.

ALGORITMO DE METRÓPOLIS

Este algoritmo es muy conocido en el mundo de la Química-Física. De los dos métodos de solución numérica básicamente utilizados en Mecánica Estadística para estudiar el comportamiento microscópico de los cuerpos (el método de Dinámica Molecular y el método Montecarlo), este último es con el que se estudian usualmente las propiedades de equilibrio, y éste fue, precisamente el utilizado por Metrópolis en su algoritmo.

El origen químico-físico de este procedimiento es el siguiente. Dada una sustancia cualquiera, no todas sus moléculas tienen la misma energía, sino que estas se encuentran distribuidas en distintos niveles, el menor de los cuales se denomina ε_0. Si la sustancia está a 0°K todas las moléculas están en su estado fundamental, y conforme es mayor la temperatura, las moléculas ocupan niveles superiores. Cada una de las maneras en que las moléculas pueden estar distribuidas entre los distintos niveles recibe el nombre de "microestado". Se denomina Ω al conjunto de todos los posibles microestados y "número de ocupación" n_i al número de partículas en el nivel i. Ver la figura 6.6

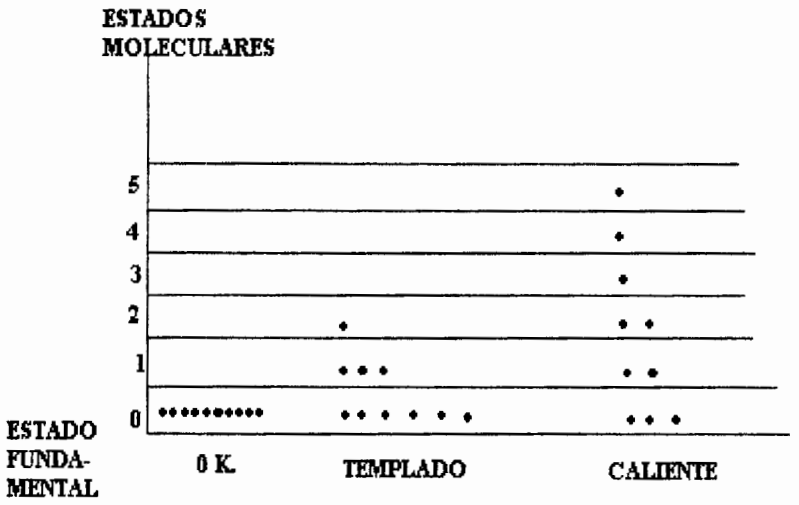


Figura 6.6

Metrópolis quería definir un algoritmo que permitiera calcular valores medios de la energía interna y de la presión dentro del formalismo de los colectivos de Gibbs, sin necesidad de recurrir a la solución analítica de las correspondientes integrales. Utilizó para ello el método de Montecarlo (que ha demostrado en múltiples ocasiones su utilidad en la resolución de integrales definidas) a la hora de elegir las n configuraciones que serían elegidas para calcular los valores medios buscados. En la elección de esas n configuraciones se toma en cuenta la probabilidad exponencial de que se presente una configuración determinada con energía potencial E_i, y por lo tanto el proceso es el siguiente:

ALGORITMO DE METRÓPOLIS

PROPÓSITO: Usar el recocido para la solución de problemas combinatorios.

DESCRIPCIÓN

PASO 1 Se colocan las N partículas de la configuración en un cuadrado de arista L, cada una con coordenadas $<x_i, y_i>$. Se define una longitud máxima de desplazamiento en un movimiento β.

PASO 2 Se calculan dos números aleatorios U_1, U_2 uniformes en [-1, +1].

PASO 3 Se calcula la nueva posición de la partícula $x_{i+1} = x_i + \beta U_1$, $y_{i+1} = y_i + \beta U_2$ (En caso de que se excedan los límites del cuadrado, se entraría por el lado opuesto).

PASO 4 Se calcula la diferencia de energía δ de $<x_{i+1}, y_{i+1}>$ respecto a $<x_i, y_i>$.

PASO 5 Se elige la nueva configuración según la siguiente lógica:

5.a Si $\delta<0$, se acepta directamente $<x_{i+1}, y_{i+1}>$como complexión por considerar.

5.b Si no, se acepta $<x_{i+1}, y_{i+1}>$ con una probabilidad $p = \exp(-\delta/kT)$ (para lo cual genera un $U' \in [0,1)$ y se acepta $<x_{i+1}, y_{i+1}>$ si $U'<p$), quedándonos de nuevo con $<x_i, y_i>$ en caso contrario.

PASO 6 Se vuelve al paso 2 para calcular la siguiente configuración a partir de la actual.

ANALOGÍA FÍSICA

TERMODINÁMICA		OPTIMIZACIÓN
Configuración	⇔	Solución Factible
Configuración Fundamental	⇔	Solución Optima
Energía de la Configuración	⇔	Costo de la Solución
Temperatura	⇔	Parámetro

Como se ve, al concepto físico de temperatura no le correspondería un significado real en el campo de la optimización, sino que se considera como un parámetro T que se irá ajustando. De esta manera se podrían imaginar similares los procesos que ocurren cuando las moléculas de una sustancia van colocándose en los diferentes niveles energéticos buscando el equilibrio a una determinada temperatura, y los que ocurren en los procesos de minimización en optimización local (para maximización sería semejante): en el primer paso, fijada la

temperatura, la distribución de las partículas en los diferentes niveles sigue la distribución de Boltzmann, por lo que cuando una molécula se mueve, ese movimiento será aceptado en la simulación si la energía disminuye, o bien con una probabilidad proporcional al factor de Boltzmann en caso contrario; al hablar de optimización, fijado el parámetro T, producimos una perturbación, aceptando directamente la nueva solución cuando su costo disminuye, o bien con una probabilidad proporcional al "factor de Boltzmann" en caso contrario.

Esta es la clave del recocido simulado, ya que básicamente es una estrategia heurística de búsqueda local, en la cual la elección del nuevo elemento del entorno N(s) se realiza aleatoriamente, y como se vio con anterioridad, este tipo de estrategia presenta el inconveniente de que si durante el proceso de búsqueda se cae en un óptimo local, la técnica no sea capaz de salir de él. Para evitarlo, el recocido simulado permite con una cierta probabilidad (cada vez menor conforme nos acercamos a la solución óptima) el paso a soluciones peores. Analizando el comportamiento del factor de Boltzmann vemos que, conforme disminuye ésta, disminuye rápidamente la probabilidad de que aceptemos una solución peor que la actual. Ver la figura 6.7.

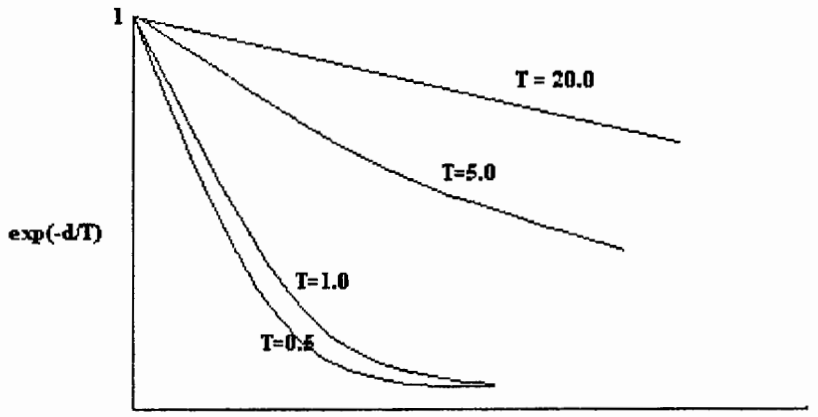


Figura 6.7

Por lo tanto, la estrategia que seguiremos en el recocido simulado será partir de una "temperatura" alta (con lo cual permitimos cambios a soluciones peores en los primeros pasos, cuando aún estamos lejos del óptimo global), y posteriormente iremos reduciendo la temperatura, disminuyendo la posibilidad de cambios a soluciones peores cuando ya nos hallamos más cercanos al óptimo buscado. El algoritmo puede representarse de la siguiente manera:

EL ALGORITMO DE RECOCIDO SIMULADO

PROPÓSITO: Resolver problemas de tipo combinatorio usando este heurístico.

DESCRIPCIÓN

PASO 1 Se selecciona la temperatura inicial T_0, la velocidad de enfriamiento (o sea, la forma de cálculo α del valor T_{i+1} a partir de T_i cuando se disminuye la temperatura, tras haber estado L(T) iteraciones en esa T), y la temperatura final T_f

PASO 2 Se genera una solución inicial perteneciente al espacio de soluciones Ω, y mientras no se llegue al final del proceso, para cada T se calcula un número L(T) de veces (antes de disminuir la temperatura) una solución que esté en el entorno $N(S_{act})$ de la actual, la cual sustituirá a ésta cuando se tiene un menor costo, o bien con una probabilidad exp $(-\delta/T)$. Para el cálculo de esa probabilidad se genera un número aleatorio uniforme [0,1), que se representa como U(0,1). Finalmente la solución ofrecida es la mejor de todas las S_{act} visitadas.

Para efectos de sencillez se considera que el algoritmo finaliza una vez que se ha alcanzado una temperatura mínima T_f, pero en general la condición de detenimiento es cuando se tiene que en tres temperaturas sucesivas no se produce un número mínimo aceptable de mejoras de la solución.

EJEMPLO 6.1
En el problema del agente viajero, las soluciones son permutaciones de las ciudades por visitar, si son n ciudades, la solución <cp(1),cp(2),...,cp(n)> indica que se parte de cp(1), luego se va a cp(2), y tras recorrer el resto se vuelve finalmente a cp(1). La implementación de recocido simulado más sencilla en este problema puede ser la siguiente:

Se genera aleatoriamente una solución inicial S_{act} (una permutación de las n ciudades) y se evalúa su costo como la suma de los recorridos. Se fija una temperatura inicial y un programa de enfriamiento de acuerdo a lo siguiente:

Como entorno $N(S_{act})$ podríamos definir la permutación obtenida al intercambiar dos elementos de S_{act}. Por lo tanto, elegimos aleatoriamente las ciudades i y j e intercambiamos su posición en S_{act} obteniendo S_{cand}. Evaluamos el costo de S_{cand} y si es menor que el de S_{act} hacemos $S_{act} \leftarrow S_{cand}$. En caso contrario hallamos $\delta = c(S_{cand})-c(S_{act})$, exp$(-\delta/T)$ y un número aleatorio U en [0,1), si U< exp$(-\delta/T)$, hacemos $S_{act} \leftarrow S_{cand}$, y en caso contrario, repetimos el proceso para el mismo S_{cand}. Cada cierto número de iteraciones se reduce la temperatura T (de acuerdo con el programa de enfriamiento definido) y cuando se llegue a la Tf final, se para, dando como solución del agente viajero la mejor de las permutaciones visitadas.

SELECCIÓN DEL PROGRAMA DE ENFRIAMIENTO

Como es obvio, no hay reglas generales para elegir el mejor "programa de enfriamiento" (conjunto de parámetros que gobiernan el enfriamiento) sino que la decisión es muy dependiente de las ideas generales asociadas con esos parámetros.

Tratando de minimizar el tiempo de CPU que se consideraba "inútil", se hicieron numerosos intentos para encontrar el programa de enfriamiento óptimo.

La expresión resultante depende de dos constantes: el máximo cambio en la función objetivo debido a uno de los movimientos que se utilizan y el "diámetro" del espacio de búsqueda. Esto permitía conservar la convergencia al óptimo e intentar buscar mejores programas de enfriamiento. Uno de los primeros intentos de automatizar el proceso de enfriamiento fue el desarrollado por S. R. White [1984] quien eligió, como la máxima temperatura que se justificaría utilizar, aquella que es proporcional a la desviación típica del costo de la función objetivo muestreada en configuraciones tomadas al azar. Esta elección, bastante razonable, fue seguida en líneas generales por otros investigadores. En general, es parte del "folklore" de SA que un buen programa de enfriamiento es aquel en el cual el equilibrio es alcanzado o aproximado en cada temperatura, a pesar de que existen buenos motivos para creer que el tiempo requerido es exponencial en la inversa de la temperatura [Sorkin, 1992].

TEMPERATURA INICIAL, T_0

Podríamos decir que, en general, una de las características que debe cumplir toda heurística de búsqueda es la de no ser dependiente de la solución inicial de partida. Esto lo consigue el recocido simulado partiendo de una temperatura inicial alta, con lo cual al principio irá erráticamente recorriendo soluciones lejanas de la óptima. Sin embargo, no parece conveniente considerar para T_0 valores fijos independientes del problema. Veamos un ejemplo. Si la solución inicial S_0 tiene un costo $c(S_0)$, y la seleccionada aleatoriamente en su entorno un costo $c(S_{cand}) = c(S_0) + 100$ ¿es aceptable pasar a ese nuevo punto?. Evidentemente, la respuesta depende del orden de magnitud de los costos: si $c(S_0)$ vale 0.01 no parece conveniente el movimiento, pero si es del orden de 10^6, no sería desdeñable, incluso para pequeños valores de T. Podríamos, por tanto, tratar de determinar T_0 en función del porcentaje de aceptaciones deseables.

Consideraremos que pudiera ser aceptable con un tanto por uno ϕ de probabilidad una solución que sea una μ por uno peor que la inicial S_0. Entonces, si es $c(S_{cand}) = (1+\mu)\, c(S_{act})$, será $= c(S_{cand})-c(S_{act}) = \mu c(S_{act})$, y por tanto

$$\phi = e^{-\delta / T_0} = e^{(-\mu/T_0)c(S_{act})} \Rightarrow T_0 = \frac{\mu}{-\ln(\phi)} c(S_{act})$$

Si, por ejemplo, consideramos como valores razonables dentro de la lógica que estamos siguiendo aceptar aproximadamente un ϕ = 13% de las veces (en la primera iteración) una solución que sea un μ = 1% peor que la actual, tendríamos

$$T_0 = \frac{0.01}{-\ln(0.13)} c(S_{act}) \approx 0.005 c(S_{act})$$

VELOCIDAD DE ENFRIAMIENTO

Siguiendo con la analogía, el valor L(T) debería ser lo suficientemente grande como para que el sistema llegue a alcanzar su estado estacionario para esa temperatura T, lo cual significaría un número de iteraciones al menos igual a $card(\Omega)^2$. Como esto no es factible, al definir el algoritmo hay que lograr un equilibrio entre los parámetros α y L, ya que si se realiza una búsqueda durante pocas iteraciones para cada temperatura, son recomendables rápidos decrementos de temperatura y viceversa.

En ambos extremos de este postulado están los trabajos de Lundy y Mees [1986], y de Connolly [1990]. El primero realiza una sola iteración para cada temperatura (es decir, L(T) = 1 $\forall$ t). Definen un límite superior U = máx{c(S_j) –c(S)} $\forall$ S_j, $S \in \Omega$, Y para cada iteración consideran un cambio de temperatura

$$\alpha(T) = \frac{T}{1+\beta T}$$

para una β <<U-1. De este modo se evitan grandes desviaciones respecto al equilibrio en cada paso. Realizan la elección de T^0 de tal forma que en las primeras iteraciones se aceptan la mayoría de los movimientos (tomando una T_o >>U, con lo cual consigue que el factor de Boltzmann sea próximo a uno.)

TEMPERATURA FINAL, T^f.

En teoría la temperatura correspondiente al "sistema frío" debería ser T = 0. Sin embargo, bastante antes de llegar a ese valor es prácticamente nula la probabilidad $e^{-\delta/T}$ de que se acepte un movimiento hacia una solución peor. Por lo tanto, normalmente se puede finalizar con valores $T_f > 0$, sin pérdida de calidad en la solución.

Siguiendo esta idea está el criterio de sistema frío de Lundy-Mees, el cual consiste en finalizar en cuanto la temperatura sobrepase un umbral mínimo T_f. Ellos han demostrado que, sí se considera que es θ la probabilidad de que finalmente se obtenga una solución para la cual su costo menos el de la óptima global sea cuando mucho ε, se cumple que

$$\frac{\theta}{1-\theta} > (n-1)e^{\frac{-\varepsilon}{T}}$$

siendo n = card(Ω) el número de elementos del espacio de soluciones. Despejando la temperatura, obtenemos que, para esos niveles de faibilidad, el valor límite T_f será:

$$T_f < -\frac{\varepsilon}{\ln(n-1)-\ln(\theta)+\ln(1-\theta)} \approx \frac{\varepsilon}{\ln(n)}$$

Al seguir esta estrategia, consiguen una acotación polinomial en la complejidad de su algoritmo, ya que entonces, tomando un valor adecuado para β, el número de iteraciones que se realizarán será menor a $\ln(n)/(\beta\varepsilon)$, es decir, proporcional a ln(n).

Otros criterios utilizados de sistema frío consisten en detener la búsqueda cuando para una T no se haya conseguido obtener una solución que mejore la mejor solución hallada hasta ese instante. Por ejemplo, Johnson et al [1989], para el problema de "partición de un grafo" (dividir los n vértices de un grafo en dos conjuntos de igual cardinalidad, de tal forma que haya un mínimo número de arcos uniendo ambos subgrafos), utilizan un r = 0.95, L(T) = 16 card($N(S_{act})$), parando (de modo análogo a como lo hace el trabajo pionero de Kirkpatrick et al [1983]) cuando se hayan procesado 5 temperaturas para cada una de las cuales no haya habido un 2% de movimientos aceptados, sin que en ese lapso se haya mejorado el costo de la solución actual.

ASPECTOS COMPUTACIONALES

En todo este tipo de algoritmos es fundamental tratar de reducir el tiempo de cómputo. Como los procesos que en cada iteración deben repetirse son la elección aleatoria de S_{cand}, el cálculo de $c(S_{cand})$ y la evaluación de $e^{-\delta/T}$, todo intento por mejorar la eficiencia de estas operaciones va a redundar en unos mejores resultados de la metaheurística. Se han propuesto algunas modificaciones al algoritmo básico con el fin de mejorar estos aspectos.

En primer lugar, la elección aleatoria del candidato en $N(S_{act})$ puede producir ineficiencias: si en el entorno de un punto está directamente el óptimo global, podemos necesitar incluso más de card(Ω) iteraciones para escogerlo aleatoriamente. Una alternativa consiste en fijar de antemano un orden de visita de los elementos del entorno en la fase final de la búsqueda. De este modo, cada vez que se realice una iteración, simplemente se escoge el siguiente de la lista, que pasa a ser el Scand que se evalúa. Esta forma de actuar ha dado mejores resultados para el problema de asignación que la elección aleatoria de los puntos del entorno, produciendo un efecto similar al de *búsqueda tabú* ya que de este modo se evita caer en ciclos de movimientos.

Otro aspecto que mejora el tiempo de computación reside en el cálculo del factor de Boltzmann: Johnson et al [1989] tras observar que en su problema la tercera parte del tiempo de computación se emplea en calcular las funciones

exponenciales, proponen crear una tabla en la que miran directamente esos valores. El tamaño de la tabla ellos lo han fijado en 1000 posiciones, tras considerar que valores no automáticamente rechazables o aceptables son, para cada T, los $\delta \in [T/200, 5T]$. Cada posición representa el $\delta = i\ (T/200)$, y por lo tanto, en la posición i se almacenará el valor $e^{-\delta/T} = e^{-i/200}$. El índice para mirar en esta tabla se obtiene calculando $\delta(200/T)$, redondeando y acotando los valores entre 1 y 1000.

Otra forma utilizada por otros autores para ahorrar tiempo de computación es realizar un cálculo sólo aproximado del valor δ, o bien definir convenientemente las funciones de costo c(S) de forma que el cálculo de c(Scand) pueda hacerse en menos tiempo partiendo del valor c(Sact) ya conocido.

En el problema TSP, si la solución actual tiene una longitud $c(S_{act}) = 1000$ y S_{cand} se obtiene al suprimir de S_{act} unos recorridos de longitud 25 y añadir otros de longitud 15, no es preciso volver a sumar las longitudes de todos los recorridos de la solución S_{cand}, sino que bastaría con hacer $1000 - 25 + 15 = 990$.

APLICACIONES DEL RECOCIDO SIMULADO

A pesar de la corta vida de esta metaheurística, son ya numerosos los problemas que han sido resueltos mediante recocido simulado, en todo tipo de optimización combinatoria: diseño de componentes eléctricos, procesamiento de imágenes, simulación física, distribución de recursos, calendarización y criptografía entre otros.

En muchas ocasiones, lo que se ha realizado es la conjunción del recocido simulado con otras heurísticas, lo cual puede realizarse de dos maneras diferentes:

a) Usando el recocido simulado para generar una solución inicial que utilizará otra heurística.
b) Usando el otro algoritmo para generar una solución inicial que utilizará el recocido simulado.

La primera aproximación podría consistir, por ejemplo, en realizar una búsqueda exhaustiva que garantice al menos el óptimo local asociado con la última solución dada por el recocido simulado. La segunda ha sido utilizada, por ejemplo, para obtener k-coloraciones de grafos, es decir, dividir los n vértices de un grafo en k conjuntos inconexos.

6.6 ALGORITMOS GENÉTICOS

Supongamos que nos enfrentamos al problema de encontrar el edificio más alto de Nueva York con los ojos vendados. En esta situación hipotética, para establecer una analogía con un problema de optimización combinatoria, también supondremos que, si bien estamos con los ojos vendados, podemos caminar hasta una dirección determinada, entrar en un edificio y "computar" su altura. Tenemos que decidir cuál es la estrategia que se seguirá si no se pueden visitar todos los edificios (estamos descartando lo enumeración exhaustiva como método válido). No obstante, vamos a suponer, por analogía con un algoritmo de computación, que no hay ningún tipo de "cansancio".

Tenemos que diseñar una estrategia. Por ejemplo, partiendo de un punto inicial elegido al azar, caminar diez kilómetros en una dirección elegida también al azar, medir la altura del edificio encontrado, y repetir este procedimiento durante un cierto número fijo de interacciones. Finalmente, nos quedaremos con la altura del más alto encontrado durante todo el proceso. Esta estrategia está basada en una caminata aleatoria y su eficiencia es evidentemente muy pobre para este problema, debido a que podríamos dirigirnos a regiones de muy baja altura y permanecer allí durante mucho tiempo.

Podemos entonces añadir una componente probabilística a esta estrategia. Al iterar, sólo aceptaremos un nuevo edificio como punto de referencia de un nuevo paso si satisface algún criterio predefinido, en analogía con aquellos utilizados por el recocido simulado, o como los empleados en búsqueda tabú, que más adelante se verán en estos apuntes. Al final, el mejor edificio encontrado es el elegido.

Supongamos ahora que tenemos los resultados obtenidos con estas dos últimas estrategias. Sería sorprendente si fueran mejores que los dados por búsqueda aleatoria. Pero si en lugar de ir de un punto a otro utilizando pasos de 10 Kilómetros utilizamos pasos del orden de unos cientos de metros, entonces es más probable que el resultado final fuese mejor que en el caso aleatorio. La razón es que no esperamos ninguna "correlación" fuerte entre las alturas de dos edificios que se encuentran a diez kilómetros. Debido a que las ciudades suelen tener los edificios más altos bastante localizados en áreas pequeñas, esperamos que la búsqueda utilizando pasos de cientos de metros sea más eficiente en la explotación de esta correlación.

Nuestro "infatigable optimizador" puede decidirse entonces por empezar a añadir más parámetros a su estrategia de búsqueda, como por ejemplo cambiar dinámicamente (es decir, basándose en la información encontrada hasta ese momento) los parámetros como la dimensión de la lista tabú, el criterio de aspiración, etc., o los parámetros que gobiernan los métodos de tipo SA, como el programa de enfriamiento, recalentamiento, cambio de la longitud de paso, etc. No descartando que se puedan obtener buenos resultados con estas extensiones, la introducción de nuevos parámetros sólo refleja nuestra

ignorancia acerca de cómo resolver elegantemente estas cuestiones, las cuales necesitan una solución.

