UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE ESTUDIOS DE POSGRADO

"METODOS HEURÍSTICOS PARA LA SOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN COMBINATORIOS".

TESIS QUE PARA OBTENER EL GRADO DE
MAESTRO EN INGENIERIA
(INVESTIGACIÓN DE OPERACIONES)
PRESENTA:

MERCED DAVID PEREZ CANO

DIRIGIDA POR:

DR. RICARDO ACEVES GARCIA

NOVIEMBRE DE 1996

01168

VNIVERSIDAD NACIONAL
AVPNOMA DE
MEXICO

DIVISION DE ESTUDIOS DE POSGRADO
FACULTAD DE INGENIERIA

RECIBI COPIA DE: ( ) TRABAJO ESCRITO

(XXXX) TESIS

DESARROLLADO POR EL ALUMNO: MERCED DAVID PEREZ CANO

PARA PRESENTAR EXAMEN:

( ) DE ESPECIALIZACION

(XX) DE GRADO

EN INGENIERIA: INVESTIGACION DE OPERACIONES

		FIRMA	FECHA
PRESIDENTE:	DR. JOSE JESUS ACOSTA FLORES		29-10-96
VOCAL:	DR. RICARDO ACEVES GARCIA		28/X/96
SECRETARIO	M EN I. IDALIA FLORES DE LA MOTA		
SUPLENTE:	M EN I. JAIME FRANCISCO GOMEZ VEGA		29/X/96
SUPLENTE:	DR. SERGIO GERARDO DE LOS COBOS SILVA		

APROBACION DEL TRABAJO O DE TESIS POR EL DEPARTAMENTO _____

PROMEDIO EN CREDITOS 12 (doce)

RAG*BJS*jac.

AGRADECIMIENTOS

Al Doctor Ricardo Aceves García.

Por su asesoría brindada durante el desarrollo del presente trabajo, el cual se vio enriquecido con sus consejos y comentarios.

A mis sinodales: **Doctor Jesús Acosta**
M. en I. Idalia Flores
M. en I. Jaime Francisco Gómez
Doctor Sergio de los Cobos.

Por el tiempo dedicado a la revisión de este trabajo así como por los comentarios vertidos para su mejor presentación.

A mis profesores y compañeros de la DEPFI.

Por los conocimientos adquiridos y por las horas gratas en que compartimos esfuerzos.

Al Instituto Mexicano de Comunicaciones.

Por todo el apoyo que recibí.

A la Ing. Silvia Zaleta Mota.

Por su apoyo.

A mis compañeros del Instituto Mexicano de Comunicaciones.

Sobre todo a los que amablemente me alentaron y ayudaron durante mis estudios.

A Salvador Domínguez y Enrique Moreno

Por su ayuda en la mecanografía y presentación de este trabajo.

DEDICATORIAS

A María Eugenia.

Por todo lo que me has dado y por que sabes que si necesitara de alguien, en ti sería en quien yo pensaría.

A Andrea y Brenda.

Por que son un gran anhelo que la vida me hizo realidad.

A mis Padres y mis Hermanos.

Por lo que me han inculcado y por todo lo que hemos compartido.

RESUMEN

Aunque se han desarrollado varios algoritmos finitos para los problemas combinatorios, ninguno de éstos métodos es uniformemente eficiente desde el punto de vista computacional, principalmente cuando aumenta el tamaño del problema. A diferencia de los programas lineales, donde problemas muy grandes se han resuelto en un tiempo razonable, los algoritmos que se aplican a problemas combinatorios generan ciertas dificultades en su comportamiento de resolución.

En la práctica, estos problemas de programación pueden ser tan complejos que los modelos que se construyan para abordarlos no se puedan resolver mediante los algoritmos tradicionales. En este caso se hace necesario aplicar la *heurística* para generar procedimientos que ofrezcan *buenas* soluciones.
Una heurística es una apelación intuitiva a una regla interna o particular para trabajar con un aspecto del problema que se esté resolviendo. En el contexto de la programación matemática, a menudo se emplea la *heurística* en conjunción con estrategias de resolución de problemas más rigurosas o generales, o como caso particular de ellas.

En la actualidad, con los avances tecnológicos en el área de informática se ha propiciado la creación de nuevas propuestas para la solución de problemas de optimización que se resuelven a partir de métodos heurísticos, basados en lo que se ha dado a llamar Inteligencia Artificial.

Dentro de toda una serie de nuevas técnicas basadas en inteligencia artificial, en investigación de operaciones se ha puesto especial atención en cuatro métodos de manipulación para la solución de problemas complejos de decisión: **Algoritmos genéticos, redes neuronales, recocido simulado y búsqueda tabú**. Los dos primeros; algoritmos genéticos y redes neuronales, están inspirados por los principios derivados de las ciencias biológicas; recocido simulado se deriva de las ciencias físicas, principalmente de la termodinámica. La búsqueda tabú se basa en las técnicas de solución de problemas de selección de alternativas.