Un cambio radical estaría dado por el uso de más de una persona "optimizadora" para ese problema, es decir, un método basado en una "población". Esto nos lleva a considerar otro tipo de cuestiones relacionadas con cuál es la mejor manera de organizar ese equipo para hacer más eficiente la búsqueda. Supongamos que comenzamos con cierto número de optimizadores dispersos en Nueva York. Cada uno de ellos mide la altura del edificio que está visitando, luego todos comunican este resultado y su posición a los otros. Entonces podemos hacer que estos optimizadores comiencen desde nuevas posiciones, las cuales serán generadas por procesos que seleccionen más frecuentemente aquellos optimizadores que estén ubicados en los edificios más altos (esperando entonces que exista una correlación). Cada vez que se desee generar una nueva dirección, se tomarán dos o más optimizadores (como "padres" de la nueva población por crear) y se establecerá una nueva posición por algún procedimiento aleatorio que sin embargo garantice que esa posición se encuentre, "en promedio", cerca de las soluciones "padres". El reposicionamiento de los optimizadores estará, entonces fuertemente influenciado por aquellos que se encuentren mejor situados.

Obviamente, la estrategia descrita anteriormente guarda una gran similitud con los Algoritmos Genéticos (AG) que son objeto central de esta sección. Análogamente han despertado la curiosidad de un gran número de investigadores que se han dedicado a las casi infinitas variantes de este esquema básico variando el número de individuos, el proceso de selección de los padres de la nueva población, modificaciones al proceso de creación de nuevas direcciones de búsqueda (recombinación), el efecto de realizar "pequeños cambios aleatorios" (mutaciones) al producto de la recombinación, el efecto de usar distintas representaciones del problema, etc.

Por otro lado cierto número de investigadores ha reconocido que para algunos problemas de optimización existe una gran literatura de heurísticas rápidas de búsqueda interactiva (búsqueda local). Han optado por adaptar para esos problemas un método de búsqueda local (a lo mejor basado en metaheurísticas como búsqueda tabú o recocido simulado) con un método basado en el uso de una población y la utilización de procesos de recombinación como los de AG. Este tipo de metaheurísticas han sido clasificadas bajo la denominación común de "Algoritmos Meméticos" (AM).

ANALOGÍAS CON LA EVOLUCIÓN.

En la Naturaleza, la Evolución, en particular la de los seres vivos, tiene algunas características que motivaron a John Holland a comenzar una línea de investigación en un área que eventualmente se transformó en lo que hoy se denomina Algoritmos Genéticos (AG). La habilidad de una población de

cromosomas para explorar el espacio de búsqueda "en paralelo" y combinar lo mejor que ha sido encontrado en él mecanismo de sobrecruzamiento (crossover), es algo intrínseco a la evolución natural y trata de ser explotada por los AGs.

Los algoritmos desarrollados inicialmente por Holland eran simples, pero dieron soluciones satisfactorias a problemas que eran considerados como "difíciles" en aquel tiempo.

El campo de los AGs. ha evolucionado desde entonces, principalmente debido a las innovaciones introducidas en la década de 1980. El propio Holland ha publicado en 1992 un artículo de divulgación sobre AGs.

Las características de la evolución que hay que tener en cuenta están descritas brevemente por Davis [1991] de la siguiente manera:

- La evolución es un proceso que opera en los cromosomas en lugar de en los seres vivos que ellos codifican.
- Los procesos de selección natural provocan que aquellos cromosomas que codifican estructuras con éxito se reproduzcan más frecuentemente que aquellos que no lo hacen.
- Las mutaciones pueden causar que los cromosomas de los hijos sean diferentes a los de los padres y los procesos de recombinación pueden crear cromosomas bastante diferentes en los hijos por la combinación de material genético de los cromosomas de los dos padres.
- La evolución biológica no tiene memoria.

Tras la publicación del libro de Holland (1975) *"Adaptation in Natural and Artificial Systems"* y de los numerosos investigadores que los utilizan como metaheurística para optimización, es que se pueden encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un proceso de "evolución simulada", en particular como un algoritmo matemático implementando en una computadora. Debido al trabajo original de Holland, esto inicialmente ocurría en forma de algoritmos que manejan un string binario, a los que se denominó "cromosomas", ya que ellos representaban un punto en el espacio de configuraciones del problema.

Como ocurre en la evolución biológica, la evolución simulada estará diseñada para encontrar cada vez mejores cromosomas mediante una manipulación "ciega" de sus contenidos. El término "ciega" se refiere al hecho de que el proceso no tiene ninguna información acerca del problema que está tratando de resolver, exceptuando el valor de la función objetivo. En la concepción original de los AGs, la función objetivo es la única información por la que se evalúa el "valor" de un cromosoma. Por lo tanto, estas metaheurísticas están basadas en integrar e implementar eficientemente dos ideas fundamentales: la habilidad de representaciones simples (como por ejemplo bit strings) para codificar

configuraciones de nuestro problema de optimización, y el poder de transformaciones simples para modificar y mejorar estas configuraciones. Las mejoras son guiadas mediante un mecanismo de control que permite a una población de cromosomas evolucionar como las poblaciones de seres vivientes lo hacen. Un algoritmo genético puede ser visto como una estructura de control que organiza o dirige un conjunto de transformaciones y operaciones diseñadas para simular estos procesos de evolución.

En la Evolución de los seres vivos, el problema al que cada individuo se enfrenta cotidianamente es el de la superviviencia. Para ello cuenta con las habilidades innatas provistas por su material genético. En el ámbito de los genes, el problema es el de buscar aquellas adaptaciones beneficiosas en un medio ambiente hostil y cambiante. Debido en parte a la selección natural, cada especie gana una cierta cantidad de "conocimiento", el cual es codificado e incorporado en la nueva conformación de sus cromosomas. Esta conformación se ve alterada por las operaciones de reproducción. Algunas de ellas son las mutaciones aleatorias, inversión de partes del cromosoma y el sobrecruzamiento, que es el intercambio de material genético proveniente de dos cromosomas padres.

En los AGs las mutuaciones aleatorias proveen cierta variación y ocasionalmente introducen alteraciones beneficiosas en los cromosomas. La "inversión" es un mecanismo que altera la ubicación de los genes en los cromosomas, permitiendo que algunos genes que se han coadaptado exitosamente se ubiquen más cerca en el cromosoma, lo que a su vez permite que se aumente la probabilidad de moverse en conjunto durante el sobrecruzamiento. Este último es el mecanismo responsable del intercambio de material genético entre dos padres para ser combinado en las estructuras hijas (Davis, 1987).

Evidentemente, es la existencia del sobrecruzamiento lo que gobierna la eficiencia de los algoritmos genéticos y los destaca nítidamente de otro tipo de metaheurísticas. Con él, las características de los padres se pueden ser combinar inmediatamente cuando se reproducen. Por lo tanto, la probabilidad de esta combinación se acrecienta cuando los padres tienen un nivel elevado de fitness (aptitud) debido a que se reproducen con mayor frecuencia.

A continuación discutiremos cómo se han introducida en los AGs las principales características de la evolución en varios niveles y también haremos una síntesis de las tendencias y los desarrollos más recientes en este campo.

COMPOSICIÓN DE UN ALGORITMO GENÉTICO

En el contexto de encontrar la solución óptima a problemas de optimización combinatoria de gran escala, un algoritmo genético básico funciona de la siguiente manera. Una población de cromosomas (los padres potenciales de

una nueva población) se mantiene a lo largo de todo el proceso evolutivo. A cada uno de ellos se le adjudica un valor de fitness que está relacionado con el valor de la función objetivo por optimizar. Cada cromosoma está, pues, representando un punto del espacio de configuraciones (el espacio de búsqueda) del problema.

Dos cromosomas (originalmente sólo restringidos a strings binarios), son seleccionados para ser los padres de una nueva solución mediante el mecanismo de sobrecruzamiento (o como dicen algunos genetistas "entrecruzamiento", crossover en inglés). En los algoritmos originales de Holland, uno de ellos era elegido de acuerdo a su valor de fitness (a mayor valor de fitness mayor probabilidad de ser elegido), mientras que el otro padre era elegido aleatoriamente. La operación de sobrecruzamiento daba dos configuraciones binarias "hijos" y se calculaban sus valores de fitness. Estas soluciones reemplazaban dos soluciones de la población que eran elegidas al azar. Este proceso se repetía tantas veces como se necesitaba (es decir, hasta que se alcanzaba algún criterio más o menos lógico para detener la simulación). Esto es una breve descripción de un modelo muy simple de evolución que tratara de incorporar los conceptos de supervivencia y selección del más apto, así como el de la reproducción sexuada.

Desde sus inicios, se han dedicado grandes esfuerzos a estudiar las múltiples variaciones de este esquema básico de los AGs. Independientemente de lo sofisticado que pueda ser el diseño de un AG, existen cinco componentes que deben ser incluidas.

- Una representación, en términos de "cromosomas", de las configuraciones de nuestro problema.

- Una manera de crear las configuraciones de la población inicial.

- Una función de evaluación que permite ordenar los cromosomas de acuerdo con la función objetivo.

- Operadores "genéticos" que permiten alterar la composición de los nuevos cromosomas generados por los padres durante la reproducción.

- Valores de los parámetros que el algoritmo genético usa (tamaño de la población, probabilidad asociadas con la aplicación de los operadores genéticos, etc.).

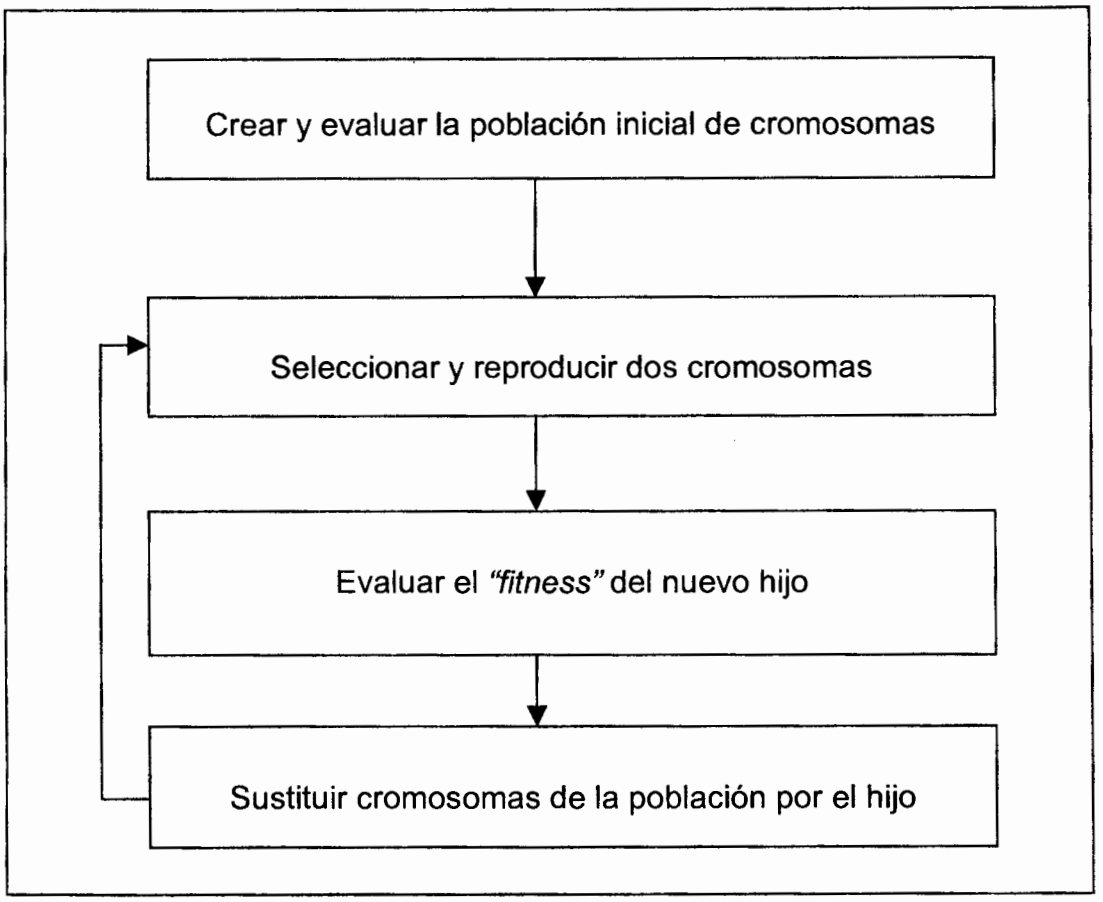


Figura 6.8. Esquema básico de los algoritmos genéticos.

Debido a que el éxito de una implementación de un AG es altamente dependiente de la selección adecuada de cada uno de sus componentes, trataremos posteriormente los temas en el diseño de un AG.

ELEMENTOS BÁSICOS

En los trabajos de Holland y en los realizados por varios de sus seguidores, los cromosomas eran representados por strings binarios (es decir, listas de ceros y unos). Este tipo de codificación ha sido utilizada en muchos campos distintos, incluso en algunos en los que esto no es particularmente obvio. Una de las ventajas es que este tipo de representación ha sido analizada (en la teoría de los AGs) con mayor detalle. También, una representación binaria es una codificación natrual para un gran número de problemas de optimización como por ejemplo el problema de la mochila. En este problema el string 10011 puede representar la configuración ($x_1 = 1$, $x_2 = 0$, $x_3 = 0$, $x_4 = 1$, $x_5 = 1$)

EJEMPLO 6.1

E el contexto del TSP la codificación anterior no es tan natural. Una posibilidad es la siguiente, para un TSP de cinco ciudades y 20 arcos, un string 01011000001000100000 puede representar una ruta que usa los arcos 2,3,4,11 y 15 para conectar las cinco ciudades (ver figura 6.8). Una representación de este tipo no es la más adecuada para el TSP ya que sólo un pequeño subconjunto de los cromosomas son soluciones factibles del TSP (obviamente muchos de los cromosomas que contienen cinco "1" y quince "0" no forman un circuito válido)

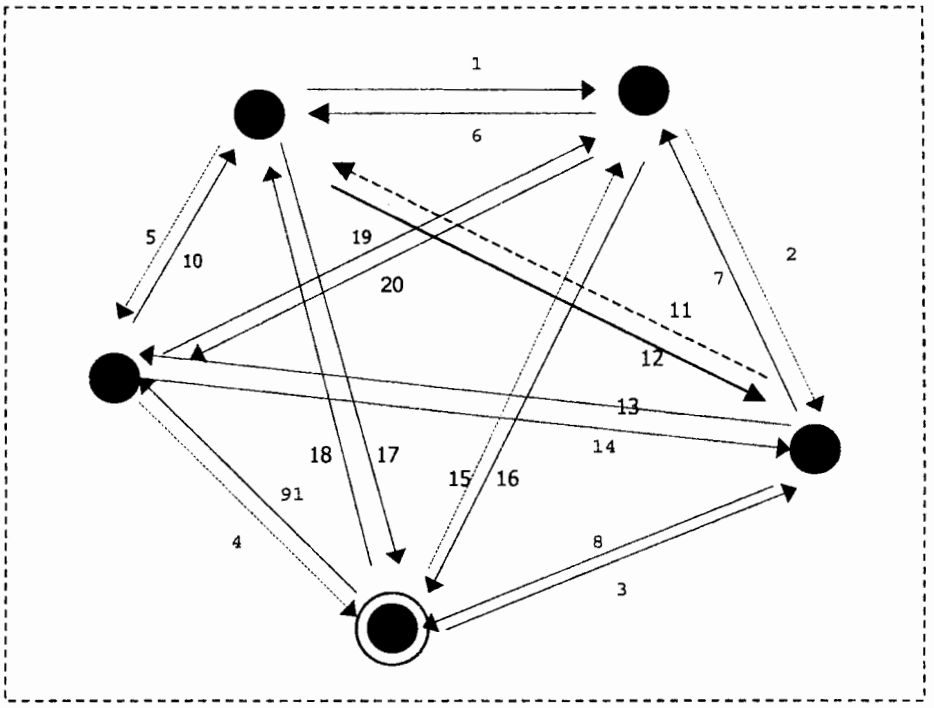


Figura 6.9. Ejemplo de un TSP con 5 ciudades.

Debido a que las representaciones binarias algunas veces no son efectivas en un gran número de problemas prácticos, algunos investigadores han comenzado a explorar el uso de otras representaciones, en general en conexión con aplicaciones industriales de los AGs. Ejemplos de otras representaciones incluyen las listas ordenadas en empacados binarios, listas combinadas en calendarización de la producción y listas de variables-elementos en arreglos de semiconductores.

En el caso del TSP, una ruta se puede representar por una lista ordenada de ciudades. Esta representación es más natural por las características propias del problema, pero requiere la creación de operadores genéticos específicos para él. Por lo tanto, la selección de una representación está fuertemente asociada con el diseño de operadores al utilizar AGs.

En el contexto de problemas de optimización combinatoria, la dificultad de utilizar strings binarios también está asociada con el hecho de tener que encontrar una representación en la cual los substrings tengan un significado relevante para el AG. Algunas de las representaciones que son consideradas como más naturales no se ven apoyadas por la noción de *schemata* que dan una buena base matemática a los AGs, aunque no obstante, algunos investigadores afirman que se obtienen buenos resultados apartándose de los alineamientos básicos de la teoría estándar de los AGs y algunos llegan a afirmar que el éxito mismo se debe a la violación de estos principios.

Independientemente de cómo se entienden estas codificaciones, su propósito es identificar características similares en el espacio de configuraciones (muestreado por la población) mientras se mantiene una conexión más cercana a la representación. En general, diferentes codificaciones dan diferentes perspectivas y una diferente resolución para investigar el espacio de configuraciones. Estas consideraciones están relacionadas con el concepto de deceptiveness.

LA POBLACIÓN INICIAL

La población inicial de un AG puede ser creada de diferentes maneras. Aunque parezca extraño, al enfrentarse con un problema nuevo se puede aprender mucho inicializando esta población de manera aleatoria. Hacer evolucionar una población que ha sido generada aleatoriamente hasta llegar a tener una bien adaptada (es decir, con configuraciones satisfactorias que sean soluciones aproximadas del problema de optimización), es una buena prueba para saber cómo está funcionando la implementación del AG debido a que características esenciales y críticas de la solución final deben ser consecuencia de este proceso evolutivo y no de las características de los métodos usados para generar la población inicial.

Para algunas aplicaciones, especialmente las de la industria, puede ser conveniente inicializar usando métodos más directos. Estas técnicas incluyen la perturbación de la salida de algún algoritmo goloso, weighted random initialization, o inicialización por perturbaciones aleatorias con una regla de selección golosa es la denominada GRASP, que en el próximo capítulo se estudia a profundidad.

Supongamos que n artículos de un problema de la mochila están ordenados de tal manera que si j>i entonces $(c_j/v_j) \geq (c_i/v_i)$ donde c_i y a_i son los coeficientes de la función objetivo y de los coeficientes de restricción para el artículo i. Una fase de construcción GRASP se puede designar para que, en cada paso, se seleccione un artículo de una lista de K mejores candidatos. La lista consiste en los K artículos que están fuera de la mochila y que tiene los menores valores de índices y además sus coeficientes de restricciones individuales no exceden la

capacidad remanente de la mochila. La construcción habrá finalizado cuando la lista de candidatos esté vacía, porque la mochila ya está llena o porque la capacidad aún existente en ella no es suficiente para poner ningún otro artículo. El valor de K controla el nivel de aleatoriedad, de manera que si K = 1 el procedimiento es completamente determinista y la misma solución golosa se produce siempre, mientras que si K = n el procedimiento es completamente aleatorio. La construcción en este caso está forzada a ser factible (es decir, los i artículos dentro de la mochila no violan la capacidad). Sin embargo, construcciones no factibles se pueden generar también, por ejemplo, aumentando artificialmente la capacidad de la mochila. Por ejemplo, el lado derecho V de las restricciones de capacidad puede ser multiplicado por $(1+\alpha)$ donde α es el máximo porcentaje de violación de capacidad. Este cambio se puede combinar con una modificación en la función objetivo para penalizar individuos por violar los vínculos de capacidad.

EVALUACIÓN DEL NIVEL DE FITNESS

La función utilizada para asignar los valores de fitness a los miembros de la población debe ser tal que dé como resultado una buena discriminación. Este es el motivo por el cual muchas implementaciones de AGs usan cierto tipo de proceso de "normalización", en el siguiente sentido. Supongamos que los resultados de la evolución de una función objetivo asignan un valor de 1.020 a un individuo muy bueno mientras que da un valor de 1.000 a uno que es muy malo. Si esos valores fueran utilizados sin alteración como medidas del valor de fitness en la fase de reproducción, sería necesario mucho tiempo para que los descendientes del primer individuo influyan más que los del peor individuo. La normalización debe hacer hincapié en la importancia de las mejoras, sin desechar todo el material genético relevante presente en la población.

Si la normalización está influenciada muy fuertemente por el mejor individuo, es muy posible que la búsqueda se concentre en aquella región del espacio de configuraciones cercana a él, perdiéndose así parte de las ventajas del paralelismo intrínseco de los AGs. Por otro lado, si no hay cierta presión hacia las mejores soluciones, podemos tener una búsqueda que es demasiado lenta.

Es muy importante, al diseñar una función de evaluación, tener en cuenta las restricciones del problema. La satisfacción de estas restricciones puede ser llevada a cabo al imponer penalidades en individuos que las violan o creando decodificadores de la representación que eviten generar individuos no factibles. Una función de evaluación que incorpora un término de penalización tiene la siguiente forma:

Fitness = Valor Objetivo Normalizado – Penalización *Medida Infactibilidad

En el supuesto de que a mejor individuo, mayor valor. Si el valor de la "Penalización" es alto y el dominio del problema es tal que es probable de la producción de individuos no válidos (infactibles), puede ocurrir entonces que cuando se encuentre un individuo válido, éste ejerza influencia sobre los otros y la población converja hacia él, independientemente de la búsqueda de mejores individuos. Esto puede ocurrir porque posiblemente los caminos hacia mejores individuos factibles requieren la producción de otros individuos no factibles como pasos intermedios y los valores de penalización hacen imposible que estas estructuras intermedias se hagan presentes en la población, se diversifiquen y se reproduzcan.

Una manera de ajustar dinámicamente el valor de la penalización es incorporar el concepto de "oscilación estratégica", que fue originalmente concebido como una manera de introducir diversificación en búsqueda tabú. La forma más directa es asumir que ha sido encontrada una medida de la no factibilidad, la cual cuantifica la violación de las restricciones. Por ejemplo, una heurística que en el TSP genera una solución con cuatro subrutas podría tener asignada una medida de no factibilidad proporcional a 3=(4-1), o ser aún refinada ponderando cada subruta como la inversa de su tamaño. También se asume que un rango adecuado del valor de penalización puede obtenerse para escalar la medida de no factibilidad relativa a la función objetivo normalizada. La penalización es entonces una función del tiempo, introduciendo un parámetro que controla la oscilación. Al disminuir la penalización, nuevas estructuras no factibles se pueden introducir en la población, diversificando de esta manera la búsqueda. La mejor estrategia, lamentablemente, será dependiente del problema a pesar de que, como en el caso de búsqueda tabú, puede ser eventualmente posible que algunas estrategias generales funcionen bien para cierta clase de problemas.

Una vez evaluado el fitness de un individuo, se lleva a cabo un test para comparar ese valor con el fitness asociado al mejor individuo creado durante todo el proceso evolutivo. Si el fitness del mejor individuo es superior al del mejor encontrado hasta ese momento, es reemplazado garantizando así que al término de la simulación el mejor individuo encontrado estará en la memoria a pesar de no estar presente necesariamente en la población final. La convergencia del proceso evolutivo en un individuo particular es generalmente usada como un criterio de finalización del AG.

SELECCIÓN Y OPERADORES GENETICOS DE SOBRECRUZAMIENTO

El propósito de la selección de padres es incrementar la probabilidad de reproducir miembros de la población que tengan buenos valores de la función objetivo. Una manera popular de visualizarlo es la denominada roulette wheel, una representación de una rueda que gira en la que cada individuo tiene una sección circular que es directamente proporcional al fitness del individuo. Un

paso de selección entonces sería análogo a un giro de esa ruleta, seleccionado el individuo correspondiente según el arco de círculo donde se detiene.

Una vez que los padres han sido elegidos, se utiliza un operador genético como el sobrecruzamiento. A veces, la elección de un mecanismo de selección adecuado permite un tratamiento matemático más riguroso como el de Prugel-Bennett y Shapiro.

El sobrecruzamiento es el procedimiento por el cual dos seres vivientes intercambian parte de su material genético para crear un nuevo organismo. En los AGs, el sobrecruzamiento recombina el material de dos cromosomas para generar nuevos individuos. El operador original de los AGs, modelado por analogía con los procesos de la naturaleza, es el one-point crossover. Este operador elige aleatoriamente un punto de ruptura de tal manera que el material genético más allá de ese punto es intercambiado entre dos padres para crear dos hijos.

EJEMPLO 6.2

Supongamos que tenemos dos padres p_1 = <010110100> y p_2 = <110010010>; dos configuraciones hijas generadas son ch_1 = <010010010> y ch_2 = <110110100> si suponemos que el punto de entrecruzamiento, elegido aleatoriamente, es '3' obtenemos lo siguiente:

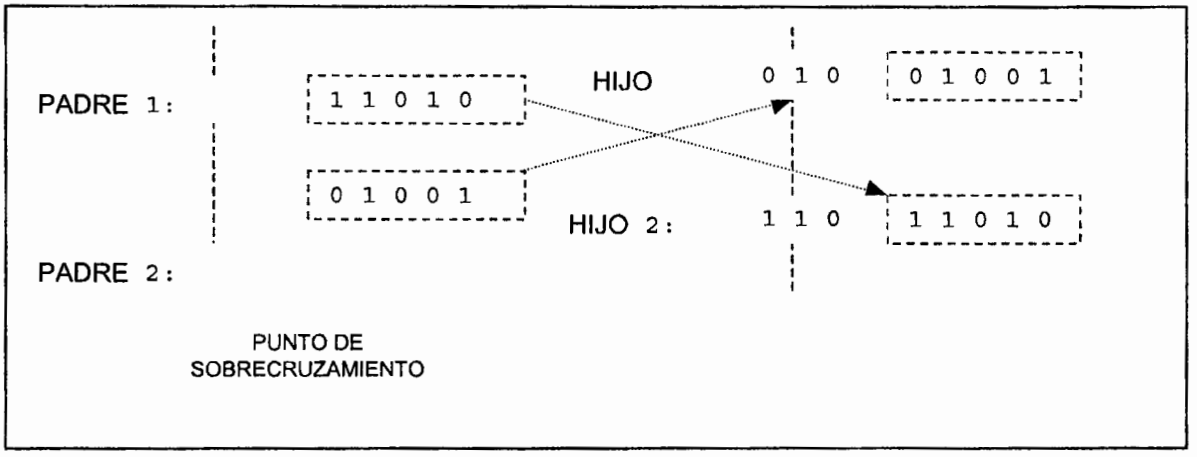


Figura 6.10 Ejemplo de sobrecruzamiento en un punto

Al irse desarrollando el campo de los AGs, se fueron haciendo notorias ciertas limitaciones del one-point crossover, principalmente al no poder combinar característica presentes en los dos cromosomas. Una solución ha sido el uso del two-point crossover. En este caso, son elegidos dos puntos de ruptura al azar y es intercambiado el material genético entre ambos.

Este sobrecruzamiento también presenta sus restricciones, lo cual llevó a Ackley en 1987 a desarrollar el "sobrecruzamiento uniforme" (uniform crossover): comenzando por el primer bit, se elege al azar un padre para que contribuya con su primer bit al primer hijo, mientras que el segundo hijo recibe el bit del segundo padre. Este proceso continúa hasta que todos los bits se han asignado. El otro proceso importante en los AG es la mutación, a pesar de que está usualmente concebida como un operador cuyo papel es secundario. En el caso binario, la mutación consiste en reemplazar con cierta probabilidad (llamada tasa de mutación, por lo general de pequeño valor) el valor de un bit. Existen básicamente dos maneras de implementarlo. La primera variante cambia el bit que la prueba de probabilidad permite (es decir, si el i-ésimo bit vale 1 y pasa la prueba de probabilidad, el nuevo string contendrá un 0 en la i-ésima posición). En la segunda variante, se genera al azar un nuevo bit para substituir el bit que pasó la prueba de probabilidad. Por tanto el 50% de las veces el nuevo bit no cambiará, y la tasa de mutación será la mitad de la de la primera variante.

Los operadores genéticos para strings de bits han acaparado la mayor atención por parte de los investigadores, en comparación con otros tipos de representación. Cuando uno se mueve de la teoría a la práctica, se tiende a usar operadores genéticos diferentes y más adaptados a los campos de aplicación.

Un operador de sobrecruzamiento que puede manipular strings no binarios es el PMX (de Partially-Matched Crossover). Dados dos cromosomas padres, el operador copia un substring de uno de los padres directamente a las mismas posiciones en el hijo. Las posiciones restantes se llenan con los valores que aún no han sido utilizados en el mismo orden en que se encuentran en uno de los padres.

EJEMPLO 6.3

Si tenemos dos strings p_1 = <1,2,4,6,3,7,5,8> y p_2 = <5,4,4,1,7,2,6,8,3> y si el substring seleccionado al azar de p_1 para ser insertado en p_2 es el <4,6,3>, esto establece una relación con el substring <1,7,2> que ocupa las mismas posiciones en p_2. Entonces la secuencia de operaciones transformarían p_2 en <5,4,4,6,3,6,8,3>, y luego, eliminando las repeticiones, quedarían <5,*,4,6,3,*,8,*> y reemplazando queda <5,1,4,6,3,*,8,*> ya que el 4 había ocupado el lugar del 1 en el substring. Continuando obtenemos <5,1,4,6,3,7,8,2>. Como se puede ver en la figura 6.11.

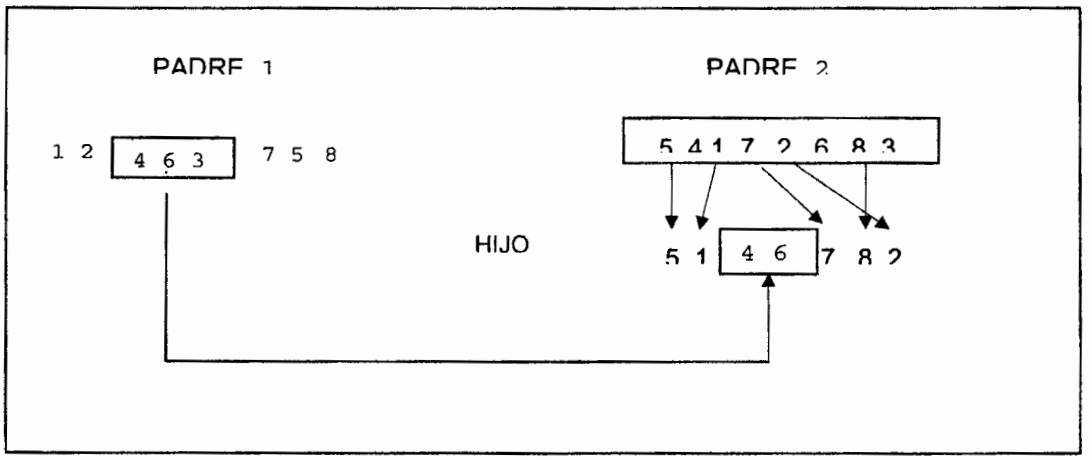


Figura 6.11. Ejemplo de aplicación del operador PMX

Otro ejemplo de sobrecruzamiento, propuesto inicialmente para el TSP, es el SEX (por Strategic Edge Crossover) introducido en 1992 por Moscato y Norman. El sobrecruzamiento comienza identificando aquellos strings (es decir conjuntos de aristas conectados) que están presentes en ambos padres. Las ciudades al final de estos strings son conectadas entre sí formando subrutas, que luego son conectadas mediante una heurística similar a la utilización por Karp.

Para construir las subrutas, es muy útil crear una estructura de datos llamada EdgeMap que puede ser definida como una lista de Edgelists. El EdgeList de una ciudad dada es la lista de las ciudades a las cuales está conectada en las dos rutas padres. Una ciudad marcada con el signo "-" indica que esta unión está presente en los dos padres. Desde una ciudad inicial elegida mediante algún procedimiento (al azar, la más alejada a las ya visitadas, la más cercana, etc.), la creación de un string involucra los siguientes pasos:

1.- Elegir la StartCity del conjunto de ciudades aún no visitadas. Se define la variable BothDirections:= False. Se define CurrentCity:=StartCity.

2.- Se retiran todas las ocurrencias del CurrentCity en la EdgeMap.

3.- Si CurrentCity tiene aún elementos en su EdgeList entonces se continúa con el paso, y si no los tiene se va al paso 5.

4.- Determinar cuál es la ciudad en el EdgeList de CurrentCity que tiene el menor número de elementos en su propia EdgeList. Ésta será la próxima CurrentCity y es insertada en el string. Los empates se resuelven aleatoriamente. Sin embargo, los empates no se rompen aleatoriamente cuando una de las ciudades tienen la misma arista en ambos padres. Tras esta secuencia se vuelve al paso 2.

5.- Si (BothDirections = False) entonces CurrentCity se declara como un EndPoint, (BothDirections: = True), CurrentCity: = StartCity y volvemos al paso 2, para comenzar a construir otra subruta desde la StartCity . En caso contrario, es decier si (BothDirections = True), entonces CurrentCity es el segundo EndPoint encontrado, y ambos EndPoints son unidos formado una subruta.

El algoritmo utilizado por Karp (ver [Lawler et al, 1985], Capítulo 6, pag. 196) puede ser adaptado para hacer el patching de las subrutas. Sea τ una permutación y sea π una permutación cíclica, es decir, una permutación con sólo un ciclo. El algoritmo de patching cambia una permutación general σ por una permutación cíclica mediante una secuencia de operaciones. Si τ es una permutación y si i y j son elementos que ocurren en dos ciclos distintos C_1 y C_2, entonces la operación de patching (i,j) une los ciclos Ci y C_2 formando un nuevo cilo mediante la inserción de las aristas (i, τ(j)) y (j, τ(i) y la eliminación de los aristas (i, τ(i)) y (j, τ(j)). Si $D_{i,\ \tau(j)}$ es el coste asociado con las aristas (i, τ(j)), entonces la operación de patching (i,j) cambia el costo de la permutación en una cantidad.

$\Delta(\tau,i,j) = D_{i,\ \tau(j)}+D_{j,\ \tau(i)}-D_{i,\ \tau(i)}-Dj_{,\ \tau(j}$

El procedimiento puede resumirse de la siguiente manera:

PATCHING DE KARP

BEGIN

$\tau \leftarrow \sigma$
{σ es la asignación inicial y τ es la permutación que se está considerando}

WHILE τ no es una permutación cíclica **DO**

BEGIN

Sean C_1 y C_2 los ciclos MAS LARGOS de τ;

{Longitud de un ciclo = número de aristas; el empate se decide aleatoriamente}

ELEGIR i $\in$ C1 y j$\in$C2 que MINIMIZAN EL COSTO DE *PATCHING* $\Delta(\tau,i,j)$; REALIZAR la operación de patching (i,j) y denominar τ a la nueva permutación;

END; {begin}
$\pi \leftarrow \tau$
END;

Otra estrategia (tentativamente llamada *nearest merger*) podría consistir en elegir las subrutas C1 y C2 para las cuales se minimiza $\Delta(\tau,i,j)$ con $i \in C_1$ y $j \in C_2$. También C_1 y C_2 podrían ser aquellas subrutas que maximizan el costo de las aristas que conectan los EndPoints. Esta última estrategia guarda analogía con la heurística de construcción llamada farthest selection, nearest insertion para el TSP ([Moscato, Tinetti, 1994]).

Existen otros operadores para problemas que están relacionados con representaciones que usan permutaciones de elementos. Incluyen el OX (por order crossover) y el CX (por cycle crossover). Para una lista de otras referencias y una explicación de estos operadores puede consultarse [Goldberg, 1989, pag. 174]. Otros operadores que tratan de preservar el orden relativo entre los elementos han sido descritos en [Moscato, 1989a]. Esto es de particular importancia en otros problemas, más allá de su motivación inicial, que fue el TSP. Otros operadores vinculados a algoritmos meméticos han sido introducidos en [Radcliffe, Surry, 1994].
Obviamente, la mutación también puede ser implementada en una representación no binaria. En el ejemplo del TSP, un hijo puede ser sujeto a mutación permitiendo a cualquier par de ciudades intercambiar posiciones de acuerdo con una cierta probabilidad. Con esta representación queda garantizado que una mutación arbitraria de este tipo sobre cualquier string siempre constituye una solución factible.

ALGORITMOS MEMÉTICOS

Una tendencia relativamente nueva en el campo de AGs es la incorporación de una técnica de búsqueda local. Tras algunos resultados desalentadores de los AGs al ser aplicados a algunos problemas (especialmente los muy difíciles) de optimización combinatoria, los investigadores comenzaron a buscar nuevas maneras para extender los AGs y obtener los mejores resultados. Un paso importante en esta dirección fue dada por Mühlenbein con el desarrollo de "algoritmos genéticos paralelos" (PGAs) que permiten que los individuos en la población mejoren su fitness mediante mejoras iterativas (conocidas como hill climbing). En los PGAs también se permite que los individuos se seleccionen entre ellos por procesos "locales" lo que facilita su implementación en sistemas concurrentes. El uso de operadores de búsqueda local también se encuentra en el trabajo de Montana y Davis [1989], Huntley y Brown [1991] , Ulder y colaboradores [991] quienes han denominado al método "búsqueda local genética".

En cierto sentido, los AGs simples con búsqueda local pueden ser vistos como métodos sofisticados de descensos múltiples. El proceso de re-inicialización hasta en este caso gobernador por reglas genéticas, y la fase de descenso se lleva a cabo como de costumbre. El éxito de estos métodos puede ser atribuido a su equilibrio entre tener una búsqueda rápida y mantener una diversidad para

evitar la convergencia prematura. Otro método puede consistir en desarrollar una búsqueda iterativa llamada Delta coding propuesta por D. Whitley y sus colaboradores. Delta coding introduce diversidad generando aleatoriamente una población nueva mientras la información de generaciones previas basando las nuevas codificaciones en soluciones parciales previas.

El método comienza haciendo una corrida inicial de un AG simple y midiendo la diversidad de la población (medida por ejemplo calculando la distancia de Hamming entre los strings padres) se perdió totalmente o se encuentra muy reducida, la mejor solución se guarda como punto inicial de la próxima iteración Delta. Ese método preserva el muestreo de hiperplanos debido a que cada corrida individual es una corrida de un AG, donde sólo la estrategia de codificación ha sido cambiada. Debido a que el número de bits usado para codificar los valores Delta es reducido en cada iteración, el espacio de soluciones está reducido.

Cada nuevo string, cuando se decodifica para evaluación de fitness, es aplicado como un nuevo valor Delta al string que se ha guardado en la iteración previa. Este proceso se traduce en la búsqueda en un espacio de solución más pequeño en torno a la mejor solución inicial. El principio subyacente de reforzar la búsqueda en regiones con soluciones de alta calidad como se noto anteriormente, en una instancia de lo que se llama estrategias de intensificación en búsqueda tabú. Las manifestaciones iniciales de la necesidad de intensificación pueden remontarse a Scatter Search que genera nuevas soluciones en determinadas regiones con relación a soluciones previas de buena calidad y a través de influir sobre los valores aplicables a variables consistentes y fuertemente determinadas. Éstas otras conexiones entre búsqueda tabú y los métodos basados en algoritmos genéticos y sus implicaciones para mezclar beneficiosamente estos métodos merecen una mayor investigación.

En el método desarrollado por Norman y Moscato, los individuos están interconectados en un anillo y cada uno de ellos realiza un período de búsqueda local (con algunas modificaciones del esquema básico del recocido simulado), compitiendo su propia existencia con los dos vecinos en el anillo, y coopera con los individuos que están distantes (teniendo en cuenta la topología de anillo). Se introducen así distintas vecindades en la competición y la cooperación. La componente cooperativa del algoritmo se basó en el uso de un operador de sobrecruzamiento que primeramente fue el OX [Goldberg, 1989], y luego fue reemplazado por el Strategic Edge Crossover (SEX) introducido por primera vez en [Moscato, Norman, 1992]. Durante la etapa de búsqueda local, un cambio que hacía cambiar la función objetivo en una cantidad ΔEMC era aceptado desacuerdo con una probabilidad dada por

$$P(\Delta E_{MC}, T) = \frac{1}{1+e^{(\Delta E_{MC}/T)}}$$

Donde T era un parámetro de control externo (interpretado como temperatura en SA).

La competencia era básicamente un desafío por la subsistencia. Cada individuo desafiaba (por la supervivencia como miembro de la población) a su vecino más inmediato en el anillo. Esta "batalla" se decidía desacuerdo a una probabilidad

$$P(\Delta E_{comp}, T) = \frac{1}{1 + e^{(\Delta E_{comp} / T)}}$$

donde si la ruta 0 desafiaba la ruta 1, y si tenían longitudes L_o y L_1 respectivamente, entonces ΔE_{comp} estaba definido como

$$\Delta \mathrm{E}_{\mathrm{comp}}(0,1) = \frac{\Delta \mathrm{L}_{0,1}}{N} = \frac{L_0 - L_1}{N}$$

Es decir que si el individuo o desafiaba al individuo 1, generábamos un número q con distribución uniforme en el intervalo [0.1] y si $q \geq P$ (ΔE_{comp} (0,1),T) nada sucedía pero si no, el individuo 1 es eliminado y una copia exacta de la ruta o lo reemplaza en el anillo.

Las fases de competencia, cooperación (reproducción de sobrecruzamiento) y búsqueda local se suceden alternándose de tal manera que una fase de búsqueda local le sigue a una cooperación, otra de búsqueda local otra de competición. Al final de estos cuatro pasos la "temperatura se disminuye según $T_n = 0{,}98 T_{n-1}$ y las cuatro fases vuelven a retirarse en el mismo orden.

La ventaja de estas secuencia cíclica es que los productos de la fase de recombinación no se vuelven a enfrentar en una competición hasta que no han sido optimizados por un proceso de búsqueda local. De esta manera se han mejorado los efectos de la recombinación (especialmente cuando un operador como el OX ha sido utilizado). El proceso de optimización se detiene cuando la diversidad del grupo es muy baja. Otra estrategia posible es, en ese momento, aumentar la temperatura nuevamente para que la población gane diversidad de nuevo (reheating) como fue originalmente propuesto en [Moscato, 1989b]. En el trabajo se analizaron otros métodos que utilizan una población y períodos de búsqueda independiente y se los denominó globalmente como *algoritmos meméticos*.

No existe motivo para que un ordenador paralelo o un sistema distribuido las fases tenia que estar en sincronismo, debido a que el método sólo requiere de una comunicación muy ocasional entre individuos y la mayor parte de la carga computacional se llevan los procesos de búsqueda local, el método no sufre pérdida de eficiencia en sistemas basados en intercambio de mensajes.

EL PROBLEMA DE LA LIGA NACIONAL DE HOCKEY (NHL)

Este esquema básico que acabamos de escribir en la sección anterior, fue utilizado por D. Costa para el programa de la National Hockey League (NHL) , un problema con un gran número de restricciones [Costa, 1992]. Utilizó un método que podemos clasificar como una instancia de un algoritmo memético donde la búsqueda local, además, estaba utilizando búsqueda tabú.

Básicamente, el problema es decidir la fecha en la que se llevarán a cabo 880 partidos entre 22 equipos, cada uno de los cuales jugará 40 como visitante y otros 40 como local. Uno de los vínculos más importantes es que cada uno de los estadios ofrece 50 fechas preferidas para jugar los partidos como locales(de donde, óptimamente, deben ser elegidas 40) También debe minimizarse la distancia total recorrida por los equipos; regularmente mientras se encuentra el equipo de gira (es decir, se trata de evitar periodos de más de tres días sin jugar en ciertas fechas por la existencia de otro equipo de eventos que se llevan a acabo en los estadios (conciertos, circos, otros deportes, etc.); la distribución de juegos debe ser lo más "uniforme" posible a lo largo del año; no se deben repartir partidos con el mismo equipo en el mismo lugar y si no ha trascurrido al menos 2 semanas; las giras jugando como visitantes no deben exceder 7 partidos ó 14 días; etc.

Estas son sólo algunas de las restricciones que tiene este problema, ciertamente uno de los más complejos de su tipo. El análisis de los resultados incluyó una comparación con el calendario oficial de la NHL para la temporada 1991-1992. Si bien el calendario producido viola 4 restricciones, se entiende que este inconveniente podría ser solucionado tras una discusión con los responsables de los equipos. Debe notarse que el proceso habitual de creación del calendario siempre implica un gran número de discusiones y cambios, y suele ser una tarea que tiene varias iteraciones. No obstante, desde otro punto de vista, el calendario producido por este método es muy superior al oficial: se podría haber reducido durante la temporada un total de 78.500 millas recorridas.

6.7 NOTAS HISTÓRICAS

La palabra heurístico viene de la palabra griega *heuriskein* que significa "descubriendo nuevos métodos para resolver problemas" o "el arte de resolver problemas".

El Problema de las Ocho Reinas o su versión general de n-reinas es típico de una clase de problemas conocidos como satisfacción de restricciones o de etiquetado consistente y se usa a menudo para demostrar ideas relacionadas con búsqueda hacia atrás.

El Rompecabezas del Ocho es una versión pequeña de la versión del rompecabezas del 15 usando una tabla de 4x4. Este problema lo introdujo Sam

Loyd por primera vez en la década de 1870, y muchos investigadores de inteligencia artificial lo usado como plataforma para probar métodos heurísticos.

En 1953 el proceso físico de recocido fue modelado exitosamente por Metrópolis, Rosenbluth y Teller. En la década de los 80 Kirkpatrick, Gellat y Vecchi (1983) introdujeron el concepto de recocido en optimización combinatoria.

A finales de los 60 John Holland consciente de la importancia de la selección natural desarrolló una técnica que permitió incorporarla en un programa de computadora. Su objetivo era lograr que las computadoras aprendieran por sí mismas. A la técnica de inventó Holland se le llamó originalmente "planes reproductivos" pero se hizo popular bajo el nombre de Algoritmo Genético, tras la publicación de su libro en 1975.

CAPÍTULO 7
BÚSQUEDA TABÚ, GRASP Y REDES NEURONALES

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I,
I took the one less traveled by,
And that has made all the difference.
Robert Frost
From The Road Not Taken

7.1 INTRODUCCIÓN

Los métodos heurísticos son siempre aplicables a problemas específicos, pero tienen varios principios ampliamente aplicables para el diseño. Uno muy popular es el principio de la glotonería que guía a los métodos glotones. Cada paso sucesivo en estos métodos se toma de tal manera que minimiza el costo inmediato (o maximiza la ganancia inmediata) tal es el caso de GRASP como veremos más adelante.

Otra forma de diseñar métodos heurísticos se basa en comenzar con la solución completa del problema, y tratar de mejorar por medio de la búsqueda local en una vecindad de la solución. El proceso continúa hasta que no es posible mejorar la solución, con esta búsqueda local, de tal manera que se tiene un mínimo local. Estos métodos se conocen como métodos de búsqueda local o intercambio heurístico o de descenso, dentro de este tipo de métodos se encuentra Búsqueda Tabú.

Entre los métodos que retoman técnicas de otras áreas como el de Recocido Simulado se encuentran las Redes Neuronales mismas que se desarrollan en este capítulo.