MÉTODOS HEURÍSTICOS PARA LA SOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN COMBINATORIOS.

I N D I C E

TESIS

COMPLETA

MÉTODOS HEURÍSTICOS PARA LA SOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN COMBINATORIOS.

INTRODUCCIÓN

En el campo teórico, la literatura de investigación de operaciones ofrece una gran variedad de algoritmos y métodos que buscan facilitar la obtención de soluciones óptimas a problemas que presentan determinadas características hipotéticas, lográndose en muchos casos importantes resultados. Sin embargo, en situaciones reales los problemas de optimización a los que hay que enfrentar son de tales dimensiones o complejidad, que en muchos de los casos el planteamiento de un modelo y la estructura de un algoritmo no son suficientes para alcanzar una solución al problema planteado. Esto ocurre cuando los modelos incluyen una gran cantidad de variables, las ecuaciones de que consta no son lineales o es demasiado complejo en el aspecto lógico.

La complejidad de los problemas de programación está determinada por la existencia o no de un algoritmo que lo resuelva en forma satisfactoria. Quizá para una instancia dada, se tenga un algoritmo eficiente, pero lo que interesa es el comportamiento del algoritmo para el peor conjunto de datos posibles, es decir, la instancia más mal comportada. Otro punto interesante es pensar en instancias grandes del problema. En razón de ello es que los problemas se clasifiquen de acuerdo al trabajo que se tiene en resolverlos. Específicamente, se pueden clasificar los problemas en cuatro clases de acuerdo a su grado de dificultad:

- *Problemas irresolubles.* Un problema se dice que es *irresoluble* si no existe (ni existirá) un algoritmo para resolverlo.

- *Problemas intratables.* Un problema se dice *intratable* si no existe un algoritmo polinomial para resolverlo.

- *Problemas **NP*** (que significa no determinístico polinomial). Es la clase de problemas que pueden resolverse en tiempo polinomial si se adivina correctamente la trayectoria computacional que debe seguirse para resolverlo.

- *Problemas Polinomiales (**P**).* Se dice que un problema pertenece a la clase P de problemas que pueden ser resueltos polinomialmente sí existe un algoritmo polinomial (conocido o no) que lo resuelve.

La clasificación anterior ubica la importancia de conocer o descubrir algoritmos polinomiales para el problema que se tenga que resolver. Un problema puede pertenecer a la clase **P** y sin embargo puede ser que no se conozca un algoritmo polinomial que lo resuelva, aunque tal algoritmo exista.

Existe un subconjunto de problemas en NP que son los problemas más difíciles en el siguiente sentido (los problemas en este conjunto son polinomialmente equivalentes): cualquier problema en NP puede reducirse a cualquier problema en el subconjunto difícil, de modo que si se tiene un algoritmo polinomial para un problema en dicho subconjunto, se pueden resolver todos los problemas en NP en tiempo polinomial. Los problemas en este subconjunto se llaman NP-*completos*.

En la práctica, los problemas de optimización que han recibido especial atención por ser ubicados dentro del grupo NP, son los que se enfrentan a situaciones de *explosión combinatoria*. La explosión combinatoria se encuentra en aquellas situaciones en la que las elecciones están compuestas secuencialmente, permitiendo una vasta cantidad de posibilidades

El término *optimización combinatoria* significa que debe buscarse la solución de entre un número finito de alternativas factibles, tales que si todas ellas se enumeran puede encontrarse la óptima. El problema en sí radica en que en la práctica, ese número finito con frecuencia asciende a millones o miles de millones de posibilidades y, por lo tanto aún utilizando un computador de alta velocidad, la enumeración completa llevaría mucho tiempo.

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, así como problemas de 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), y en física (determinación de estados de energía mínimos).

Considérese el siguiente problema de optimización:

$$\text{Minimizar } C(x) : x \in X \subset R^n.$$

La función objetivo puede ser lineal o no lineal, diferenciable o no, y el conjunto **X** es discreto. R^n indica que se encuentra en el espacio vectorial de los números reales

La condición $x \in X$, sintetiza las restricciones sobre el vector x. Esas restricciones pueden incluir desigualdades lineales o no lineales. En muchas aplicaciones, puede especificar condiciones lógicas.

Algunos ejemplos clásicos que entran en este esquema, cuyo planteamiento se especifica en el capítulo I, son:

a) El problema del agente viajero.
b) El problema de asignación cuadrática.
c) El problema de calendarización.
d) El problema de la mochila.
e) El problema de localización.
f) El problema de cobertura.
g) El problema de flujo de redes.