Este capítulo se desarrolla como sigue: En la segunda sección se presenta el método de la Búsqueda Tabú, en la tercera Memoria de Corto Plazo y sus elementos, en la cuarta Oscilación Estratégica, en la quinta Reencadenamiento de Trayectorias, en la sexta Conclusiones de Búsqueda Tabú. En la séptima Grasp, en la octava Estrategias de Grasp y sus Componentes, en la novena Fase Constructiva de Grasp, en la décima Aplicaciones de Grasp. En la úndecima Redes Neuronales, el la duodécima Arquitectura de RN, en la treceava Aproximación Estadística para el Problema de Optimización, en la catorceava Conclusiones de RN y finalmente en la quinceava Notas Históricas.

7.2 BÚSQUEDA TABU

La búsqueda tabú (tabú search, TS, en inglés) es un tipo de búsqueda por entornos que según su primer definidor, Fred Glover: "guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local". Al igual que en la búsqueda local, la búsqueda tabú selecciona de modo agresivo el mejor de los movimientos posibles en cada paso. Al contrario que en la búsqueda local (que siempre se mueve al mejor de su entorno y finaliza con la llegada a un óptimo local), la búsqueda tabú permite moverse a una solución del entorno aunque no sea tan buena como la actual, de modo que se pueda escapar de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores.

La amplia gama de éxitos en aplicaciones prácticas de optimización han provocado un crecimiento acelerado de búsqueda tabú en años recientes. Se han establecido nuevos records usando TS o híbridos de TS con otras heurísticas o procedimientos algorítmicos, para encontrar mejores soluciones a problemas en programación, secuenciación, asignación de recursos, planeación de inversiones, telecomunicaciones y en muchas otras áreas.

Una muestra de las aplicaciones se da en la tabla 7.1.

Búsqueda tabú se basa en la premisa de que para poder calificar de inteligente la solución de un problema, debe incorporar memoria adaptativa y exploración sensible. El uso de memoria adaptativa contrasta con diseños "desmemoriados", tales como los que se inspiran en metáforas de la física y la biología, y con diseños de "memoria rígida", como aquellos ejemplificados por ramificación y acotamiento, y sus "primos" de IA.

El énfasis en la exploración sensible en búsqueda tabú, ya sea en búsqueda tabú, ya sea en una implementación determinística o probabilística, se deriva de la suposición de que una mala elección estratégica puede producir más información que una buena elección al azar. (En un sistema que emplea memoria, una mala elección basada en una estrategia puede dar claves útiles acerca de cómo podrían hacerse modificaciones provechosas a la estrategia. Incluso en un espacio significativamente aleatorio –los cuales afortunadamente no son lo suficientemente persistentes del mundo real- un diseño específico puede ser más apto para descubrir la impronta de la estructura, y por lo tanto para descubrir oportunidades para explotar las condiciones donde el azar no es determinante).

Secuenciación Flow-time Cell Manufacturing Procesadores Heterogéneos Scheduling Planificación de la fuerza laboral Horarios escolares	**Telecomunicaciones** Enrutamiento de llamadas Asignación de caminos Diseño de redes para servicios Planificación de descuentos a clientes Arquitectura inmune a fallos Redes ópticas síncronas
Diseño CAD Redes tolerantes a fallos Diseño de redes de transporte Planificación de espacio en arquitectura Diseño de redes de carga fija Problemas de corte irregular Distribución en planta (layout)	**Producción, Inventarios e Inversión** Fabricación flexible JIT MRP con restricciones de capacidad Selección de componentes Planificación de inventarios
Localización Asignación cuadrática Semi-asignación cuadrática Asignación Generalizada Multinivel Multicommodity	**Enrutamiento** Enrutamiento de vehículos Enrutamiento con ventanas de tiempo Enrutamiento multi-modal Enrutamiento con flota fija Agente Viajero Secuenciación de convoyes
Lógica e Inteligencia Artificial Satisfactibilidad máxima Lógica probabilística Clustering Reconocimiento / Clasificación de Patrones Integridad de los datos Entrenamiento de Redes Neuronales Diseño de Redes Neuronales	**Grafos** Partición de un grafo Coloración de un grafo Partición de clicas Problemas de clicas máximas Grafos Planares Máximos Problemas de P-Medianas
Tecnología Inversión sísmica Distribución de energía eléctrica Diseño estructural de ingeniería Volumen mínimo de elipsoides Construcción de estaciones espaciales Colocación de células de circuitos Explotación de crudo off-shore	**Optimización combinatoria en general** Programación 0-1 Optimización de carga fija Programación no lineal no convexa Redes Todo-Nada Optimización Entera Mixta

Tabla 7.1

Los elementos básicos de búsqueda tabú tienen varias características importantes que se resumen en la tabla 7.2

Memoria Adaptativa
Selectividad (incluyendo olvido estratégico) Abstracción y descomposición (a través de memoria explícita y por atributos) Tiempo: • Recencia de eventos • Frecuencia de eventos • Diferenciación entre corto y largo plazo Calidad e Impacto: • Atracción relativa de elecciones alternativas • Magnitud de cambios en relaciones de estructura o restricciones Contexto: • Interdependencia regional • Interdependencia estructural • Interdependencia secuencial
Exploración sensible Imposición estratégica de limitaciones e inducciones *(condiciones tabú y niveles de aspiración)* Enfoque concentrado en buenas regiones y buenas características de las soluciones *(procesos de intensificación)* Caracterización y exploración de nuevas regiones prometedoras *(procesos de diversificación)* Patrones de búsqueda no monótonos *(oscilación estratégica)* Integración y extensión de soluciones *(Reencadenamiento de trayectorias)*

Tabla 7.2

La base para la búsqueda tabú puede describirse como sigue:

Dada una función f(x) a optimizar en un conjunto X, búsqueda tabú empieza de la misma manera que cualquier búsqueda local, procediendo iterativamente de un punto (solución) a otro hasta satisfacer un criterio dado de terminación. Cada $x \in X$ tiene un entorno (o vecindad) asociado $N(x) \subseteq X$, y cada solución $x' \in N(x)$ se puede alcanzar desde x mediante una operación llamada movimiento.

TS rebasa la búsqueda local empleando una estrategia de modificación de N(x)a medida que la búsqueda progresa, reemplazándola por otro entorno N*(x). Un aspecto clave de búsqueda tabú es el empleo de estructuras especiales de memoria que sirven para determinar N*(x) y así organizar la manera en la cual se explora el espacio

Las soluciones que se admiten en N*(x) por parte de estas estructuras de memoria se determinan de varias formas. Una de ellas, que da a la TS su nombre identifica soluciones encontradas sobre un horizonte especificado (e implícitamente algunas soluciones identificadas con ellas), y les prohíbe pertenecer a N*(x) clasificándolas como tabú. (En principio la terminología tabú implica un tipo de inhibición que incorpora una connotación "cultural" –esto es, algo que está sujeto a la influencia de la historia y el contexto, y que sin embargo puede superarse bajo condiciones apropiadas-).

El proceso mediante el cual las soluciones adquieren un estatus tabú tiene varias facetas, diseñadas para promover un examen agresivo y guiado de nuevos puntos. Una manera útil de visualizar e implementar este proceso es el reemplazo de la evaluación original de soluciones mediante evaluaciones tabú, las cuales introducen penalizaciones para desalentar en forma significativa la elección de soluciones tabú (es decir, aquellas que serán preferentemente excluidas del entorno N*(x), de acuerdo a su dependencia en los elementos que conforman el estatus tabú).

Además, las evaluaciones tabú incluyen también periódicamente incentivos para estimular la elección de otro tipo de soluciones, como resultado de niveles de aspiración e influencias a largo plazo.

Las siguientes subsecciones describen como TS saca provecho de la memoria (y por ende del proceso de aprendizaje) para llevar a cabo estas funciones.

La memoria usada en TS puede ser explícita o basada en atributos, aunque ambas modalidades no son excluyentes. La memoria explícita conserva soluciones completas, y consiste típicamente en una elite de soluciones visitadas durante la búsqueda (o entornos altamente atractivos pero inexplorados para tales soluciones). Estas soluciones especiales se introducen estratégicamente para ampliar N*(x), y así presentar opciones útiles que no se encuentran en N(x), y así presentar opciones útiles que no se encuentran en N (x).

La memoria en TS se diseña también para producir un efecto más sutil en la búsqueda a través del uso de una memoria basada en atributos, la cual guarda información sobre atributos de las soluciones que cambian al moverse de una solución a otra. Por ejemplo, en un contexto de grafos o redes, los atributos pueden consistir de nodos o arcos que se añaden, se suprimen o se sustituyen por los movimientos ejecutados. En formulaciones de problemas más abstractos, los atributos pueden corresponder a valores de variables o funciones. A veces los atributos también pueden combinarse estratégicamente para crear otros atributos, mediante procedimientos de hashing o por segmentaciones relacionadas con IA o por métodos de "construcción de vocabulario".

7.3 MEMORIA DE CORTO PLAZO Y SUS ELEMENTOS

Una distinción importante en TS resulta al distinguir entre memoria de corto plazo y memoria de largo plazo. Cada tipo de memoria está acompañada de sus propias estrategias especiales. La memoria de corto plazo más comúnmente usada, lleva la cuenta de los atributos de solución que han sido cambiados en el pasado reciente, y es llamada memoria basada en recenciantes (en hechos recientes). Para explotar esta memoria, los atributos seleccionados que se presentan en soluciones recientemente visitadas son designados como "tabú-activos", y las soluciones que contienen elementos tabú-activos, combinaciones particulares de estos atributos, son las que se convierten en tabú. Esto evita que algunas soluciones del pasado son las que se convierten en tabú. Esto evita que algunas soluciones del pasado reciente pertenezcan a N*(x) y por lo tanto de que sean revisitadas. También se evita que otras soluciones que comparten tabú-activos se vuelvan a visitar. El uso de evaluaciones tabú, que asignan penalizaciones grandes a conjuntos apropiados de atributos tabú-activos, permite que el estatus tabú varíe por grados.

MANEJO DE MEMORIA BASADA EN LA RECENCIA

El proceso de maneja creando una o más listas tabú, las cuales registran los atributos tabú-activo y explícita e implícitamente identifican su estatus actual Se denomina tenencia tabú a la duración que un atributo permanece tabú-activo (medido en número de intenciones). La tenencia tabú puede variar para diferentes tipos o combinaciones de atributos, y además puede variar para diferentes intervalos de tiempo o etapas de la búsqueda. Esta tenencia variable permite la creación de diferentes tipos o etapas de la búsqueda. Esta tenencia variable permite la creación de diferentes tipos de ajuste entre estrategias de corto y largo plazo. Esto produce también una forma dinámica y robusta de búsqueda.

Ejemplos de estructuras de memoria basada en hechos recientes y reglas asociadas para la implementación de éstas se dan más adelante.

NIVELES DE ASPIRACIÓN

El criterio de aspiración introduce un elemento importante de flexibilidad en la búsqueda tabú. El estatus tabú de una solución (o un movimiento) puede ser ignorado si ciertas condiciones se cumplen, en la forma de niveles de aspiración. En efecto estos niveles de aspiración dan umbrales de atracción, los cuales controlan el hecho de que las aspiraciones pueden ser consideradas admisibles a pesar de estar clasificadas como tabú. Es obvio que cualquier solución que sea mejor que cualquiera de las encontradas anteriormente merece ser considerada admisible, incluso aunque para alcanzarla debamos utilizar un movimiento prohibido. Criterios similares para la calidad de las soluciones producen criterios de aspiración para subconjuntos de soluciones que

pertenecen a regiones comunes o que comparten características especificadas (tales como un valor funcional particular o un nivel de imposibilidad). Más adelante se darán ejemplos adicionales de criterios de aspiración.

ESTRATEGIAS PARA LA LISTA DE CANDIDATOS

El carácter agresivo de TS se ve reforzado buscando el mejor movimiento disponible que pueda ser determinado con una cantidad apropiada de esfuerzo. Debe tomarse en cuenta que el significado de "mejor" no está limitado solamente a la evaluación de la función objetivo. (Como anteriormente se señaló, las evaluaciones tabú están afectadas por penalizaciones e incentivos, que son determinados por la historia de la búsqueda). Las estrategias para la lista de candidatos se usan para restringir el número de soluciones examinadas en una iteración dada, para los casos en los que N* (x) es grande o la evaluación de sus elementos es costosa.

Dada la importancia que TS da a la selección de elementos para el proceso de búsqueda, es critico contar con reglas eficientes para la generación y evaluación de buenos candidatos. Aún en casos donde las estrategias para la lista de candidatos no se usan explícitamente, las estructuras de memoria que den actualizaciones eficientes de evaluaciones del movimiento de una iteración a otra, y que reduzcan el esfuerzo de encontrar mejores o casi mejores movimientos, son parte integral de las implementaciones TS. La actualización inteligente puede reducir apreciablemente los tiempos de solución, y la inclusión de estrategias para la lista de candidatos, para los problemas grandes, puede aumentar significativamente los beneficios resultantes. Se indicarán tipos útiles de estrategias para la lista de candidatos en la Sección 4. La operación de estos elementos de corto plazo está ilustrada en la figura 7.1

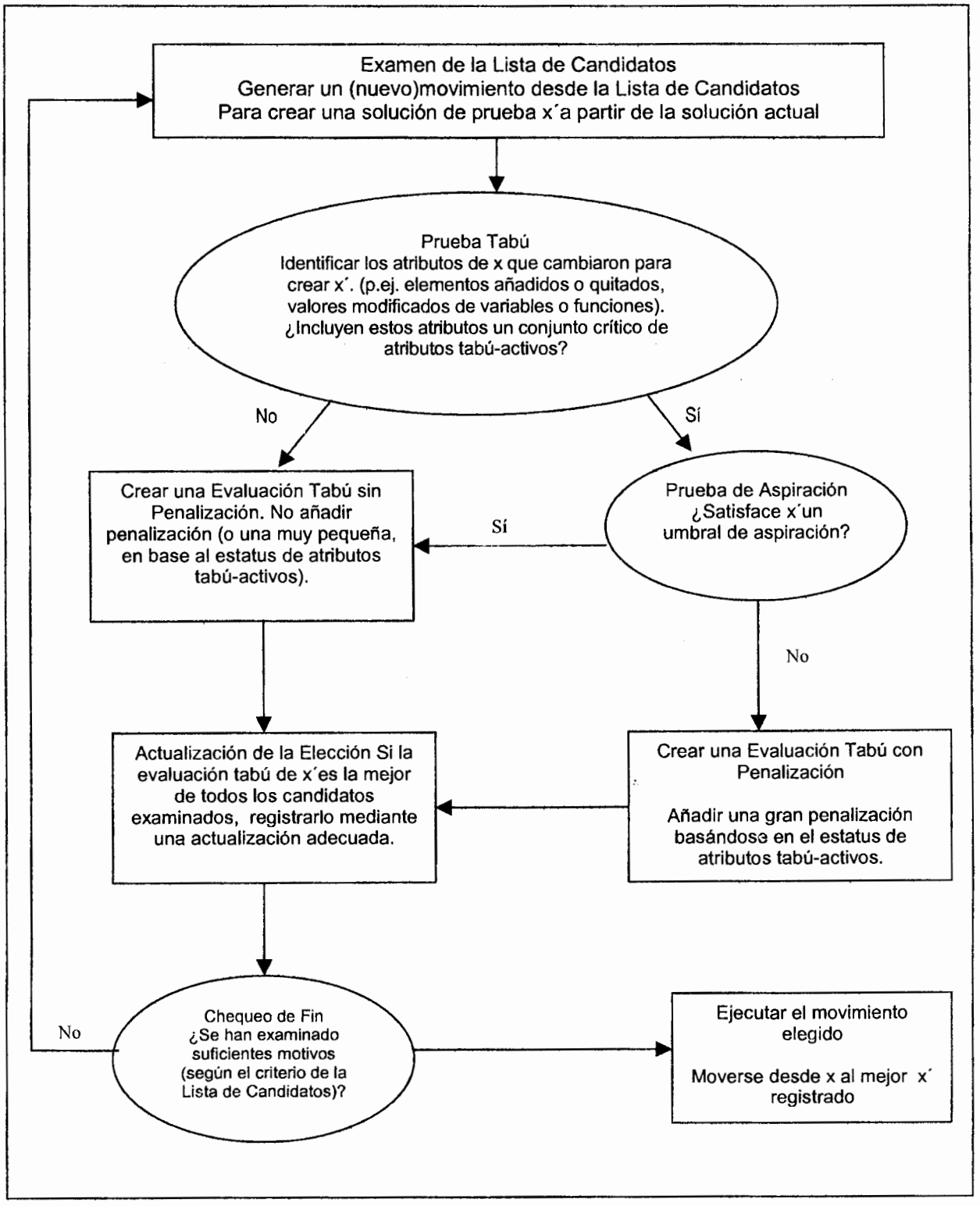


Figura 7.1. Esquema del proceso tabú de evaluación (memoria a corto plazo)

La representación de las penalizaciones en la figura 7.1.(ya sea como "grandes" o "muy pequeñas") refleja un efecto de umbral: el estatus tabú o bien produce una evaluación sumamente deteriorada o bien sirve para romper empates entre las soluciones cuyas evaluaciones son las más altas. Tal efecto puede ser – por supuesto – modulado para desplazar las evaluaciones hacia niveles intermedios entre estos extremos. Si todos los movimientos actualmente disponibles conducen a soluciones que son tabú (con evaluaciones que normalmente las hubieran excluido de ser seleccionadas), las penalizaciones harán que se escoja una solución "menos tabú".

Se puede observar que la secuencia de la Prueba Tabú y la Prueba de Aspiración en la figura 7.1. puede ser intercambiada (esto es, empleando la prueba tabú sólo si el umbral de aspiración no se satisface). Además, la evaluación tabú se puede modificar creando incentivos basados en el nivel de aspiración, de la misma forma en que se modifica al crear penalizaciones basadas en el estatus tabú. En este sentido, las condiciones de aspiración y las condiciones tabú pueden ser consideradas como "imágenes especulares" entre sí.

La variante de TS llamada búsqueda tabú probabilística mantiene un diseño similar, con una componente de corto plazo que puede ser representada por la misma figura. Este enfoque además mantiene un registro de las evaluaciones tabú generadas durante el proceso de selección de un movimiento. Basándose en este registro, el movimiento es seleccionado probabilísticamente del conjunto de aquellos evaluados (o de algún subconjunto de los mejores miembros de este conjunto), evaluando los movimientos de tal manera que aquellos con valores más altos resulten especialmente favorecidos. Discusiones más extensas acerca de la búsqueda tabú probabilística pueden ser encontradas en Glover [1989, 1993], Soriano y Gendreau [1993] y Crainic et al. [1993].

El estatus tabú es frecuentemente permitido para que sirva como un umbral todo-o – nada (sin referencia explícita a las penalizaciones o incentivos) mediante la exclusión directa de selección de opciones tabú (sujetas al resultado de las pruebas de aspiración). Bien las evaluaciones modificadas sean explícitamente usadas o no, el movimiento seleccionado puede no ser el que tenga el mejor valor para la función objetivo, y consecuentemente la solución con el mejor valor para la función objetivo encontrada durante toda la historia de la búsqueda será registrada separadamente. (Una forma típica de criterio de aspiración indica que tal solución será escogida como la siguiente por visitar).

MEMORIA DE LARGO PLAZO.

En algunas aplicaciones, los componentes de la memoria TS de corto plazo son suficientes para producir soluciones de muy alta calidad. Sin embargo, en general TS se vuelve significativamente más potente incluyendo memoria de largo plazo y sus estrategias asociadas.

Tipos especiales de memoria basada en frecuencia son fundamentales en consideraciones de largo plazo. Estas operan introduciendo penalizaciones e incentivos determinados por el rango relativo de tiempo durante el que los atributos han permanecido a soluciones visitadas durante la búsqueda, permitiendo diferenciación por regiones. Las frecuencias de transición mantienen un registro de con qué frecuencia cambian los atributos, mientras que las frecuencias de residencia mantienen el registro de las duraciones relativas de los atributos en las soluciones generadas. (Este tipo de memorias son acompañadas algunas veces por formas extendidas de memoria basada en recencia).

Quizás sorprendentemente, el uso de memoria de largo plazo no requiere secuencias de larga duración antes de que sus beneficios se hagan visibles. Frecuentemente, sus mejoras comienzan a manifestarse en un lapso de tiempo relativamente corto, y puede permitir que los esfuerzos para llegar a la solución finalicen un poco antes que de otra manera posible, debido a que se encuentran soluciones de muy alta calidad dentro de un rango corto de tiempo.

Los métodos más rápidos para la secuenciación de tareas en job shops y flow shops, por ejemplo, se basan en la inclusión de memoria de largo plazo TS. (Esto incluye la memoria explícita de un tipo posteriormente descrito.) Por otro lado, también es cierto que la oportunidad de encontrar soluciones aún mejores conforme el tiempo crece – en el caso de que una solución óptima no haya sido ya encontrada- se mejora al usar memoria de largo plazo en adición a la memoria de corto plazo.

ESTRATEGIAS DE INTENSIFICACIÓN

Dos componentes altamente importantes de largo plazo de búsqueda tabú son las estrategias de intensificación y las estrategias de diversificación. Las estrategias de "intensificación" están basadas en la modificación de reglas de lección de tal manera que se favorezcan combinaciones de movimientos y características de solución que históricamente hayan sido buenas. Pueden además iniciar un regreso hacia regiones atractivas para buscar en ellas más extensamente. Un ejemplo simple de este segundo tipo de estrategia de intensificación se muestra en la figura 7.2, en donde la estrategia para seleccionar soluciones elite se muestra en itálicas para remarcar su importancia.

Dos variantes han demostrado tener bastante éxito. La primera, debida a Voss [1993], introduce una medida de la diversificación para asegurar que las soluciones registradas difieran una de otra en un grado deseado, y borra toda memoria de corto plazo antes de reanudar el proceso desde la mejor de las soluciones registradas. La otra variante, debida a Nowicki y Smutniki [1993], mantiene una lista secuencial de longitud limitada que añade al final una nueva solución sólo si es mejor que cualquier otra previamente vista.

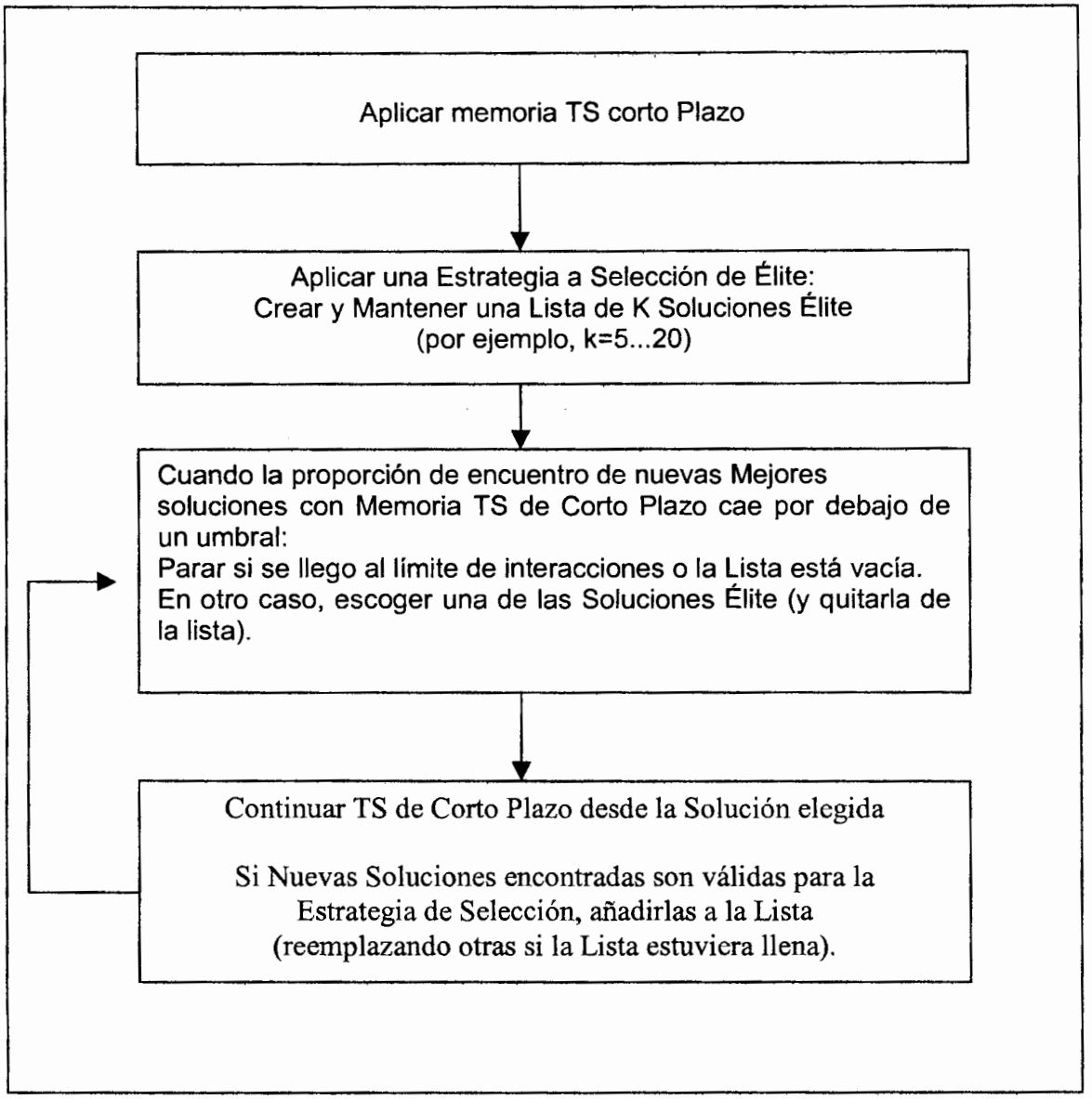


Figura 7.2. Enfoque simple de intensificación en Búsqueda Tabú.

Siempre se escoge el último miembro actual de la lista (y se suprime) como base para reanudar la búsqueda. Sin embargo, también se guarda la memoria de corto plazo TS que acompañó a esta solución, y el primer movimiento inhibe además el movimiento previamente tomado de esta solución, con lo que se inicia un nuevo camino de solución. Algunas veces este enfoque de recuperar soluciones elite seleccionadas es llamado backtracking (desandar). Sin embargo, no está relacionado al backtracking de los métodos de búsqueda en árboles y simplemente consiste en crear una cola limitada de prioridad.

Esta segunda variante se relaciona con la estrategia de reanudar la búsqueda desde entornos no visitados, previamente generados (Glover [1990]). Tal

estrategia mantiene el registro de la calidad de estos entornos para seleccionar un conjunto elite, y limita la atención a tipos específicos de soluciones, como son los entornos de óptimos locales o entornos de soluciones visitadas en pasos inmediatamente anteriores a alcanzar tales óptimos locales. Este tipo de estrategia de "entorno no visitado" se ha examinado poco. Cabe notar, sin embargo, que las dos variantes previamente indicadas han producido soluciones de una notoria alta calidad.

Otro tipo de enfoque de intensificación es la "intensificación por descomposición", donde se pueden imponer restricciones a partes del problema o la estructura de solución para generar una forma de descomposición que permita un enfoque más concentrado en otras partes de la estructura.

Un ejemplo clásico se produce en el TSP, donde los bordes que pertenezcan a la intersección de recorridos elite pueden ser "atrapados" en la solución, para así poder enfocarse en la manipulación de otras partes del recorrido. El uso de intersecciones puede ser considerado como una instancia extrema de una estrategia más general para explotar la información de frecuencia, por un proceso que busca identificar y restringir los valores de variables consistentes y fuertemente determinadas.

La intensificación por descomposición también incluye otro tipo de consideraciones estratégicas, como basar la descomposición no sólo en indicadores de fuerza y consistencia, sino también en las oportunidades de elementos particulares para interactuar productivamente.

EJEMPLO 7.1
En el contexto del problema del viajero, una descomposición se puede basar en identificar subcadenas de un recorrido elite, donde dos o más subcadenas pueden ser asignadas a un conjunto común si contienen nodos que estén "fuertemente atraídos" para ser enlazados con nodos de otras subcadenas en el conjunto. Un conjunto de subcadenas de bordes disjuntos puede ser considerada por un proceso de intensificación que opere en paralelo en cada conjunto, sujeto a la restricción de que la identidad de los puntos finales de la subcadenas no sea alterada. Como resultado de la descomposición, los mejores nuevos conjuntos de subcadenas pueden ser reembolsados para crear un nuevo recorrido. Tal proceso puede ser aplicado a múltiples alternativas de descomposición en formas más generales de intensificación por descomposición.

ESTRATEGIAS DE DIVERSIFICACIÓN

Las estrategias de diversificación en TS, como su nombre sugiere, están diseñadas para conducir la búsqueda hacia nuevas regiones. Con frecuencia están basadas en modificar las reglas de elección para llevar a la solución atributos que no hayan sido usados frecuentemente. Alternativamente, pueden

introducir dichos atributos al reiniciar parcial o completamente el proceso de solución. Los mismos tipos de memorias previamente descritos son útiles como fundamente para tales procedimientos, aunque estas memorias sean mantenidas a través de subconjuntos (generalmente más largos) de soluciones, que aquellos mantenidos por estrategias de diversificación.

Un enfoque simple de diversificación que mantiene una memoria basada en frecuencia sobre todas las soluciones previamente generadas, y que ha probado tener mucho éxito para problemas de scheduling, es mostrado en la figura 7.3.

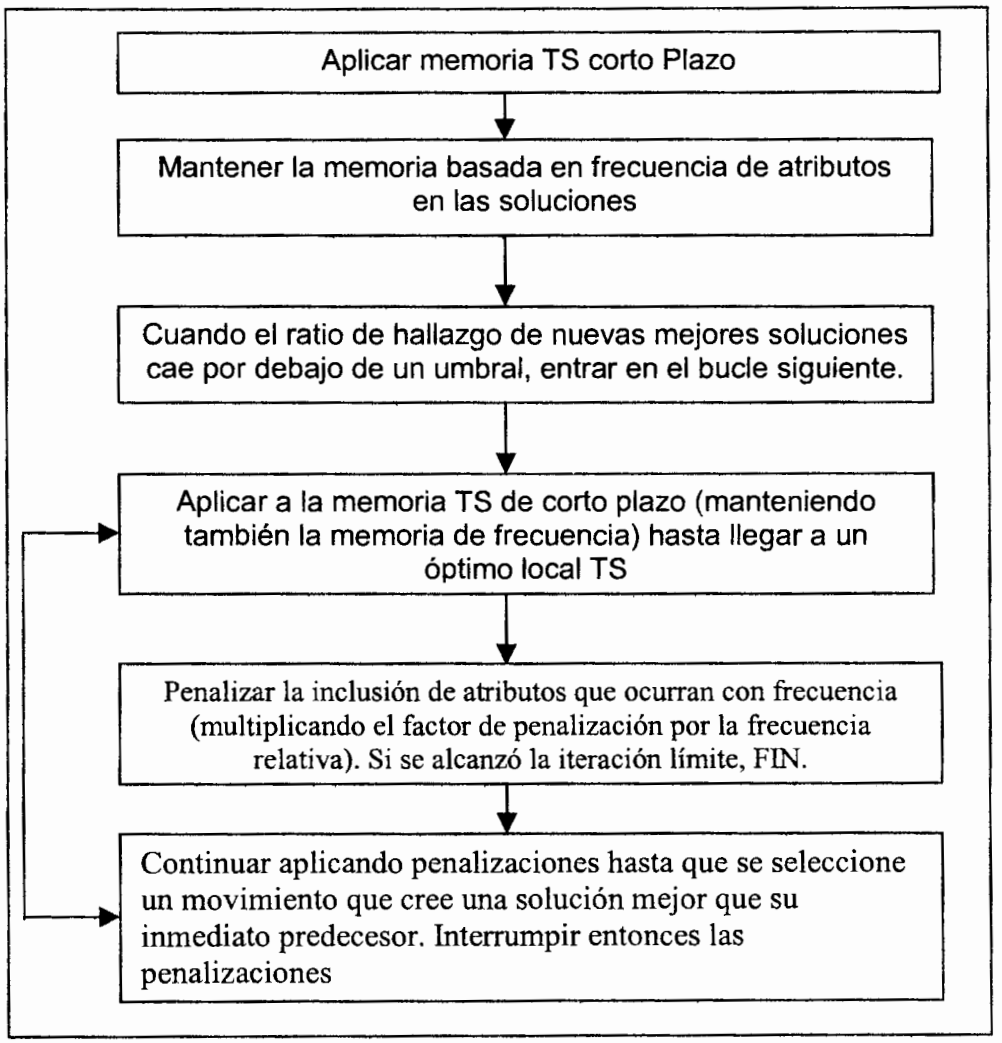


Figura 7.3. Enfoque simple de diversificación en Búsqueda Tabú.

Mejoras significativas en la aplicación de memoria de corto plazo de TS han sido logradas mediante el procedimiento de la figura 7.3.

Los óptimos locales de TS alcanzados por este enfoque, y usados como base para lanzar una secuencia de pasos con el objeto de diversificar, pueden naturalmente diferir de los verdaderos óptimos locales ya que las reglas de selección de búsqueda tabú pueden excluir algunos movimientos mejoradores.

El éxito de este enfoque sugiere el mérito de incorporar una variante de TS que siempre continúe a una verdadero óptimo local una vez que un movimiento mejorador se convierta en una elección aceptable (basado en un criterio de aspiración que es activado sólo después de ejecutar un movimiento mejorador). En este enfoque, mientras existan movimientos que adicionalmente mejore, el criterio de aspiración permitirá que uno de ellos sea seleccionado mediante una regla de evaluación tabú que penalizará las opciones basándose a su estatus tabú (restringiendo la atención al conjunto de mejora).

Una vez que un verdadero óptimo local es alcanzado, el criterio de aspiración especial se interrumpe hasta que un nuevo movimiento mejorador se selecciona usando reglas de búsqueda tabú estándar. Este enfoque incorpora una instancia de la aspiración por búsqueda de dirección, y puede ser refinado de formar útil tomando en cuenta "esferas de influencia".

Un área fértil de investigación es el modo en el cual las memorias basadas en frecuencia son usadas para implementar las estrategias de intensificación y diversificación (aparte de la forma de definir estas memorias sobre diferentes subconjuntos). En la figura 7.4. se ilustran dos patrones generales distintos para explotar este tipo de memorias.

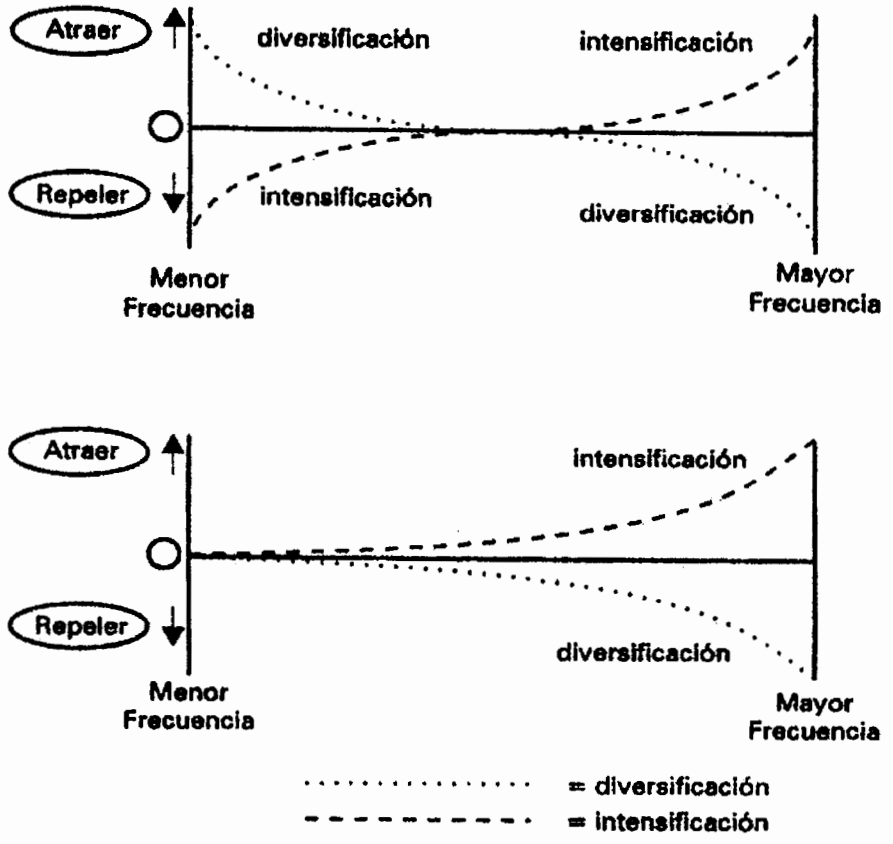


Figura 7.4. intensificación y diversificación: grado de atracción/repulsión según la frecuencia

De las variaciones naturales en estos patrones puede ser inferida una variedad de alternativas adicionales. Las estrategias de diversificación que crean reinicios parciales o completos (de la búsqueda) son importantes para los problemas y estructuras de entorno donde una trayectoria de solución puede ser aislada de entre alternativas nuevas y valiosas a menos que se introduzca un cambio radical

Las estrategias de diversificación pueden utilizar también una forma de largo plazo de memoria basada en recencia, cuyos resultados incrementan la tenencia tabú de los atributos de solución. Una versión simple de este enfoque que ha producido buenos resultados se muestra en la figura 7.5.

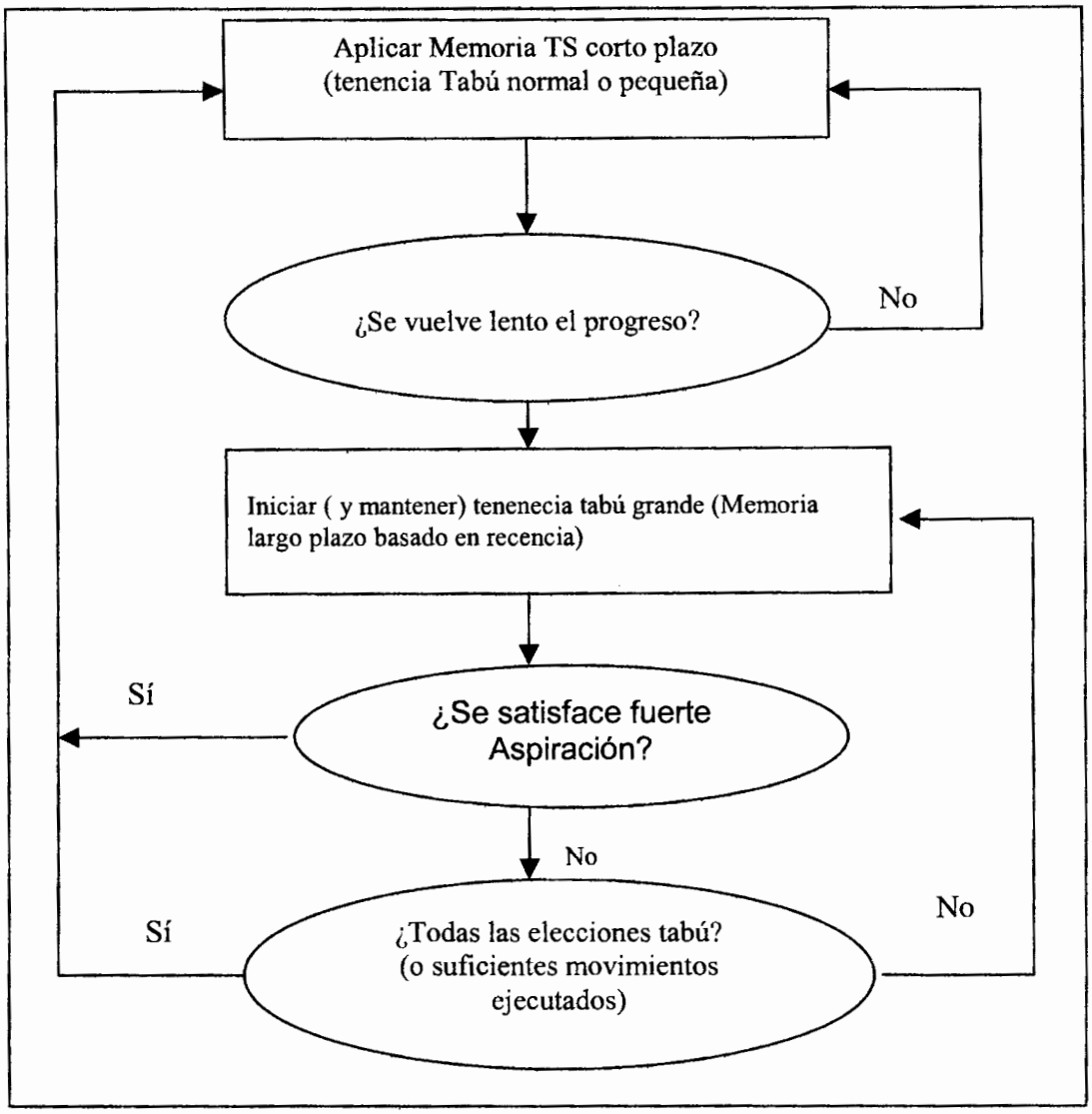


Figura 7.5 Diversificación usando memoria de largo plazo basada en recencia

La determinación de formas efectivas de equilibrar los aspectos de intensificación y diversificación representan un área de investigación prometedora. La meta desde la perspectiva de TS es diseñar patrones para compartir comunicación e información entre subconjuntos de procesadores con el fin de lograr los mejores compromisos entre funciones de intensificación y diversificación.

7.4 OSCILACIÓN ESTRATÉGICA

La oscilación estratégica está estrechamente ligada a los orígenes de búsqueda tabú, y proporciona un medio para lograr una interacción muy efectiva entre intensificación y diversificación ya sea a mediano o a largo plazo. El enfoque opera orientando los movimientos con relación a un cierto nivel crítico, identificándolo ya sea por una etapa de construcción o un intervalo dado de valores para una función.

Tal nivel crítico representa a menudo un punto donde el método se detendría normalmente. Sin embargo, en vez de detenerse al alcanzar este nivel, las reglas para elegir los movimientos se modifican, para permitir que la región definida por el nivel crítico sea traspasada. El enfoque, entonces, permite llegar hasta cierta profundidad previamente especificada más allá el nivel crítico y traspasarlo, sólo que esta vez se hará en dirección opuesta a la que se hizo anteriormente, y el método se dirige a un nuevo punto de retorno.

El proceso de aproximarse y traspasar una y otra vez el nivel crítico en direcciones diferentes crea un comportamiento oscilatorio, el cual da al método su nombre. Se puede establecer un control sobre este comportamiento generando evaluaciones y reglas modificadas de movimiento, dependiendo de la región explorada y la dirección de la búsqueda. La posibilidad de recorrer nuevamente una trayectoria se evita mediante los mecanismos usuales de búsqueda tabú.

EJEMPLO 7.2

Un ejemplo muy sencillo de este enfoque se presenta con el problema multidimensional de la mochila, donde los valores de las variables binarias se cambian de 0 a 1 hasta alcanzar la frontera de factibilidad. El método continúa entonces hacia la región infactible usando el mismo tipo de cambios, pero con un evaluador modificado. Después de un número prefijado de pasos, la dirección se revierte eligiendo movimientos que cambian las variables de 1 a 0. Los criterios de evaluación para conducir el proceso hacia la mejoría varia de acuerdo a si el movimiento ocurre dentro o fuera de la región factible (y si se dirige hacia la frontera o se aleja de ella), acompañado de restricciones asociadas sobre cambios admisibles a los valores de las variables. Una implementación de tal enfoque por Frevill y Plateau [1986, 1992] ha generado soluciones de muy alta calidad para problemas multidimensionales de la mochila.

Un tipo de aplicación un poco diferente se presenta en problemas de teoría de grafos donde el nivel crítico representa una forma de estructura de grafo deseada, capaz de ser generada por adiciones (inserciones) progresivas de elementos básicos tales como vértices, aristas o subgrafos. Un tipo de enfoque de oscilación estratégica para entre problema consiste en un proceso constructivo de introducción de elementos hasta alcanzar el nivel crítico, y después introducir elementos adicionales para cruzar la frontera definida por el nivel crítico. La solución obtenida puede cambiar su estructura una vez que esta frontera haya sido cruzada (sería el caso en que un bosque se transforma en grafo que contiene ciclos), y así puede ser que se requiera un entorno diferente, produciendo reglas modificadas para elegir movimientos. Las reglas nuevamente cambian para proceder en la dirección opuesta, eliminando elementos hasta que se recupere nuevamente la estructura que define el nivel crítico.

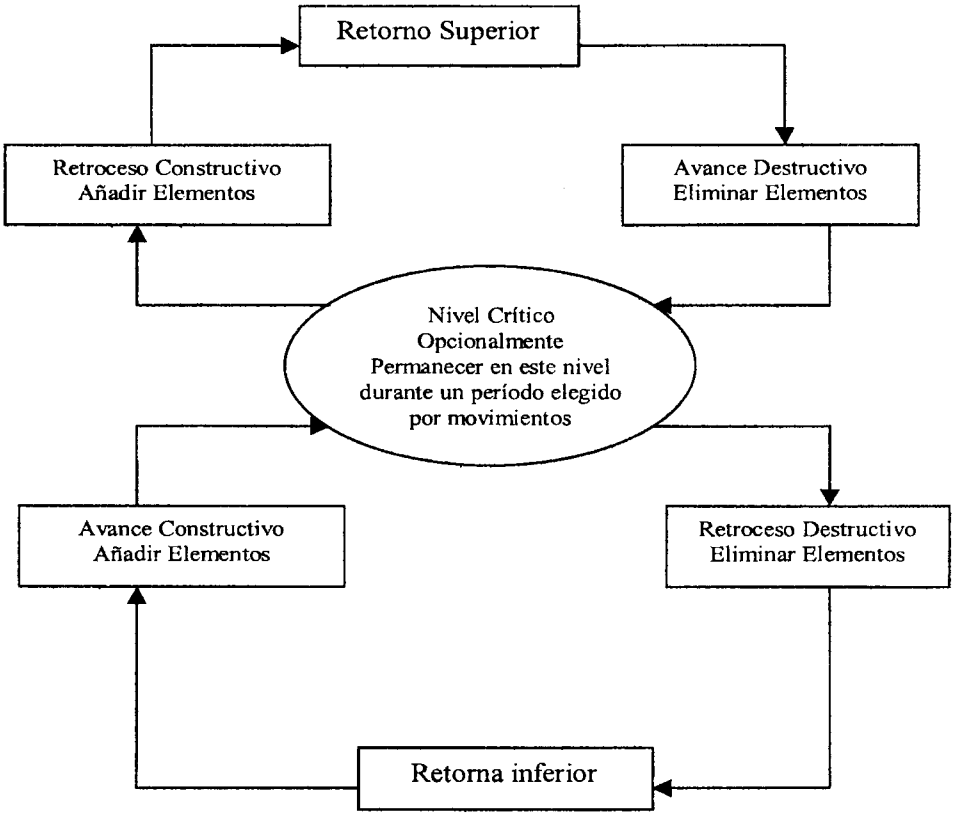
Tales cambios en las reglas con características típicas de la oscilación estratégica, y proporcionan una vitalidad heurística muy rica. La aplicación de distintas reglas puede acompañarse de cruces de un lado a otro de la frontera a diferentes profundidades en lados diferentes. Una opción es acercarse a, y alejarse de, la frontera, permaneciendo en el mismo lado, sin cruzarla (es decir, eligiendo un cruce de "profundidad cero").

Ambos ejemplos constituyen un tipo de oscilación estratégica constructivo/destructivo, donde los pasos constructivos "añaden" elementos (o dan a las variables el valor 1) y los pasos destructivos "eliminan" elementos (o dan a las variables el valor 0). En estos enfoques a menudo e importante emplear algún tipo adicional de búsqueda en las proximidades del nivel crítico, y especialmente emplear algún tiempo en el nivel crítico, como preludio de oscilaciones mayores que se sumerjan a profundidades mayores.