Un problema de optimización combinatorio puede formalizarse como una pareja (X, C), donde X, que se ubica en los números reales **R**, denota al conjunto finito de todas las soluciones factibles y C la función de costo, estableciéndolo como

$$C : X \rightarrow R.$$

En el caso de minimización, el problema consiste en encontrar $i_{opt} \in X$ que satisfaga

$$C(i_{opt}) \leq C(i) \quad \forall\, i \in X$$

A la solución i_{opt} se le llama una *solución globalmente óptima* y $C_{opt} = C(i_{opt})$ denota el costo óptimo, mientras que X_{opt} denota el conjunto de soluciones óptimas.

Por lo que un problema de *optimización combinatoria* se puede establecer como un conjunto, I, de instancias de manera informal, donde una instancia estará formada por los "datos de entrada" y la información suficiente para obtener una solución, mientras que un problema es una colección de instancias del mismo tipo.

Existe un amplio rango de procedimientos heurísticos y optimales para resolver problemas que pueden escribirse en forma polinomial (P). Dichos procedimientos se pueden caracterizar a través de sucesiones de movimientos, los cuales permiten pasar de un punto a otro.

A un movimiento s como un mapeo definido sobre un subconjunto $X(s)$ de X puede denotársele de la siguiente forma:

$$s : X(s) \quad \forall\, X$$

Asociado con $X(s)$ esta el conjunto $S(s)$ el cual consiste de aquellos movimientos $s \in S$ que pueden aplicarse a x, por lo que se tiene la siguiente definición:

Definición 1. *Sea (X, C) una instancia de un problema de optimización combinatoria. Entonces una estructura de vecindades es un mapeo*

$$N : X \ \forall 2^X,$$

que define para cada solución $i \in X$ *un conjunto* $S_i \subset S$ *de soluciones "que son cercanas" a i en algún sentido. El conjunto* S_i *se llama vecindad de la solución i y cada* $j \in S_i$ *se llama un vecino de i. Además, se supone que* $j \in S_i \Leftrightarrow i \in S_j$.

Es decir, se considera que un *movimiento* es una transición de una solución factible a otra solución factible (transformada), el cual se puede describir mediante un conjunto de uno o varios *atributos*. Por ejemplo, un atributo en un intercambio consiste en identificar al par de elementos que cambiaron de posición, mecanismo que utilizan diferentes métodos de optimización clásica como son los de programación lineal, programación entera, programación no lineal, etc.

Sin embargo, para la mayoría de problemas de optimización combinatoria que están contenidos en la clase definida como no polinomial completa (NP-completa) no se conocen, y probablemente no existan, algoritmos polinomiales para resolverlos, de tal manera que si se quiere resolver instancias pequeñas de estos problemas, entonces se pueden emplear algoritmos exponenciales, pero al resolver instancias grandes, los algoritmos resultan imprácticos pues el tiempo de solución crece demasiado cuando el número de alternativas crece.

Entre las técnicas que existen para resolver problemas de optimización combinatoria de manera exacta, se pueden enumerar los siguientes:

- Métodos de planos de corte.
- Métodos de ramificación y acotamiento.
- Métodos de enumeración implícita
- Programación polinomial cero-uno.

La implantación de código por computadora de las técnicas anteriores, permiten obtener algoritmos que resuelven problemas de un número restringido de variables, En la mayoría de los casos problemas combinatorios de a lo más 80 variables.

Cuando se tiene un problema de optimización combinatoria clasificado como NP-completo y se desea obtener una instancia grande del mismo, la única opción que queda es dar una "buena solución" al problema, pero no necesariamente la mejor; por lo que en forma paralela a la búsqueda de la mejor solución se han desarrollado una serie de métodos *heurísticos* para obtener *buenas soluciones* de los problemas de optimización combinatoria.

Los algoritmos heurísticos pueden dividirse en dos categorías: algoritmos hechos para un problema específico y algoritmos generales aplicables a una gran variedad de problemas de optimización combinatoria. La desventaja para los algoritmos de la primera categoría es su limitada aplicabilidad debido a la dependencia del problema, por lo que es deseable tener algoritmos heurísticos generales que obtengan soluciones cercanas a la óptima para la mayoría de problemas de optimización combinatorio.

Con los importantes avances en al área de la computación han aparecido en últimas fechas una serie de nuevas técnicas heurísticas, que retoman los procedimientos básicos de los algoritmos exactos, con la particularidad de que son combinados con técnicas cibernéticas de inteligencia artificial.

Dichas técnicas son:

- Redes neuronales.
- Algoritmos genéticos.
- Recocido simulado.
- Búsqueda tabú.

Estas técnicas se basan en los principios de la *inteligencia artificial* en el sentido de los procedimientos que realizan obedecen a razonamientos lógicos y analógicos inspirados en los razonamientos humanos.

El objetivo que se persigue en el presente trabajo es hacer una caracterización de los problemas de optimización combinatoria y presentar algunas de las técnicas heurísticas que existen para resolverlos.