Alternativamente, si se permite un mayor esfuerzo para evaluar y ejecutar cada movimiento, el método puede usar "movimientos de intercambio" (interpretados de la manera más amplia) para permanecer en el nivel crítico durante períodos más prolongados. Una opción muy sencilla, por ejemplo, es usar tales intercambios para alcanzar un óptimo local cada vez que el nivel crítico se alcanza. Una estrategia de aplicación de intercambios similares a niveles adicionales se sugiere por un principio de optimalidad próxima, el cual establece que buenas construcciones a un nivel, muy probablemente estarán cercanas a buenas constucciones a otro nivel. Una versión simple de una forma constructiva/destructiva de oscilación estratégica se ilustra en la figura 7.6.

Como se puede observar en la tabla que acompaña la figura 7.6, la oscilación puede operar también al incrementar o disminuir las cotas de cierta función g(x). Tal enfoque ha servido como base para algunas aplicaciones muy efectivas, donde g(x) ha representado objetos tales como asignación de fuerza de trabajo, valores de funciones objetivo, y niveles de factibilidad/infactibilidad, para llevar la búsqueda a probar a varios niveles de las regiones asociadas.

Cuando los niveles se refieren al grado de factibilidad o infactibilidad, g(x) es una función con valores vectoriales asociada con un conjunto de restricciones del problema (que pueden expresarse, pro ejemplo, como $g(x) \leq b$). En este caso, controlar la búsqueda mediante acotamientos de g(x) puede verse como si se estuviera manipulando una parametrización del conjunto elegido de restricciones. Una alternativa que a menudo se prefiere es hacer g(x) una función de penalización Lagrangeana o una restricción subrogada, para así evitar funciones de valores vectoriales y permitir compromisos entre distintos grados de violación de las diferentes restricciones.



Tipo movimiento	Ejemplo de Alternativas	
Constructivo	$x_j = 1$	Incrementar límites (L,U) de g(x)*
Destructivo	$x_j = 0$	Disminuir límites (L,U) d g(x)
Interactivo	$x_j = 1, x_q = 0$	Mantener límites de g(x)

*g(x) es función escalar o vectorial de la solución x

Fig. 7.6 Oscilación estratégica usando movimentos contructivos/destructivos

7.5 REENCADENAMIENTO DE TRAYECTORIAS

El reencadenamiento de trayectorias proporciona una integración muy útil de las estrategias de intensificación y diversificación. Este enfoque se basa en explorar trayectorias que "conectan" soluciones elite para genera nuevas soluciones, empezando desde una de estas soluciones llamada solución inicial, y generando una trayectoria en el espacio de entornos que conduce hacia otras soluciones, llamadas soluciones guía. Esto se logra seleccionando movimientos que introducen atributos contenidos en las soluciones guía.

En enfoque puede verse como una instancia extrema (altamente enfocada) de una estrategia que busca incorporar atributos de soluciones de muy alta calidad, creando inducciones que favorezcan estos atributos en los movimientos seleccionados. Sin embargo, en vez de utilizar una inducción que simplemente apoye la inclusión de tales atributos, el enfoque de reencadenamietno de trayectorias subordinada todas las demás consideraciones a la meta de elegir movimientos que introduzcan los atributos, el enfoque de reencadenamiento de trayectorias subordina todas las demás consideraciones a la meta de elegir movimientos que introduzcan los atributos de las soluciones guía, con el fin de crear una "buena composición de atributos" en la solución actual. En cada paso de la composición se determina eligiendo el mejor movimiento, mediante los criterios usuales de elección, del conjunto restringido de movimientos que incorporan un máximo número (o un valor con peso máximo) de los atributos de las soluciones guía.

Específicamente, una vez identificada una colección de una o más soluciones elite para guiar la trayectoria a partir de una solución dada, los atributos de estas soluciones guía reciben pesos "preemptivos" como inductores para ser seleccionadas. Los atributos que se presentan en un número mayor de las soluciones guía reciben mayores pesos, permitiendo un sesgo que haga énfasis creciente en soluciones de calidad más alta o que posean características especiales. Generalmente, no es necesario que un atributo aparezca en una solución guía para alcanzar un estatus favorecedor. En algunos contextos, ciertos atributos pueden compartir grados de similaridad, en este caso puede ser útil visualizar un vector solución "emitiendo votos" para favorecer o disuadir atributos particulares. Únicamente a las formas más fuertes de criterios de aspiración se les permite ignorar este tipo de regla de elección, de manera que la trayectoria no se desvíe a menos que una solución vecina sea mejor que cualquiera de las soluciones guía (y que la solución inicial), la regla de guía se restablece después de que la desviación sigue su curso.

En una colección dada de soluciones elite, el papel de las soluciones iniciales y las soluciones guía pueden alternarse. Esto es, puede generarse simultáneamente un conjunto de soluciones actuales, desarrollando trayectorias diferentes, y permitiendo que una solución inicial sea reemplazada (como una solución guía para otras) cada vez que su solución actual asociada satisfaga un criterio de aspiración suficientemente fuerte. Debido a que sus papeles pueden

ser intercambiados, las soluciones iniciales y guías reciben el nombre común de soluciones referenciales.

Una forma idealizada de tal proceso se muestra en la figura 4.7. La colección seleccionada de soluciones referenciales consiste en los tres miembros, A, B y C. Las trayectorias se generan permitiendo que cada una actúe como solución inicial y permitiendo que una o ambas soluciones restantes operen como soluciones guía. Las soluciones intermedias que se encuentran a lo largo de las trayectorias no se muestran. La representación de las trayectorias como líneas rectas es por supuesto una simplificación, ya que al elegir entre movimientos disponibles en el entorno actual, se producirá en general una trayectoria considerablemente más compleja.

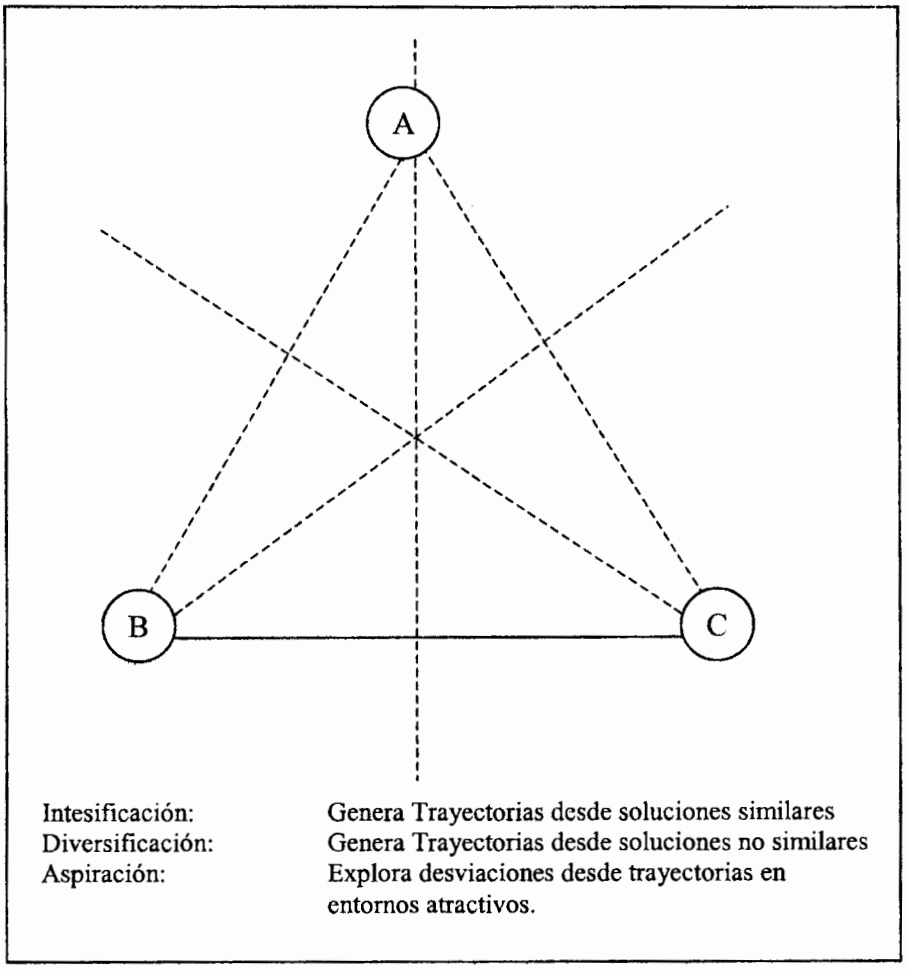


Figura 7.7. Reencadenamiento de trayectorias en el espacio de entornos.

Como indica la figura 7.7, se permite al menos una continuación de trayectoria más allá de cada solución inicial/guía. Tal continuación puede llevarse a cabo penalizando la inclusión de atributos desechados durante una trayectoria, incluyendo atributos de soluciones guía que se pueden desechar en forma forzada para poder continuar una trayectoria. (Se puede repeler también una

solución inicial de las soluciones guía penalizando la inclusión de sus atributos desde fuera.) Las variantes probabilísticas de TS operan en el contexto de reencadenamiento de trayectorias de la misma forma que lo hacen en otros, sustituyendo evaluaciones para reglas determinísticas por probabilidades de selección, sesgándolas fuertemente en favor de las evaluaciones más altas.

En reencadenamiento de trayectorias, las regiones prometedoras pueden explorarse más intensamente modificando los pesos asignados a los atributos de las soluciones guía, y alterando el sesgo asociado con la calidad de la solución y con características de la solución seleccionada.

La figura 7.8 muestra el tipo de variación que puede resultar, donde el punto X representa una solución inicial y los puntos A, B, y C representan soluciones guía. Variaciones de este tipo dentro de un dominio prometedor están motivadas por el principio de optimalidad próxima que ya fue discutido en conexión con la oscilación estratégica. Para elecciones apropiadas de los puntos de referencia (y entornos para generar trayectorias desde ellos), este principio sugiere que es probable encontrar puntos elite adicionales en las regiones recorridas por las trayectorias, al iniciar nuevas búsquedas desde puntos de alta calidad en estas trayectorias han hallado evidencias de que los espacios de soluciones combinatorias poseen muy a menudo topologías que pueden explotarse fructíferamente mediante tal enfoque.

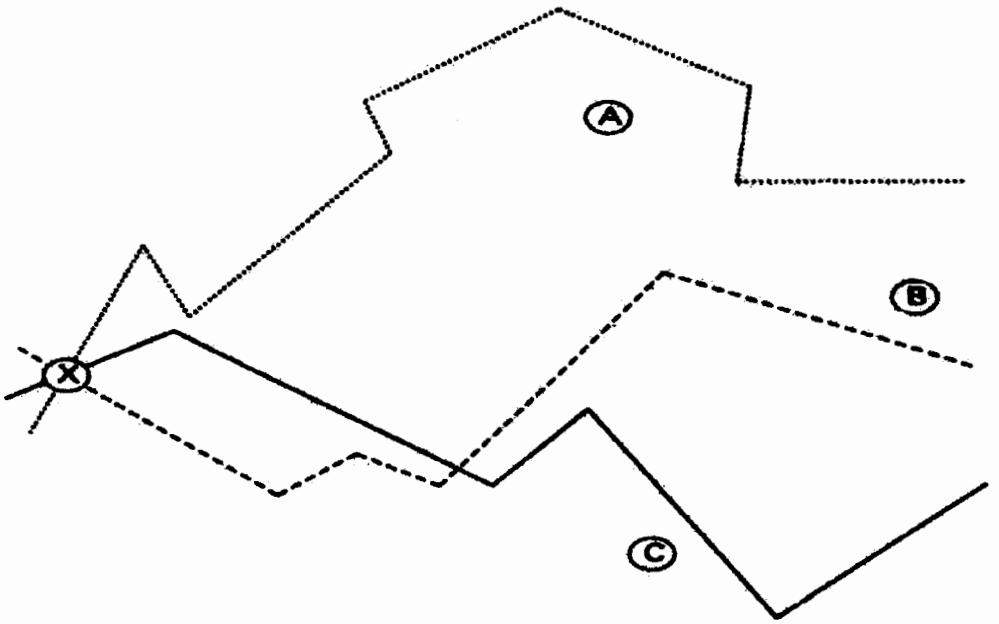


Figura 7.8 Reencadenamiento de Trayectorias

7.6 CONCLUSIONES

Los éxitos de búsqueda tabú en la práctica han promovido la investigación hacia formas de explotar con mayor intensidad sus ideas subyacentes. Al mismo tiempo, aún falta por explorar muchas facetas de estas ideas. Temas como la

identificación de las mejores combinaciones de memoria de corto y largo plazo, y los mejores equilibrios de estrategias de intensificación y diversificación, tienen aún muchas facetas por examinar, y sin duda algunas de ellas abrigan descubrimientos importantes para desarrollar métodos de solución más poderosos en el futuro.

Existen contrastes evidentes entre las perspectivas de TS y los puntos de vista favorecidos actualmente por las comunidades de inteligencia artificial y redes neuronales, particularmente en lo que concierne al papel de la memoria en los procedimientos de búsqueda. Sin embargo, esto puntos de vista también se complementan entre sí, lo cual brinda la posibilidad de crear sistemas que integren sus preocupaciones fundamentales. En este territorio ya se ven venir avances, con la creación de modelos tabú de entrenamiento y aprendizaje, máquinas tabú y procedimientos de diseño tabú. Los resultados de este trabajo han sido prometedores para el desarrollo de modelos y paradigmas conexionistas suplementarios –produciendo niveles de eficiencia superiores a aquellos basados en máquinas de Boltzmann, y procesos para modificar conexiones de redes que dan mapeos más confiables de entradas a salidas.

Los años recientes han sido innegablemente testigos de avances significativos en la solución de problemas difíciles de optimización, pero también debe reconocerse que falta mucho por aprender.

7.7 GRASP (Greedy Randomized Adaptive Search Procedures)

GRASP es la más joven de las metaheurísticas surgidas durante la década de los 80 con el fin de resolver problemas difíciles en el campo de la optimización combinatoria. Una traducción literal de las palabras sería "Procedimientos de Búsqueda Ávidos, Aleatorizados y Adaptativos". Cada una de estas palabras caracteriza una de las componentes de esta metaheurística, la cual (a diferencia de sus parientes mayores que operan obre soluciones obtenidas previamente con el fin de mejorarlas), dirige la mayor parte de su esfuerzo a construir soluciones de alta calidad que son posteriormente procesadas con el fin de conseguir otras aún mejores.

GREEDY

La traducción de greedy según el diccionario es codicioso, avaro, goloso, voraz. En nuestro contexto se refiere a una estrategia que se ha empleado en una gran diversidad de problemas, obteniéndose en algunos de los casos la solución óptima y en otros buenas soluciones. La lógica de la estrategia es que se quiere alcanzar una meta en etapas sucesivas, en cada una de ellas se debe tratar de hacer lo mejor que se pueda.

Por ejemplo, pensemos en un ladrón que llega por la noche a un almacén con una gran bolsa, dispuesto a llevarse el botín de su vida. Este ladrón tiene a su alcance

toda una variedad de objetos que puede cargar consigo; es obvio que su objetivo será llevar aquellos que al venderlos le proporcionen el mayor beneficio y, para no despertar sospechas al tratar de colocarlos entre sus conocidos llevará a lo más uno de cada clase. Además, sólo puede transportar una carga limitada por el tamaño de la bolsa y su musculatura. La estrategia de un ladrón voraz sería empezar a tomar aquellos objetos que le proporcionen mayor beneficio por unidad de peso. Un ejemplo sencillo nos mostrará que esta no es la mejor forma de elegir su cargamento.

Si el ladrón se encuentra con tres objetos 1, 2 y 3 cuyos pesos y valores respectivos son 20, 40 y 60 kg. cada uno y 120, 200 y 240 pesos cada uno, y si el ladrón es capaz de cargar 100 kg., el objeto que elegirá en primer instancia será el 1, ya que le proporciona 6 pesos/kg (el más alto), a continuación elegirá el 2 con una razón de 5, y con esto agotará su capacidad de carga, ya que ambos objetos representan 60 kg. y ya no podrá cargar el objeto 3; el valor total del hurto sería de 320 pesos.

Por otra parte es fácil ver que si el ladrón eligiera los objetos 2 y 3, cuyo peso total es de 100 kg., podría llevarse un botín de 440 pesos.

RANDOMIZED

Se dice que un algoritmo es aleatorizado si su respuesta está determinada no sólo por los datos de entrada sino también por los valores producidos por un generador de números aleatorios. Varias de las demás heurísticas como algoritmos genéticos y recocido simulado comparten esta característica.

Una estrategia aleatorizada viene a ser particularmente útil cuando existen varias formas según las cuales un algoritmo puede efectuar un paso determinado, pero resulta difícil garantizar que alguna de ellas sea la mejor elección. Es frecuente que un algoritmo tenga que hacer muchas elecciones durante su ejecución. Si el beneficio de las buenas elecciones supera al costo de las malas, una selección al azar de las buenas y malas elecciones puede a la larga producir buenos resultados.

Esta es la idea en la cual se apoya este componente de GRASP. Después de varias réplicas de soluciones se habrá extraído en realidad una muestra de la población de todas las configuraciones posibles de un problema, y por supuesto se espera que esta muestra no sea representativa si no que tenga un fuerte sesgo hacia los elementos de más alta evaluación.

ADAPTIVE

El término adaptativo se ha usado frecuentemente en el contexto de sistemas de control, ya que en un número de campos muy importantes de la ingeniería es

necesario construir controles para sistemas que muestran con el tiempo cambios sustanciales en sus características dinámicas, cambios que se deben en la mayoría de los casos, a un medio ambiente en continua evolución.

7.8 ESTRATEGIAS DE GRASP Y SUS COMPONENTES

Esta metaheurística tiene una técnica de tipo iterativo. Cada iteración de GRASP consiste en una fase de construcción y otra de postprocesamiento. Se puede hacer una analogía con el método de las dos fases de la programación lineal. En la Fase I se construye una solución factible, así como una Fase II donde partiendo de tal solución se aplica el método simplex hasta llegar al óptimo.

Hay sin embargo una diferencia importante: mientras que en el problema lineal el único objetivo de la primera fase es llegar a una solución factible sin importar la calidad de ésta (de hecho se ignora completamente la función objetivo original), GRASP por su parte toma muy en cuenta la función objetivo en esta fase constructiva, con la intención de que al término de la iteración se cuente con una solución de alta calidad, sobre la que se efectúa un post-procesamiento o Fase II. Este post-procesamiento consistía –en los primeros días de GRASP- en procedimientos locales de búsqueda (de allí su nombre) y generalmente se reducían a un k-intercambio.

Una descripción del procedimiento dada en pseudocódigo es la siguiente:

Procedure GRASP
INPUT
WHILE (criterio de parada no satisfecho) **DO**
Preprocesamiento(Solución);
Fase Constructiva(Solución);
Postprocesamiento(Solución);
Actualizar la Solución(Solución, Mejor_Solución);
END;{while}
RETURN(Mejor Solución);

DISEÑO DE GRASP

El núcleo de la heurística es la primera fase, ya que la forma en que se lleve la segunda fase puede incluir métodos que exploren vecindades definidos en forma sencilla hasta procedimientos más sofisticados.
La primera tarea en el diseño de un GRASP es generalmente la selección de una función miope, que guíe la construcción de soluciones. La idea detrás de cualquier enfoque miope es la de efectuar el mejor movimiento (o tomar la mejor decisión) en cada paso. Para evaluar todos los posibles movimientos en cada paso de la Fase I, se utiliza una medida miope adaptativa, donde el término adaptativo se refiere al

hecho de que la función considera decisiones tomadas en pasos anteriores, previos al momento de considerar la siguiente decisión.

Esta fase consiste en ir construyendo iterativamente una solución factible, añadiendo un elemento cada vez. Si por ejemplo la instancia del problema a construir es una red y la solución buscada es un árbol con cierta propiedad, la construcción deberá partir de un bosque. En este caso los componentes que habría que agregar en cada paso podrían ser arcos, nodos o conjuntos de ambos. Además de medir la contribución de una componente en cada paso, debe cuidarse de que no se vaya a formar un ciclo en la red resultante.

En este punto es importante observar que si se sabe que ciertas subestructuras forman parte de una solución óptima, éstas podrán ser el punto de partida para iniciar la fase constructiva de GRASP. El procedimiento para obtener tales subestructuras es lo que se ha denotado como preprocesamiento.

Por ejemplo se conoce que en la solución óptima de un árbol de expansión mínima siempre se encuentran presentes uno o más arcos de peso mínimo, con las que se puede iniciar la construcción del árbol a partir de un bosque consistente en dos nodos unidos por un arco.

Este preprocesamiento es útil por varias razones siendo una de ellas el hecho de que puede reducir significativamente la etapa de construcción; otra no menos importante es que si se conocen diferentes subestructuras iniciales, cada una puede servir como punto de partida para obtener diferentes soluciones al problema, ya que esto produce mayor diversificación, lo cual es uno de los propósitos de GRASP

PROCEDIMIENTOS LOCALES DE OPTIMIZACIÓN.

El proceso de construcción de GRASP no garantiza optimalidad local, incluso con respecto a entornos definidos de forma tan simple como un intercambio de parejas. La idea es que GRASP produce una diversidad de buenas soluciones esperando que al menos una de ellas esté en un entorno de la solución óptima, o al menos en la de una solución muy cercana al valor óptimo.

Para ilustrar la metodología de GRASP veremos el siguiente ejemplo:

PROBLEMA DEL CONJUNTO INDEPENDIENTE MÁXIMO

Dada una red G=(N, A) donde N representa el conjunto de nodos y A, el conjunto de arcos de G, un conjunto independiente es un conjunto de nodos con la propiedad de que al tomar cualquier par de elementos, no existe ningún arco en A que los una. En el problema del conjunto independiente máximo se requiere encontrar un conjunto independiente de cardinalidad máxima.

7.9 FASE CONSTRUCTIVA DE GRASP

Para este problema debemos identificar antes que nada quienes serán, los nodos con los cuales se construirá la solución. Ya que la solución buscada es un subconjunto de N, lo más natural es usar nodos como elementos constructores, e irlos añadiendo, uno en cada paso constructivo, usando una función miope adaptativa para guiar la construcción, y cuidando que en cada paso se obtenga un subconjunto de N con la propiedad deseada (es decir, que sea un conjunto independiente). Se hará uso para este fin del concepto de grado de un nodo. Por lo tanto usaremos S* como el conjunto independiente a construir. Como subestructura inicial se usa el conjunto vacío, el cual por definición resulta ser independiente. Sea k=0 el contado del número de pasos constructivos, $N_k = N$ y $A_k = A$

A continuación se escoge una función miope. Para ello se sigue el siguiente razonamiento, si un nodo tiene grado cero (es decir no existe ningún arco en A que lo una a otro nodo de la red), es claro que este nodo forma parte del conjunto independiente, lo que nos lleva a la siguiente conclusión: cuanto menor sea el grado de un nodo, más fácil resultará incluirlo en un conjunto independiente. Parece entonces razonable usar como elección miope al nodo con menor grado en la red. Una vez elegido este nodo (si hay más de uno se puede establecer alguna regla para desempatar) se añade al conjunto independiente que estamos construyendo, y con esto concluirá el primer paso constructivo.

Al tratar de seleccionar un nuevo nodo, la situación habrá sido modificada por la selección hecha en el paso anterior. Una modificación obvia es el hecho de que ningún nodo adyacente al recién elegido se toma en consideración para el siguiente paso, ya que al agregarlo se viola la propiedad de independencia que queremos preservar. En el siguiente paso debemos por lo tanto reducir las alternativas al conjunto obtenido suprimiendo el nodo ya elegido, así como los nodos adyacentes a él. La otra modificación resulta de observar que al suprimir estos nodos, el conjunto resultante induce una subred de la red original G, en el cual el índice de cada nodo puede ser diferente. Si estas modificaciones se toman en cuenta, en el siguiente paso constructivo se obtendrá una función miope adaptativa.

En vez de seguir el procedimiento miope de selección, la fase constructiva de GRASP considera una lista restringida de candidatos, formada por todos los nodos de grado pequeño, aunque no necesariamente de grado mínimo. Sea D el grado mínimo de los nodos en la red, es decir: $D = \min \{ d_k(n) \mid n \in N_k \}$ y sea $\alpha > 0$ el parámetro de restricción por valor. La lista restringida por valor de candidatos es:

$$LRC = \{ n \in N_k \mid d_k(n) < (1+\alpha)D \}$$

De esta lista se escoge un nodo n al azar, y se coloca en el conjunto independiente, es decir $S^* = S^* \cup \{n\}$. El pseudocódigo para la fase constructiva de este problema sería el siguiente:

Procedure GRASP(G, S*)
$K=0$; $S^*=\phi$; $N = N_k$; $A = A_k$; $G_k = (N_k, A_k)$, $d_k(w) = d(w)$, $w \in N$;

WHILE ($N_k \neq \phi$) **DO**
Formar LRC;
Elegir al azar $n \in N$;
$S^* = S^* \cup \{n\}$;
$N_{k+1} = N_k - \{n\}$-ady(n);
$A_{k+1} = A_k - \{n, ady(n)\}$;
$K=k+1$;
Recalcular $d_k(w)$ para todo $w \in N$
END;{while}

Consideremos el ejemplo de la figura a) siguiente:

EJEMPLO 7.3
Sean los nodos {a,b,c,d,e,f} de grado 2 excepto el nodo e de grado 4 y α=0.6. La lista restringida de candidatos es {a,b,c,d,f}. Suponga que el nodo elegido al azar en la LRC sea el a. El conjunto independiente inicial sería S^* = {a} y la red resultante sería la mostrada en la figura b). En la red G_1 todos los nodos tienen el mismo grado y por lo tanto LRC = {c,e,f}. Si se selecciona el nodo e, el conjunto independiente resultante sería S^* = {a,e}. Observe que este conjunto es máximo, es decir ya no se puede agregar ningún otro nodo de N sin perder la propiedad de independencia. Aquí termina la primer iteración de GRASP.

Si en otra iteración se hubiera elegido inicialmente el nodo b, se obtendría al conjunto independiente S^*={b} y la red resultante G_1 sería entonces el que está en la figura c). En el siguiente paso constructivo LRC = {c,d,f} y si se selecciona el nodo f, se tiene S^* = {b,f} dejando dos nodos aislados en G_2. Si de estos dos que conforman la LRC se selecciona el nodo d, se obtiene el conjunto independiente máximo S^* = {b,f,d}

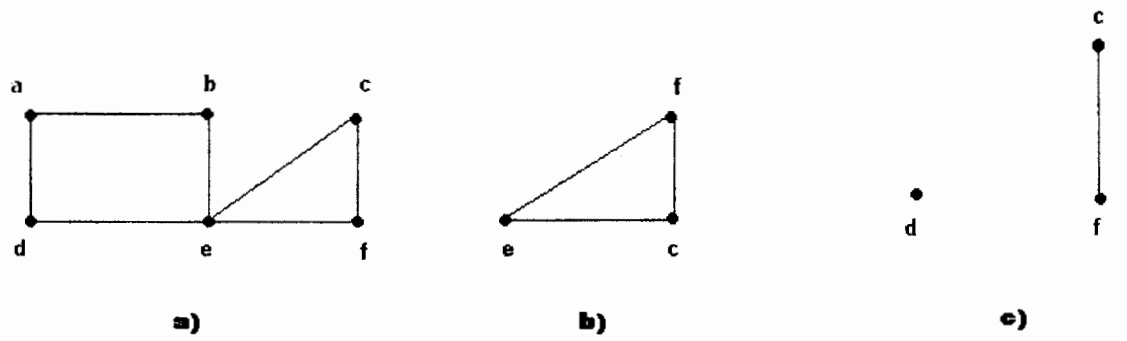


Figura 7.9

FASE DE POSTPROCESAMIENTO

Como ya se había mencionado, al terminar una fase constructiva, no hay ninguna garantía de que se haya obtenido la solución óptima. Conviene usar un procedimiento de búsqueda que mejore las soluciones obtenidas en la primera fase. Aquí se puede hacer uso de otras metaheurísticas como búsqueda tabú, recocido simulado o genéticos. Una idea muy sencilla consiste en un procedimiento de búsqueda local sobre k-vecindades. La idea es tomar un subconjunto independiente S de tamaño m, y considerar todos los subconjuntos de S de cardinalidad k (para un parámetro k conocido, positivo y menor que m). Para cada uno de estos subconjuntos $W_k = \{w_1,w_2,\ldots,w_k\}$ aplicar una búsqueda exhaustiva para encontrar un conjunto independiente máximo en la red inducido por los nodos de G que no son adyacentes a los nodos en el conjunto $S' = S-W_k$. Si el conjunto independiente que resulte, por ejemplo M, tiene una cardinalidad mayor que k, el conjunto de nodos $S' \cup M$ es un conjunto independiente y su cardinalidad es mayor que la de S. este procedimiento puede aplicarse al conjunto independiente.

7.10 APLICACIONES DE GRASP

La primer aplicación de GRASP que aparece en la literatura data de 1989, y la gran mayoría de los artículos comenzaron a aparecer en 1991. Algunas áreas de aplicación son:

- Secuenciación, Programación y Planificación.
- Teoría de Gráficas
- Cobertura de Conjuntos
- Problemas de Localización
- Problemas de Transporte
- Asignación Cuadrática
- Lógica

CONCLUSIONES

GRASP es una herramienta poderosa para la construcción de soluciones de muy alta calidad, sin embargo existen todavía muchos aspectos por explorar, algunos de ellos son:

Es muy importante la Lista Restringida de Candidatos, una manera de mejorarla consiste en elegir buenos parámetros para resolver un problema. Empíricamente se sabe que una lista de tamaño 3 proporciona los mejores resultados.

Otro aspecto es la selección al azar de un elemento de la lista. La regla genérica es que se elija cualquier elemento con la misma probabilidad. Sin embargo, si a cada elemento se le asigna una probabilidad de acuerdo con ciertas reglas, se podrían controlar los procesos para dirigir la construcción en algún sentido. Por ejemplo en el caso que vimos del conjunto independiente una manera de asignar probabilidades sería guardar en una matriz el número de veces que aparece un nodo en las soluciones construidas; una vez que se tiene la LRC calcular la frecuencia relativa con la que han aparecido cada uno de los elementos, y sesgar la probabilidad de elección de manera que sean favorecidos aquellos con menos apariciones.

La conjunción de GRASP con otras heurísticas es también un campo prometedor para investigaciones futuras.

7.11 REDES NEURONALES

Las redes neuronales constituyen una floreciente tecnología que puede ayudar principalmente en una gran cantidad de problemas. Son sistemas de cálculo que se asemejan a las características biológicas del cerebro, y están siendo adaptadas para su uso en una variedad de aplicaciones comerciales, militares y tecnológicas, que van desde el reconocimiento de patrones hasta la optimización y planeación.

ANTECEDENTES BIOLÓGICOS

A pesar de que el cerebro humano ya había sido estudiado desde la Edad Media e incluso antes, su estructura profunda no fue desvelada hasta fines del siglo pasado. En ese momento los científicos, en controversia, se dividían en reticularistas y neurologistas. Para los primeros el cerebro es una glándula cuya secreción se distribuye por todo el cuerpo por medio de fibras nerviosas.

En cambio, para los neurologistas, el cerebro es un sistema complejo constituido por unidades individuales bien diferenciadas, llamadas neuronas, unidas unas a otras por una malla de fibras nerviosas. El trabajo desarrollado entre 1894 y 1911 por el médico español Ramón y Cajal disipó la controversia a favor de la teoría de los neurologistas . Cajal empleó una técnica descubierta por Golgi en 1888 basada en la tinción de fibras nerviosas e identificó las neuronas como las unidades

constitutivas del cerebro. El examen de esta estructura tan compleja por medio del microscopio electrónico reveló que a pesar de la gran variedad existente de neuronas, todas tienen las mismas partes principales:

- Cuerpo (o soma)
- Dendritas
- Axón (o cilindro eje)

En la figura 7.10 se muestran las partes de una neurona

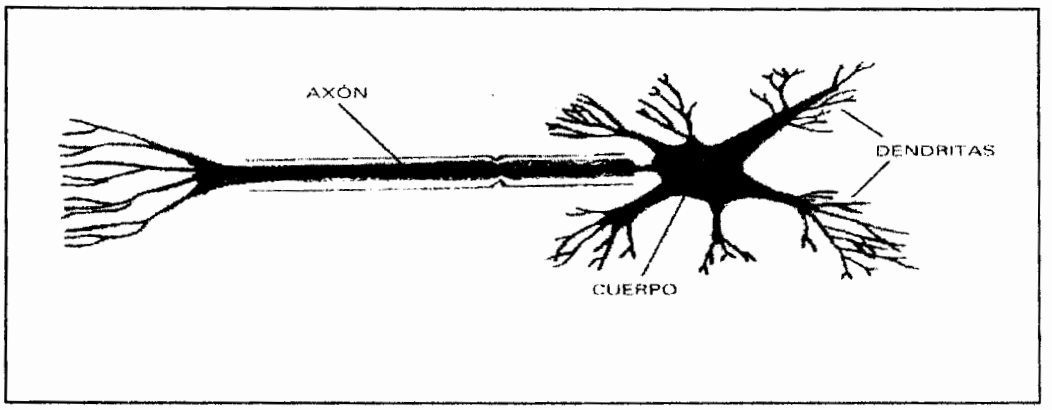


Figura 7.10

Esta estructura se corresponde con la de un proceso, con sus respectivas funciones de entrada y salida. Las dendritas reciben las señales de las neuronas adyacentes y las transmiten al cuerpo en forma de un potencial eléctrico. Estas señales eléctricas son integradas por el cuerpo celular (soma). Si este potencial eléctrico es superior a un valor umbral, el soma genera un corto impulso eléctrico. Este impulso se transmite por el axón, que es una fibra nerviosa con una longitud que varía entre unos milímetros y varios metros. El axón se ramifica y dirige el impulso a varias neuronas vía sinapsis.

La sinapsis es la unión del axón con otras neuronas. En las redes neuronales, las neuronas se consideran como cajas negras, y por tanto no se necesita una descripción más detallada de la neurona. No obstante, no es posible comprender la capacidad del cerebro para realizar funciones tan complejas como la percepción visual o el control motriz si se permanece al nivel de la neurona.

MODELOS DE NEURONAS

El primer modelo artificial de neurona fue establecido por Mc. Culloch y Pitts (1943). Este modelo consistía en un dispositivo no lineal de múltiples entradas con interconexiones "con peso". En este modelo las interconexiones afectadas por los pesos representaban las dendritas; el cuerpo celular se representaba por una función no lineal y transmitía la salida. Esto se puede ver en la figura 7.11

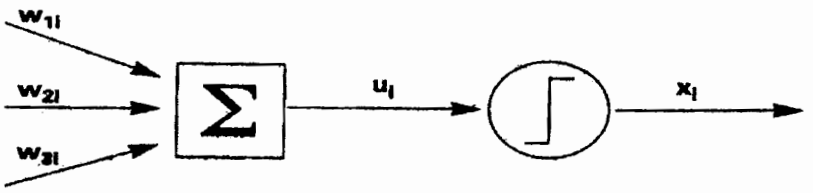


Figura 7.11

Veamos un caso muy simple en el cual la neurona tiene dos entradas

Veamos un caso muy simple en el cual la neurona tiene dos entradas, representadas por un vector x = (x_1,X_2). A los pesos para las interconexiones los llamaremos w_1 y w_2, y la función no lineal es una función umbral g(x), cuyo valor umbral es Θ , siendo la salida un escalar y. Esto se puede representar como:

$$y = g\left(\sum_{k=1}^{2} w_k x_k\right)$$

con

$$g(h) = \begin{Bmatrix} -1 \text{ si } h \leq \Theta \\ -1 \text{ si } h \rangle \Theta \end{Bmatrix}$$

A pesar de su simplicidad, este modelo artificial puede resolver funciones booleanas. Por ejemplo, esta simple red es capaz de resolver la función booleana OR: para los valores boléanos + 1 (cierto) y –1 (falso), basta tomar los pesos w_1 = w_2=1, y el umbral Θ = -0.5.

Cuando este modelo se extiende a vectores de entrada n-dimensiones reales, la ecuación es :

$$y = g\left(\sum_{k=1}^{n} w_k x_k\right)$$

A la función g se le denomina función de transferencia o ganancia, y representa cómo la neurona calcula la salida y para una entrada x, y unos pesos dados. Cuando g es una función umbral, como en el caso anterior, la neurona se llama unidad umbral. Pero esta clase de funciones umbrales no es la única posible. En la figura 7.12 se muestran otras funciones de ganancia de uso frecuente.

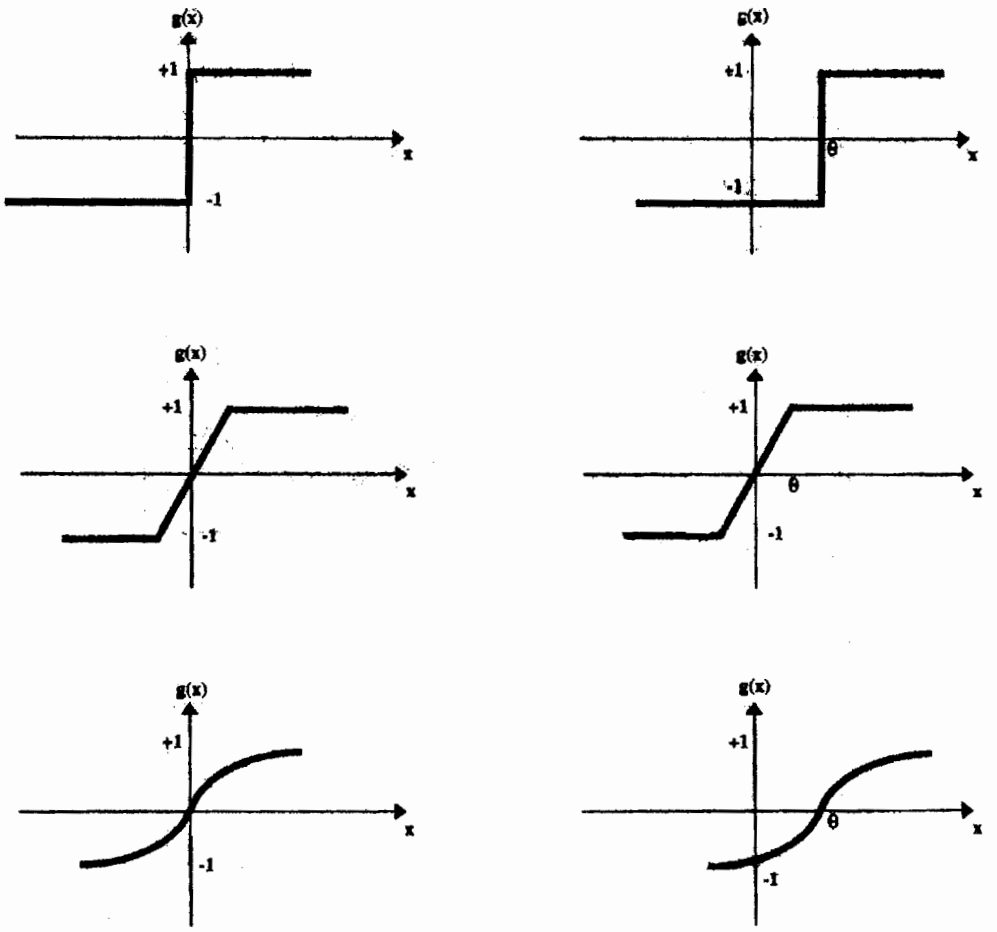


Figura 7.12

7.12 ARQUITECTURAS

Una red neuronal artificial está formada por un conjunto de neuronas interconectadas entre ellas. El modo en que se interconectan constituye la arquitectura de la red.

Hay varias arquitecturas y las más comunes son las redes por capas, las redes recurrentes y las redes de conexión lateral Estas arquitecturas están muy ligadas a la regla de aprendizaje para adiestrar a la red. En este apartado se presentan estas arquitecturas y el siguiente las correspondientes reglas de aprendizaje.

REDES NEURONALES POR CAPAS

La estructura de una red neuronal de este tipo está dispuesta en capas. En estas redes cada capa de neuronas recibe señales sólo de las capas previas. Por esto se llaman redes feed-forward o perceptrón. La primera red feed-forward fue

presentada por Rosenblatt (1957), y tenía una capa de entrada para el vector de entrada, una unidad para cada dimensión del espacio de entrada y una capa de salida constituida por un conjunto de unidades umbral. La figura 7.13 ilustra la arquitectura del perceptrón. En la primera capa no se realiza cálculo (capa de entrada) y por este motivo se considera a esta red como una capa.

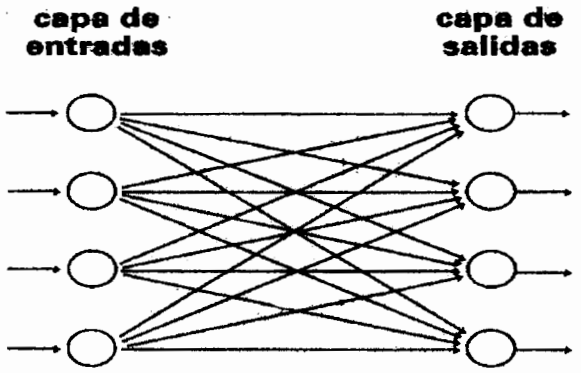


Figura 7.13

La segunda clase de red por capas es la multicapa con feed-forward, también denominada perceptrón multicapa. En esta clase, se tiene al menos una capa (las denominadas "capas ocultas" entre las de entrada y la de salida. Cuando cada unidad de una capa se conecta a cada nodo de la capa adyacente siguiente, la red se denomina "totalmente conectada" en contraposición a la que es "parcialmente conectada". En la figura 7.14 se tiene un ejemplo de cada red.

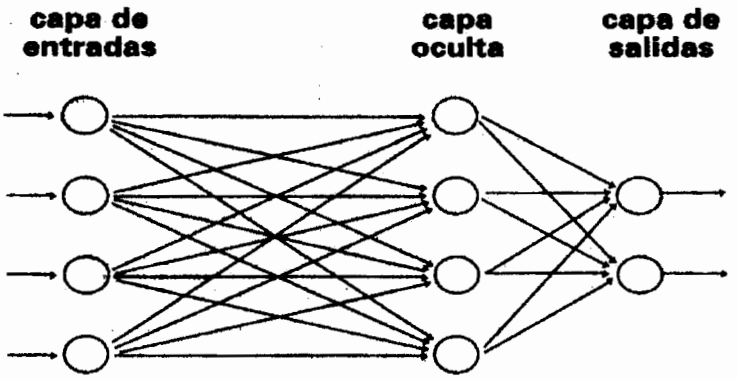


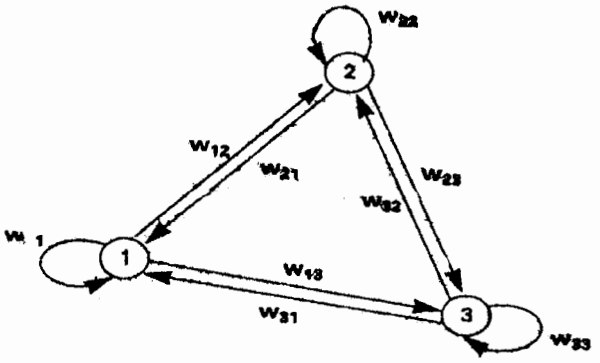
Figura 7.14

REDES NEURONALES RECURRENTES

En estas redes cada neurona se conecta a todas las neuronas de la red, incluso consigo misma, lo que significa que se tiene bucles en la arquitectura de la red. Las entradas de una neurona son las salidas del resto de las neuronas de la etapa previa. En estas redes es muy importante la dinámica de las mismas. La salida de

cada neurona se llama "estado de la neurona. La dinámica de la red es simplemente la evolución de la red desde un estado a otro. La red es estable cuando su estado permanece igual tras varias interacciones.

La estabilidad es un concepto muy importante en esta clase de redes. La red de Hopfield es un caso muy importante de esta clase. La figura 7.15 ilustra un ejemplo de red de Hopfield.



Ejemplo de red recurrente con tres neuronas
Figura 7.15

REDES CONECTADAS LATERALMENTE

En esta clase de redes, las neuronas se colocan en los nodos de un retículo dimensión 1 ó 2 (aunque puede haber dimensiones mayores, no se suelen utilizar). La figura 7.16 ilustra un ejemplo de una red de dimensión uno, conectada lateralmente y otra de dimensión 2. La red de esta clase más común es la de Kohonen (1982)

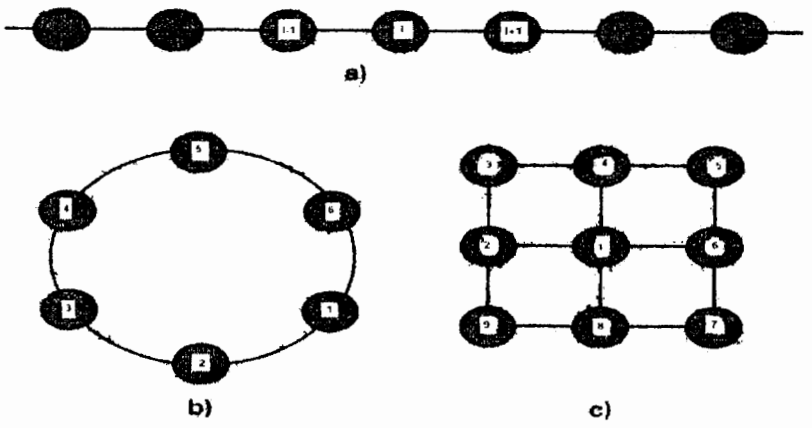


Figura 7.16. Tres arquitecturas diferentes para el modelo de Kohonen: a) una cadena, b) un anillo y c) una matriz bidimensional

REGLAS DE APRENDIZAJE

La característica más interesante de la redes neuronales artificiales es su capacidad de aprendizaje. Hay dos modos principales de aprendizaje: el aprendizaje supervisado y el no supervisado.

En la práctica, las reglas de aprendizaje especifican cómo adaptar los pesos sinápticos. Esta adaptación se realiza mediante una interacción continua entre la red neuronal y el entorno.

La primera hipótesis sobre el proceso de aprendizaje en los cerebros naturales fue formulada por Hebb (1949). Introdujo el siguiente principio: "Los pesos de sinapsis aumentan cuando tanto las neuronas pre-sinápticas como la post-sinápticas se activan simultáneamente.

Una componente esencial en el aprendizaje supervisado es la existencia de un maestro o supervisor. La misión de la red neuronal es dar la salida deseada para cada entrada. Cuando se pasa un vector de entrada a la red, el instructor sabe cual ha de ser la salida (salida deseada). La regla de aprendizaje adaptará los pesos de las sinapsis de tal modo que el error entre la salida real y la deseada sea mínimo. El perceptrón y las redes multicapa utilizan este método de aprendizaje.

El procedimiento seguido para "enseñar" a una red neuronal es el siguiente: se diseña un conjunto de ejemplos de entrenamiento. Cada ejemplo consta de un par de vectores, uno de entrada y otro de salida deseada. Durante la fase de entrenamiento, los pesos sinápticos se adaptan para minimizar el error entre las salidas real y deseada. Cuando termina la fase de entrenamiento (por ejemplo cuando el índice de error sea menor del 1%) dejan de variar los pesos sinápticos y quedan fijados para las operaciones siguientes. A continuación viene la fase de generalización Basándose en los pesos determinados en la fase de entrenamiento, la red encontrará la salida adecuada, incluso aunque el supervisor no conozca dicha salida.