Se pretende, en primer lugar, cumplir con una inquietud particular por motivar a que se preste una mayor atención en las investigaciones sobre problemas combinatorios y en los nuevos métodos para resolverlos que han surgido en los últimos años. En segundo lugar, realizar una investigación que nos introduzca en el campo de las técnicas de búsqueda heurística basadas en la inteligencia artificial, las cuales se presentan en la actualidad de manera incipiente, pero que representan la piedra angular hacia cual habrán de dirigirse muchas nuevas propuestas metodológicas en investigación de operaciones.

Para alcanzar este objetivo, el presente trabajo se divide en cuatro capítulos temáticos. En el primero, se hace un planteamiento de los problemas combinatorios, en especial los relacionados a programación lineal entera y entera mixta. A continuación se hace una caracterización de los problemas combinatorios y los casos mas difundidos en los textos. Posteriormente se plantean factores que son importantes para la elección de un modelo que habrá de seguirse para resolver un problema y finalmente se establecen los factores que determinan la dimensión de los algoritmos que se utilizan para resolverlos, así como la complejidad computacional que les es intrínseca.

En el segundo capítulo se hace una revisión breve de los principales métodos que se aplican académicamente para resolver problemas de tipo combinatorio, sobre todo los de programación entera y entera mixta. Los métodos que se ven en este capitulo son: *el de planos de corte*, con sus variantes fraccional, entero puro y entero mixto; *el de ramificación y acotamiento*; el de

enumeración implícita; y finalmente, el de *programación polinomial*.

El capítulo tercero trata sobre lo que es la heurística, su aplicación en problemas de tipo combinatorio y la importancia que tiene en la solución de problemas cuya dificultad estriba en la gran cantidad de variables sobre la que se debe buscar la optimalidad.

El cuarto capítulo se presenta un análisis sobre lo que es la programación en cómputo bajo los principios de inteligencia artificial, comparándola con la programación algorítmica convencional. A continuación se hace una descripción de los siguientes métodos heurísticos: redes neuronales, algoritmos genéticos, recocido simulado y búsqueda tabú. A cada uno de ellos se hace una caracterización, sus fundamentos, su aplicación en algún problema de tipo combinatorio, así como los resultados más importantes que se han reportado en publicaciones especializadas.

Por último se presentan las conclusiones generales y perspectivas que se quedan abiertas para profundizar en los temas que trata este trabajo.

CAPITULO I

CARACTERIZACIÓN DE LOS PROBLEMAS COMBINATORIOS

1.1 Introducción.

Un problema de optimización combinatoria es cualquier problema de optimización que tiene un número finito de soluciones factibles.

El término *optimización combinatoria* significa que hay sólo un número finito de alternativas factibles que si todas se enumeran, puede encontrarse la óptima. El problema radica en que en la práctica ese número con frecuencia asciende a millones o miles de millones de posibilidades y por tanto aún para computadoras de alta velocidad sería imposible llegar en un tiempo razonable a la combinación de valores en las variables de un modelo de este tipo que maximice o minimice la función objetivo.

En la actualidad los problemas de optimización combinatoria tienen una abundante aplicación en la vida real, sobre todo en lo que se refiere a la administración eficiente del uso de recursos escasos para incrementar la productividad. Muchos de los problemas operacionales tales como distribución de bienes, planeación de la producción y secuenciación de máquinas son de este tipo. Dentro de los problemas de planeación se incluyen los de inversión de capital, localización de medios y selección de carteras. También problemas de diseño de redes aplicables a redes de telecomunicación y transporte, circuitos o sistemas de producción automática.

Una característica de los problemas de tipo combinatorio radica en que éstos,en la mayoría de los casos no guardan dificultad alguna en su planteamiento. En muchos casos esto llega a ser muy simple. Sin embargo, el conflicto surge, como se ha indicado, cuando el número de variables es muy grande ya que la combinación de éstas en la búsqueda de los valores óptimos crece en muchas de las veces de manera exponencial.

Dentro de los problemas de optimización combinatoria se ubican aquellos cuya solución implica que los valores de las variables y de la función objetivo sean cantidades enteras. Este tipo de problemas, donde una o todas sus restricciones deben presentar valores enteros, son los que con mayor frecuencia aparecen en la vida real; y por tanto los que por su carácter combinatorio ocupan un primer plano en las publicaciones orientadas a la investigación de operaciones.

En razón de ello, los casos de problemas combinatorios que se consideran en este trabajo son del tipo lineal entero en sus variantes, entero mixto, entero puro y combinatorio.

Como primer apartado en este capítulo se hace la definición y planteamiento genérico de los programas lineales *entero, entero mixto y combinatorio*. Posteriormente se plantean los principales problemas de estos tipos en forma descriptiva. Y finalmente, se analizan los diversos factores que determinan la complejidad computacional de los problemas de programación lineal y combinatoria.

1.2 Programación entera-mixta, entera y combinatoria.

A lo largo de este trabajo se asume que la función objetivo y las restricciones de desigualdad son lineales; que una función de minimización es equivalente a maximizar la negativa de la misma función y que una restricción de igualdad puede ser representada por dos desigualdades. También se puede requerir que las variables sean no negativas.

Para el desarrollo de este trabajo los problemas combinatorios que se consideran son del tipo lineal entero en sus variantes entero mixto, entero puro y combinatorio entero[1].

El problema de programación lineal entero mixto (PEM) puede escribirse como

como (PEM) $\quad \text{Max } \{cx + hy : Ax + Gy \leq b,\ x \in Z_+^n,\ y \in R_+^p\}$

donde Z_+^n es el conjunto de vectores enteros no negativos de dimensión **n**, R_+^p es el conjunto de vectores no negativos reales de dimensión p, y $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_p)$ son variables desconocidas.

Una caracterización de este problema esta especificada por los datos (c, h, A, G, b), donde c es un vector n, h es un vector p, A una matriz de m x n , G una matriz m x p y b es un vector m. No se hacen distinciones entre vectores fila y columna a menos que se haga necesario. Este problema se llama mixto en razón de que se presentan ambos tipos de variables reales: entera y continua.

El conjunto $S = \{x \in Z_+^n,\ y \in R_+^p,\ Ax + Gy \leq b,\}$ se denomina *región factible* y un $(x,y) \in S$ es llamado *solución factible*. Una instancia se dice que es factible si $S \neq \emptyset$.

---

[1] Esta distinción está hecha con base en el planteamiento de Neuhauser, G. y Wolsey, L. en *Integer and combinatorial optimization*. Wiley, New York 1988.

La función

$$z = cx + hy$$

es llamada *función objetivo*. Un punto factible (x^0, y^0) para el cual la función objetivo es tan grande como sea posible esto es,

$$cx^0 + hy^0 \geq cx + hy \text{ para todo } (x, y) \in S,$$

es llamado valor óptimo o solución.

Un caso factible de PEM puede no tener una solución óptima. Se dice que un caso es no acotado si para un $\omega \in R^1$ hay un $cx + hy > \omega$. Se utiliza la notación $z = \infty$ para un caso no acotado. Por tanto, cualquier caso de factibilidad de PEM tiene una solución o es no acotado. Este resultado requiere del supuesto de que los datos sean racionales.

De esta forma, resolver un caso de PEM significa producir una solución óptima o demostrar que es no acotado o infactible

El *problema lineal entero puro* (PE) en su forma general se representa como:

$$(PE) \quad \text{Max } \{cx: Ax \leq b, x \in Z_+^n\}$$

y es un caso especial de PEM en el que no hay variables continuas.

En muchos modelos, las variables enteras se utilizan para representar relaciones lógicas, por lo que se limitan a ser iguales a 0 ó 1. Así es como se obtiene el PEM 0-1 (respectivamente PE 0-1) en el cual la condición $x \in Z_+^n$ es reemplazada por $x \in B^n$, donde B^n es el conjunto de vectores binarios de dimensión n.

Puesto que no hay una definición generalmente aprobada de un problema de optimización combinatorio, la mayoría de los problemas así llamados son PE 0-1 que tratan con conjuntos finitos y colecciones de subconjuntos. El siguiente es un problema de optimización combinatorio genérico. Sea $N = \{1, \ldots, n\}$ un conjunto finito y sea $c = (c_1, \ldots, c_n)$ un vector n. Para $F \subseteq N$, se define $c(F) = \sum_{j \in F} c_j$. Supóngase que se tiene una colección de subconjuntos $\mathcal{F}$ de N, el problema de optimización combinatorio es

$$(PC) \quad \text{Max } \{c(F): F \in \mathcal{F}\}$$

La optimización entera y combinatoria se refiere a problemas de maximizar o minimizar una función de varias variables sujetas a ciertas limitaciones de igualdad y desigualdad y a

restricciones enteras sobre algunas o todas las variables. Por el contenido del modelo general, existe una notablemente variedad de problemas puede ser representada por modelos discretos de optimización.

Un uso importante y muy común es el de la *selección binaria* de variables 0 - 1. En este caso se considera un evento que puede o no ocurrir, y se supone que es parte del problema decidir entre estas dos posibilidades. Para modelar con tal dicotomía, se utiliza una variable binaria x donde

$$x = \begin{cases} 1 & \textit{si el evento ocurre} \\ 0 & \textit{si el evento no ocurre} \end{cases}$$

El evento que se considere puede ser cualquier cosa, dependiendo de la situación específica que se este considerado.