En el proceso de aprendizaje no supervisado, no se dispone de instructor. En este caso no hay conjunto de entrenamiento, y no se conocen las salidas deseadas. La tarea de una red no supervisada es clasificar los vectores de entrenamiento en clases y grupos. La tarea de aprendizaje se realiza aplicando el concepto de similitud. La similitud de dos vectores se suele medir por su producto escalar (ésta no es la única medida). La regla de aprendizaje competitiva se usa para realizar un proceso no supervisado de aprendizaje. Cuando se presenta un vector de entrada a la red, se selecciona una neurona (lo que significa que todas las neuronas tratan de ser seleccionadas), pero sólo la ganadora es seleccionada. El modelo de Kohonen es un ejemplo de red que utiliza el método no supervisado.

7.13 APROXIMACIÓN ESTADÍSTICA PARA EL PROBLEMA DE LA OPTIMIZACIÓN

En este apartado exponemos la relación entre la termodinámica y los problemas de optimización combinatoria. En la última década se ha visto que los problemas de optimización combinatoria están relacionados, de forma sencilla, con la física estadística:

- Una instancia de un problema de optimización combinatoria es equivalente a una muestra en física estadística.
- Una prueba en optimización combinatoria es una configuración en física estadística.
- La función de costo equivale a la energía.
- Una solución óptima corresponde a la energía mínima
- El costo mínimo es el estado de energía mínima.

Consecuentemente, el primer paso para solucionar el problema de la optimización en el marco de la física estadística es inscribirlo en términos de estados que sean variables discretas en un espacio euclideano. Entonces se define una función global de energía sobre el conjunto de todos los estados posibles. El sistema evoluciona hacia un estado de energía mínima que debe satisfacer las condiciones del problema y también optimizar la función de costo.

Los problemas complejos de optimización son generalmente muy heterogéneos. Esto está relacionado con las propiedades de desorden y frustración que tienen. Por ejemplo, en el problema del agente viajero, este se frustra porque se esfuerza en ir inmediatamente a la ciudad más cercana, lo que desafortunadamente, es una mala estrategia porque debe respetar la condición de visitar todas las ciudades sólo una vez. La no homogeneidad de este problema hace que sea muy grande el número de estados metaestables. Una manera de escapar de estos estados es introducir "ruido", que en un sistema termodinámico se mide por medio de la temperatura. Se define una distribución de probabilidad sobre el conjunto de todas las distribuciones posibles, y la mayor probabilidad se asigna a la energía más baja. Esta distribución se parametriza por medio de un parámetro de control: la temperatura. Conforme disminuye la distribución se hace más discriminadora. A una temperatura de 0° la probabilidad se concentra en torno a la configuración de menor energía. Esto sugiere que en vez de minimizar directamente la función de costo, podrían evaluarse medias estadísticas a un nivel dado de ruido. Observando entonces el cambio de estas medias a medida que el nivel de ruido decrece hacia cero. Este método fue propuesto por Peterson y Soderberg (1989) y Herault y Niez (1991) para distintos problemas de optimización. Estos son los conceptos básicos para emplear la aproximación de física estadística para codificar y solucionar problemas de optimización.

EL ALGORITMO DE KOHONEN

El algoritmo de Kohonen, también denominado en inglés *self-organizing feature map algorithm*, es un método de inspiración biológica diseñado para construir una representación estructurada de un espacio de entradas de gran dimensión. El objetivo del algoritmo de Kohonen es encontrar una función del espacio de entrada X, en un retículo discreto N (por lo general bidimensional) de neuronas formales. Esa función trata de reflejar las relaciones topológicas de vecindad entre las entradas en la disposición de las correspondientes neuronas en el retículo.

Este modelo tiene la habilidad de conseguir una representación del espacio de entrada en un espacio de menor dimensión. Esto ha sido probado y analizado por Kohonen para el caso de un modelo de red compuesta de una matriz lineal de neuronas con entradas escalares, introduciendo una medida de desorden y demostrando que esta medida decrece la mayor parte del tiempo como resultado del algoritmo de entrenamiento. La prueba no fue realmente formal, ya que de hecho la medida del desorden en ocasiones crece.

El proceso de entrenamiento es un proceso markoviano según Cottrell y Fort (1987), cuyos estados vienen dados por los valores de los pesos. Demostraron que en el caso unidimensional, los estados de la red que se corresponden con la función ordenada son estados absorbentes del proceso de Markov.

DESCRIPCIÓN DEL MODELO DE KOHONEN

Comenzaremos hablando de la **estructura de la red**. Se tiene un conjunto de N neuronas, cada una de las cuales recibe señales de entrada:

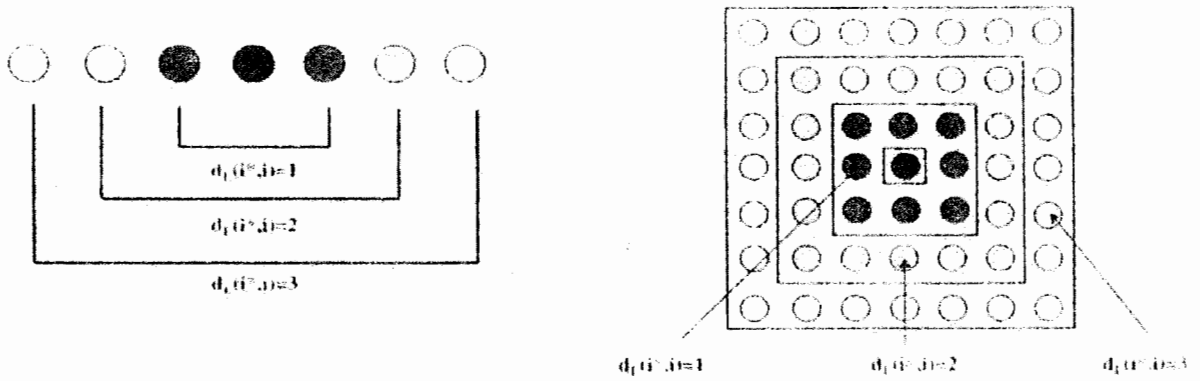
$$\vec{m}_j \in \Re^d, \vec{m}_j =< m_{j1},...,m_{jN} >$$

donde d es la dimensión del espacio de entrada. En esta red tenemos dos tipos de conexiones: las conexiones entre neuronas de la red, y las conexiones entre las neuronas y su entorno. Esta red se distribuye por lo general en un retículo de dimensión arbitraria, aunque muchas aplicaciones usan retículos bidimensionales. Sin embargo, en los problemas de optimización combinatoria usaremos retículos unidimensionales. En este caso, las neuronas se colocan en un anillo o cadena, y cada neurona está conectada a todos los entradas. Por lo tanto, si el espacio de entradas es d-dimensional, el vector de pesos sinápticos asociados con la neurona i es

$$\vec{w}_i = (w_{1i},...,w_{di})$$

Por lo que se refiere a las intersecciones laterales, cada neurona no sólo está conectada con el espacio de entradas sino que también lo está con otras

neuronas (intersecciones laterales). Para cada neurona se define un entorno, así como unas relaciones de entorno entre neuronas, las cuales definen los vecinos más cercanos a la neurona, los segundos más cercanos, etc. A esta relación se le llama **distancia lateral**, la cual se define de modo que la distancia de una neurona a su vecino más cercano es 1, a su segundo vecino más cercano es 2, etc. Como se puede ver en la siguiente figura 7.17.



Distancia lateral d_L (i*,i), entre la neurona seleccionada i* y la neurona i
Figura 7.17

A la fase de competencia entre neuronas le sigue la fase de cooperación. Una vez se ha seleccionado la neurona i* , se aplica un procedimiento de adaptación con un principio simple: el vector sináptico w_{i^*} es empujado en la dirección de la señal de entrada **m**, consiguiendo un efecto de especialización de las neuronas. Cada neurona responderá con preferencia a un input específico, y la actualización de los pesos w_i mejorará la respuesta de la neurona ganadora ante el mismo input, y (ésta es la parte cooperativa del algoritmo) mejorará la respuesta de las neuronas en el entorno de i* ante ese input. Sin embargo, la mejora decrece según se hace mayor la distancia lateral, siendo la función lateral la encargada de controlar la cantidad mejorada. La función más utilizada para representar la interacción lateral es

$$\Gamma(i,i^*) = \frac{1}{2}\exp\left(\frac{-d_L^{\,2}(i,i^*)}{2\sigma^2}\right)$$

Donde $d_L(i,i^*)$ es la distancia lateral entre la neurona i y la neurona elegida i*. Otra función muy común es la denominada "sombrero mexicano", introducida por Kohonen. Esta función incrementa la respuesta de las neuronas más próximas a la ganadora, decreciendo la respuesta de las más alejadas. La figura 7.18 muestra dicha función.

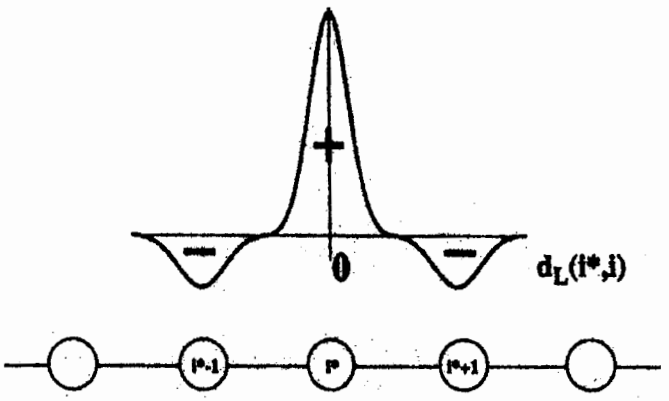


Figura 7.18

Respecto a la **regla de aprendizaje**, la actualización de los pesos de las neuronas viene dada por

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)\Gamma(i,i^*)(m_j(t) - w_{ij}(t)) \quad \forall j = 1,...,d$$

siendo $\Gamma(i,i^*)$ la interacción lateral dada por la ecuación de gamma y $\eta(t)$ el parámetro de adaptación, cuyo valor decrece con el tiempo de forma que el algoritmo converja, es decir, los pesos w_i dejen de cambiar.

ALGORITMO DE KOHONEN

PROPÓSITO: Resolver problemas de optimización combinatoria aplicando redes neuronales.

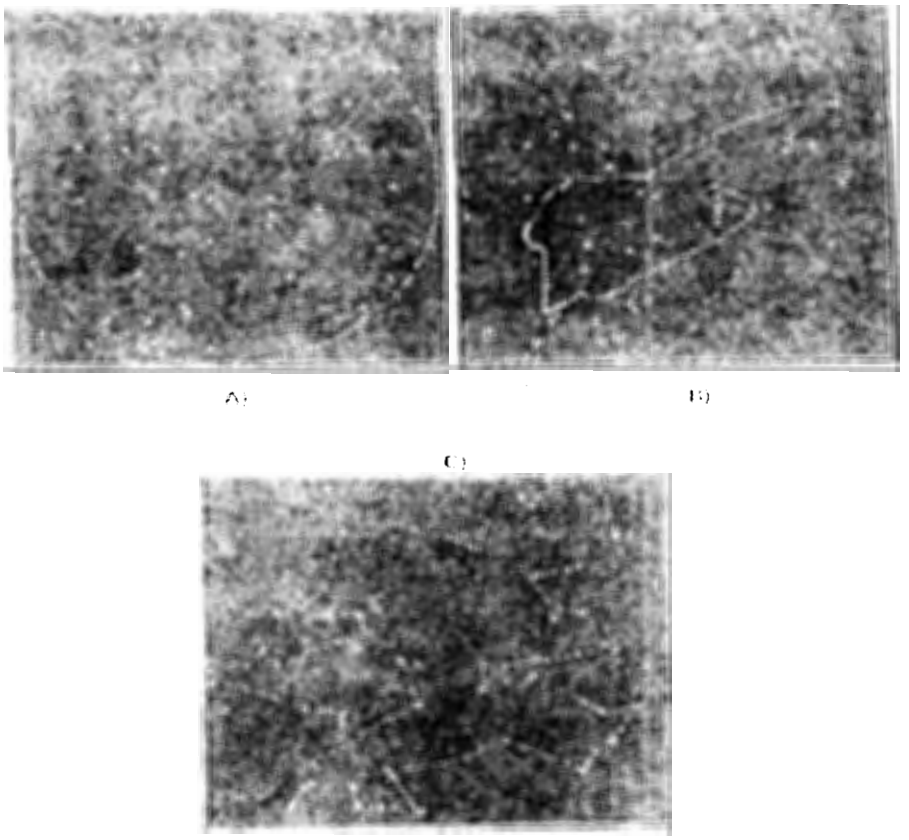
DESCRIPCIÓN

PASO 1 *Inicialización*. Generar aleatoriamente los pesos sinápticos w_{ij}.
PASO 2 *Selección de la entrada*: seleccionar aleatoriamente el input **m**.
PASO 3 *Selección de la neurona ganadora*: seleccionar la neurona más parecida a la entrada según la ecuación del vector de entrada
PASO 4 *Regla de adaptación*: actualizar los pesos sinápticos de las neuronas según la ecuación de $w_{ij}(t+1)$
PASO 5 *Actualizar* : α y η.
PASO 6 *Criterio de detenimiento*: los pesos w_{ij} ya no cambian

LAS REDES NEURONALES Y EL PROBLEMA DEL AGENTE VIAJERO.

Usando el modelo de Kohonen se considera una arquitectura especial de red neuronal. Las neuronas estarán organizadas espacialmente sobre un anillo unidimensional. En el primer paso, elegimos una ciudad, y comparamos su posición con la de todas las neuronas, eligiendo la más cercana y moviéndola hacia la ciudad. Las neuronas vecinas de la elegida son desplazadas también

hacia la ciudad pero con una intensidad decreciente de acuerdo con la distancia lateral. En la figura 7.19 se presenta la evolución del algoritmo desde el estado inicial hasta el final.



Evolución del anillo desde el estado inicial A), tras varias iteraciones B), y el estado final C)
Figura 7.19

Para evitar la oscilación de una neurona alrededor de ciudades vecinas, el número de neuronas debe ser mayor que el de ciudades. En la figura 7.20 se observa la evolución de la función lateral según se va actualizando el parámetro α.

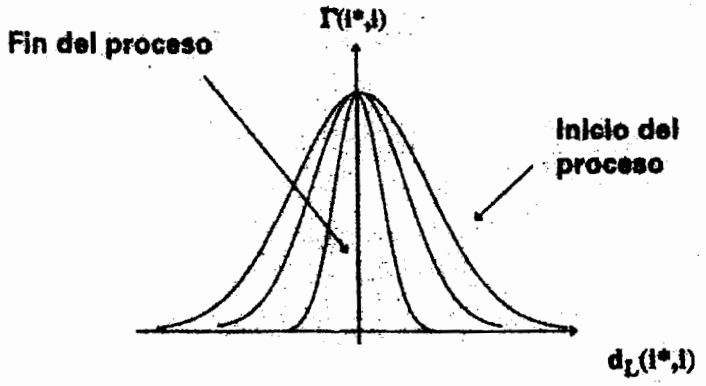


Figura 7.20

ALGORITMO DE KOHONEN PARA EL TSP

PROPÓSITO: Resolver el Problema del Agente Viajero usando redes neuronales.

DESCRIPCIÓN

PASO 1 *Inicialización.*
En el estado inicial el anillo es un círculo con M neuronas para N ciudades (N<M)

PASO 2 *Selección de la entrada*:
Elegir aleatoriamente una ciudad, representada por el vector **m**= $<m_1, m_2>$ en el espacio euclideano.

PASO 3 *Selección de la neurona ganadora*:
Seleccionar la neurona más parecida a la entrada según la ecuación:

$$\|\vec{w}_{i^*} - \vec{m}\| \leq \|\vec{w}_i - \vec{m}\| \quad \forall i = 1,...,M$$

PASO 4 *Regla de adaptación*:
Actualizar los pesos sinápticos de las neuronas según la ecuación de $w_{ij}(t+1)$ con d= 2

PASO 5 *Actualizar* : α y η.

PASO 6 *Criterio de detenimiento:*
Todas las ciudades tienen alguna neurona a menos distancia que 10^{-4}.

7.14 CONCLUSIONES

Aunque existen diferentes enfoques para la solución de problemas de optimización usando redes neuronales artificiales solamente nos centramos en el método de Kohonen por considerar que es uno de los más representativos, este método se basa en las redes auto-organizadas y una de sus variantes son las redes elásticas de Durbin y Willshaw. Otros métodos como el modelo de

Hopfield que se basa en la física estadística cuenta como una de sus variantes el método del campo medio. En este enfoque es fácil encontrar una formulación del problema de optimización de modo que podamos utilizar los métodos de la física estadística, aunque una vez que hallamos la función de energía correspondiente, no es seguro que lleguemos a la solución óptima, y en muchas ocasiones se queda atrapado en mínimos locales de mala calidad. El método del campo medio da mejores resultados que el modelo simple de Hopfield, aunque no para todos los problemas. Por ejemplo, en el agente viajero y en el de enrutamiento, en general los resultados no son demasiado buenos.

Por el contrario, las redes auto-organizadas ofrecen muy buenos resultados en comparación con los métodos clásicos (aunque no obtengan los mejores resultados para el agente viajero), pero ofrecen la dificultad de cómo extenderlas a problemas más generales que los de rutas. Este enfoque resulta muy prometedor para problemas de rutas con restricciones laterales.

Todas estas redes están basadas en computadoras tradicionales tipo Von Newmann. Es de esperar que las implementaciones físicas de estas redes provoquen significativas mejoras en su eficiencia.

7.15 NOTAS HISTÓRICAS

En 1986 Glover describió el concepto básico de búsqueda tabú como "una metaheurística que se sobrepone a otra". La técnica en general consiste en evitar desperdiciar tiempo en circuitos a través de movimientos que penalicen movimientos en cada iteración que nos lleven a puntos en el espacio de soluciones que hayan sido visitados previamente (de ahí el nombre de Tabú). La búsqueda tabú es bastante nueva, Glover atribuye su origen en 1977. Este método aún es objeto de investigación y está en constante mejora y desarrollo. Una motivación parcial para el método de búsqueda Tabú es la observación de que el comportamiento humano opera con un elemento aleatorio que arroja un comportamiento inconsistente dadas situaciones similares. Como Glover puntualiza, la tendencia resultante de desviarse de un curso programado, podría lamentarse como fuente de error pero también podría probar ser una fuente de ganancias. El método tabú opera de esta manera con la excepción de que los nuevos cursos o caminos no se seleccionan aleatoriamente. En lugar de esto, la búsqueda tabú procede de acuerdo a la suposición de que no hay un punto para aceptar una nueva solución (pobre) a menos que ya se haya evitado una trayectoria que ya se había investigado. Esto asegura nuevas regiones del espacio de soluciones del problema que se investigarán con el propósito de evitar mínimos locales y poder así encontrar la solución deseada.

BIBLIOGRAFÍA

1.- **ADENSO, D.** (Coordinador). "Optimización Heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería". Editorial Paraninfo 1996.

2.- **AGIN, N.** "Optimun Seeking with Branch and Bound". Management Science 13, p B-176-185, 1966.

3.- **AHO, HOPCROFT & ULLMAN** "Estructura de Datos y Algoritmos". Ed. Sitesa, México 1988.

4.- **BALAS, E** "An additive Algorithm for Solving Linear Programs with Zero-One Variables" . Operations Research 13, p. 517-546, 1965.

5.- **BALAS, E.** "A Note on the Branch and Bound Principle" .Operations Research 16, p 442-445, 1968.

6.- **BALINSKI, M.L.** "Integer Programming: Methods, Uses, Computation". Man. Sci. 12, No. 3 p. 253-313, 1965.

7.- **BOFFEY, T.B** "Graph Theory in Operations Research".Capítulo 3, Mac Millan 1982.

8.- **CHISTOFIDES, N.** "Combinatorial Optimization". Ed. John Wiley, 1979.

9.- **DAKIN, R.J.** "A Tree-Search Algorithm for Mixed Integer Programming Problems", The Computer Journal 8, p. 250-255, 1965.

10- **DRIEBEEK, N.J.** "An Algorithm for the Solution of Mixed Integer Programming Problems", Man. Sci. 12, No. 7, 576-587, 1966.

11.- **FUENTES MAYA, S.** "Programación Entera" . DEPFI, 1985.

12.- **FRED GLOVER, M.** " Tabu Search". kluwer Academic Publishers, 4ta. Edición 2001.

13.- **GEOFFRION, A.M. Y MARSTEN, R.E.** "Integer programming Algorithms: A Framework and State-of the Art Survey". Management Science 18, no. 9 p.465-491, 1972.

14.- **GONDRAN, M. Y MINOUX, M.** "Graphs and Algorithms". Ed. John Wiley, 1979.

15.- **HOROWITZ, E Y SAHNI, S.** "Fundamentals of Computer Algorithms". Capítulo 8, Computer Science Press, 1984.

16.- **JIANYU, Y.** "El Problema del Viajero y sus Extensiones". Tesis de Maestría DEPFI.UNAM, 1985.

17.- **KOLESAR, P.J.** "A Branch and Bound Algorithm for the Knapsack Problem". Management Science 13, p. 723-735, 1967.

18.- **LAND, A.H. Y DOIG, A.G.** "An Automatic Method of Solving Discrete Programming Programming Problems". Econométrica 28, 497-520, 1960.

19.- **LARDER, H.** "The Origin Of Operational Research". Opns Res. 32, p. 465-475, 1984.

20.- **LAWLER, E.L., WOOD, D.E.** "Branch-and-Bound Methods: A Survey". Opns. Res.14, p.669-719,1966.

21.- **LAWLER, E.L Y LENSTRA, J.K. RINNOY KAN, A.H.G. Y SHMOYS D.B.** "The Traveling Salesman Problem". Opns. Res. 34, p. 698-717, 1963.

22.- **LITTLE, J., MURTY, K., SWEENY, D. Y KAREL, C.** "An Algorithm for the Traveling Salesman Problem". Ed. Wiley, 1985.

23.- **MARTELLO, S. y TOTH, P.** "Algorithm for the Solution of 0-1 Single Knapsack Problem", Computing 21, p.81-86, 1978.

24.- **MITTEN, L.G.** "Branch and Bound Methods: General Formulation and Properties" Opns. Res. p.24-35, 1969.

25.- **MORSE, P.M.** "The Beginnings of Operations Research in the United States" Opns Res. 34, 10-17, 1986.

26.- **NEMHAUSER, G y WOLSEY L.** "Integer and Combinatorial Optimization" , Wiley Ed. 1988.

27.- **OCHOA-ROSSO F.** "Applications of Discrete Optimization Techniques to Capital Investment and Network Synthesis Problems" Research Report R68-42 Massachusetts Institute of Technology, Cambridge 1968.

28.- **PARKER, G.** "Discrete optimization" Academic Press 1988.

29.- **PRAWDA, J.** "Métodos y Modelos de Investigación de Operaciones" Limusa, 1979.

30.- **RUY , E., NELSON, M.** "Algoritmos e Heurísticas" Editora da Universidade Federal Fulmínense (EDUFF)", Niteroi – RJ 1994.

31.- **SALKIN, H.M.** "Integer programming", Addison-Wesley, 1974.

32.- **TAHA, H.A.** "Integer Programming Theory, Applications and Computations", Academic Press, 1975.

32.- **THENSEN, A** "Computer Methods in O.R." Capítulo 8, Academic Press, 1978.

34.- **ZIONTS, S.** "On An Algorithm for the Solution of Mixed Integer Programming Problems", Man. Sci.15, 113 – 116.

Esta obra se terminó de imprimir
en mayo de 2002
en el taller de imprenta del
Departamento de Publicaciones
de la Facultad de Ingeniería
Ciudad Universitaria, México, D.F.
C.P. 04510

Secretaría de Servicios Académicos

El tiraje consta de 300 ejemplares
más sobrantes de reposición