1.2.1 Presentación de casos.

Algunos de los problemas de tipo combinatorio más representativos son los que se presentan a continuación

1.2.1.1 El problema de la mochila.

Se Supone que hay n proyectos. El j-ésimo proyecto, $j = 1, \ldots, n$, tiene un costo de a_j y un valor de c_j. Cada proyecto es o no hecho, esto es, no es posible hacer una fracción de alguno de los proyectos. También hay un presupuesto b disponible para encontrar los proyectos. El problema de elegir un subconjunto de proyectos que eleve al máximo la suma de la valores sin que se exceda el límite presupuestario es lo que se conoce como el problema 0-1 de la mochila y puede escribirse como

$$Max\, Z = \sum_{j=1}^{n} c_j x_j$$

sujeto a

$$\sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j = 0 \text{ ó } 1, \qquad j = 1,2,\ldots,n,$$

o bien

$$x_j \geq 0, \text{ entero}, \qquad j = 1,2,\ldots,n,$$

Aquí el j-ésimo suceso corresponde al j-ésimo proyecto. Este problema es llamado problema de la mochila en razón del análisis hecho al problema del excursionista que debe decidir qué poner dentro de una mochila, bajo la limitación del peso o volumen sobre lo que puede llevarse. En general, los problemas de este tipo pueden tener varias limitaciones.

Los modelos tipo mochila se usan para resolver problemas de inversiones, problemas de confiabilidad de redes, de cargo fijo, de selección de proyectos, corte de inventarios, control de presupuestos, etcétera. La versión más popular del problema contiene una sola restricción lineal, pero casi cualquier problema lineal entero y muchos problemas combinatorios se pueden reducir a este. Asimismo, el problema de la mochila se presenta también como un subproblema en varios algoritmos de programación lineal pura y entera.

1.2.1.2 El problema de asignación.

Otro problema clásico es el de asignar personas a trabajos. Se supone que hay n personas y m trabajos, donde $n \geq m$. Cada trabajo debe ser hecho por exactamente una persona; también, cada persona puede hacer, cuando mucho, un solo trabajo. El costo que implica que la persona j haga el trabajo i es c_{ij}. El problema consiste en asignar la gente a los trabajos de tal forma que se minimice el costo total de completar todos los trabajos. Para formular este problema, conocido como el problema de asignación, se introducen variables $x_{ij} = 0 - 1$, donde $i = 1,\ldots,m$ y $j = 1,\ldots,$ n corresponden al ij-ésimo evento de asignar la persona j al trabajo i. Puesto que exactamente una persona debe hacer trabajo i, se tienen las restricciones

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \text{para } i = 1,\ldots,m.$$

Además, puesto que ninguna persona puede hacer más de un de trabajo, se tienen también las restricciones

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad \text{para } j = 1,\ldots,n.$$

Es fácil verificar que si $x \in B^{mn}$ satisface las dos ecuaciones anteriores, y por tanto es posible obtener una solución factible al problema de asignación, cuya función objetivo es.

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

En el problema de asignación los $m + n$ de elementos son particionados en conjuntos de trabajos y gente. Pero en otros modelos de este tipo, no se puede asumir tal partición, como sucede con *el problema de apareamiento*, el cual es una generalización del problema de asignación.

1.2.1.3 Problemas de cobertura de conjuntos, empacado de conjuntos y partición de conjuntos.

Una manera común de definir $\mathcal{F}$ conduce a importantes clases de problemas de optimización combinatoria conocidos como cobertura de conjuntos, empacado de conjuntos, y problemas de partición de conjuntos. Sea M = {1,. . . ,m} un conjunto finito y sea (M_j) para $j \in N = \{1,. . . ,n\}$ una colección determinada de subconjuntos de M. Por ejemplo, la colección podría consistir de todos los subconjuntos de tamaño k, para alguna $k \leq m$. Se dice que $F \subseteq N$ cubre a M si $\cup_{j\in F} M_j = M$. En el PC conocido como *problema de cobertura de conjuntos*, $\mathcal{F}$ = {F: F cubre M}. También se dice que $F \subseteq N$ es un paquete con respecto a M si $M_j \cap M_k = \varnothing$ para todo $j, k \in F$, $j \neq k$. En el PC conocido como el problema de empacado de conjuntos, $\mathcal{F}$ = (F:F es un empaquetado de conjuntos con respecto a M). Si $F \subseteq N$ es de cobertura y de empaque, entonces F se dice que es una partición de M. En el problema de conjuntos cubiertos, c_j es el costo de M_j y se busca el costo mínimo de cobertura; sin embargo en el problema de empaquetado de conjuntos, c_j es el peso o valor de M_j y se busca el empaque con el peso máximo.

Estos problemas son en realidad formulados como PE 0-1, donde A es la matriz m x n de incidencia en el conjunto $\{M_j\}$ para $j \in N$; esto es, para $i \in M$,

$$a_{ij} = \begin{cases} 1 & si\ i \in M_j \\ 0 & si\ i \notin M_j \end{cases} \qquad x_j = \begin{cases} 1 & si\ j \in F \\ 0 & si\ j \notin F \end{cases}$$

Entonces F es una cubierta, empaquetado o partición si y sólo si $x \in B^n$ satisface $Ax \geq 1$, $Ax \leq 1$, $Ax = 1$, respectivamente, donde 1 es un vector m cuyos componentes son iguales a 1.

El problema de empaque es un caso especial PE 0-1 con una matriz A = O-1 (una matriz cuyos elementos son 0 ó 1) y b= 1. Nótece que un problema de asignación con m trabajos y m personas es un problema de conjuntos particionados en que M = (1, . . . , m, m+1, . . ., 2m) y M_j para j = 1, . . ., m2 es un subconjunto de M que consiste en un trabajo y una persona.

Muchos problemas prácticos pueden ser formulados como problemas de conjuntos cubiertos. Una aplicación típica se refiere a la ubicación de instalaciones. En este caso se supone un conjunto

de sitios potenciales N = (1,. . . ,*n*) para la ubicación de ciertas instalaciones. Poner una instalación (que puede ser un centro de abastecimiento) en el lugar j tiene un costo de c_j. También se considera un conjunto de M regiones M = (1,. . . ,*m*) que tienen que ser atendidas. El subconjunto de regiones que pueden atenderse desde una instalación ubicada en *j* es M_j. Entonces el problema de elegir un conjunto de costo mínimo para la ubicación de instalaciones de tal forma que cada región pueda ser atendida desde una de las instalaciones es un problema de conjuntos cubiertos.

Ahora bien, otras relaciones más complejas pueden ser modeladas utilizando variables binarias como en los casos que se presentan a continuación.

Por ejemplo, la relación que ambos o ninguno los sucesos 1 y 2 deban ocurrir esta representada por la igualdad lineal $x_2 - x_1 = 0$ para las variables binarias x_1 y x_2. De manera similar, la relación de que el suceso 2 pueda ocurrir si y solo si el suceso 1 ocurre esta representado por la desigualdad lineal $x_2 - x_1 \leq 0$. Más generalmente, considérese una actividad que puede operarse en cualquier nivel *y*, $0 \leq y \leq u$. Ahora supóngase que la actividad puede emprenderse únicamente si algún suceso representado por la variable binaria *x* ocurre. Esta relación es representada por la desigualdad lineal $y - ux \leq 0$ puesto que $x = 0$ implica que $y = 0$ y $x = 1$, debido a la restricción inicial $y \leq u$.

Considérense ahora dos modelos que usan esta relación.

1.2.1.4 El Problema de ubicación de instalaciones.

Estos problemas, como lo nuestra la ilustración del modelo de conjuntos cubiertos, se refiere a la ubicación de instalaciones en los lugares apropiados, de tal forma que permitan cubrir ciertos clientes económicos al menor costo.

Dado un conjunto N = {1,. . . ,*n*} de ubicación potencial de instalaciones y un conjunto de clientes I = {1,. . . ,*m*}, ubicar una instalación en *j* cuesta c_j para $j \in N$. Este problema es más complicado que el de conjuntos cubiertos, porque cada cliente tiene una demanda para un cierto bien, y el costo total de satisfacer la demanda del cliente *i* desde una instalación en *j* es h_{ij}. El problema de optimización consiste en escoger un subconjunto de ubicación de instalaciones y de clientes atendidos por estas instalaciones que mintmicen los costo totales. En el caso del problema de ubicación de instalaciones *ilimitado*, no hay restricción sobre el número de clientes que una instalación puede servir.

La variable binaria x_j es igual a 1, si una instalación es ubicada en *j* y $x_j = 0$ de otra manera. Además, se introduce la variable continua y_{ij}, que es la fracción de la demanda del cliente *i* que es satisfecha desde la instalación *j*. La condición para que la demanda de cada cliente sea satisfecha esta dada por

$$\sum_{j \in N} y_{ij} = 1 \quad \text{para } i \in I.$$

Además, puesto que el cliente i no puede ser servido desde j a menos que una instalación este ubicada en j, se tiene la restricción

$$y_{ij} - x_j \leq 0 \quad \text{para } i \in I \text{ y } j \in N.$$

Por lo tanto, el problema de ubicación de instalaciones *ilimitado* es el PEM

$$Min \sum_{j \in N} c_j x_j + \sum_{i \in I} \sum_{j \in N} h_{ij} y_{ij}$$

sujeto a las restricciones anteriores y $x \in B^n$, $y \in R^{mn}$.

Puede ser irreal suponer que una instalación puede atender a cualquier número de clientes. Supóngase que una instalación ubicada en j tiene una capacidad de u_j y el i-ésimo cliente tiene una demanda de b_i. Ahora se establece que y_{ij} sea la cantidad de bienes enviados desde la instalación y_j a el cliente i y que h_{ij} sea el costo de embarque por unidad del bien. Para formular el problema de ubicación de instalaciones con capacidad como un PEM, se hace que

$$\sum_{j \in N} y_{ij} = b_i \quad \text{para } i \in I,$$

y

$$\sum_{j \in N} y_{ij} - u_j x_j \leq 0 \quad \text{para } j \in N.$$

1.2.1.5 El problema de flujo de redes con cargo fijo.

Dada una red (ver Figura 1) con un conjunto de nodos V (instalaciones) y un conjunto de arcos **A**. Un arco $e = (i, j)$ que une los puntos desde el nodo i a el nodo j significa que hay una ruta directa desde el nodo i al nodo j. Asociado con cada nodo i, hay una demanda b_j. El nodo i es una demanda, abastecimiento, o punto de tránsito dependiendo sobre si b_j es, respectivamente, positivo, negativo, o cero. Supóngase que la demanda neta es cero, esto es, $\Sigma_{i \in I} b_i = 0$. Cada arco (i, j) tiene una capacidad de flujo u_{ij} y una unidad de costo de flujo h_{ij}.

Sea y_{ij} el flujo sobre el arco (i, j). Un flujo es factible si y solo si satisface

$$y \in R^{|A|}$$

$$y_{ij} \leq u_{ij} \qquad \text{para } (i, j) \in \mathbf{A}.$$

$$\sum_{j \in N} y_{ji} - \sum_{j \in V} y_{ij} = b_i \quad \text{para } j \in \mathrm{V}.$$

Esta última corresponde a las restricciones de conservación de flujo. El problema

$$Min \sum_{i,j \in A} h_{ij} y_{ij}$$

sujeto a

$$y \in \mathrm{R}^{|A|}$$

$$y_{ij} \leq u_{ij} \qquad \text{para } (i, j) \in \mathbf{A}.$$

$$\sum_{j \in N} y_{ji} - \sum_{j \in V} y_{ij} = b_i \quad \text{para } j \in \mathrm{N}.$$

es conocido como el *problema de flujo de redes*.

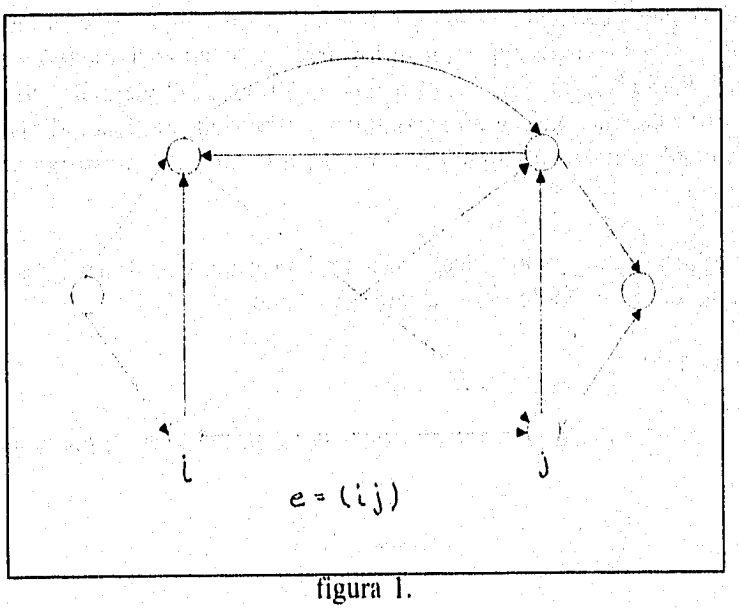

figura 1.

El problema de flujo de redes con cargo fijo se obtiene al asignar un costo fijo de c_{ij} si hay un flujo positivo sobre el arco (i, j). Ahora se introduce una variable binaria x_{ij} para indicar que

se sigue el arco (i,j). La restricción $y_{ij} = 0$ si $x_{ij} = 0$ está representada por

$$y_{ij} - u_{ij}x_{ij} \leq 0 \qquad \text{para } (i, j) \in \mathbf{A}.$$

por tanto se obtiene la fórmula

$$Min \sum_{i,j \in A} c_{ij}x_{ij} + h_{ij}y_{ij}$$

sujeto a

$$\sum_{j \in N} y_{ji} - \sum_{j \in V} y_{ij} = b_i \quad \text{para } j \in \mathrm{V},$$

$$y_{ij} - u_{ij}x_{ij} \leq 0 \qquad \text{para } (i, j) \in \mathbf{A}.$$

El modelo de flujo con cargo fijo es útil para una variedad de problemas de diseño que involucran flujos materiales en redes. Estos incluyen sistemas de abastecimiento de agua, sistemas de calefacción y redes de camino.

1.2.1.6 El Problema del vendedor viajero

Considérese un conjunto de nodos $\mathrm{V} = \{1, \ldots, n\}$ y un conjunto de arcos $\mathbf{A}$. Los nodos representan ciudades, y los arcos representan los pares ordenados de ciudades entre los cuales es posible viajar. Para $(i, j) \in \mathbf{A}$, c_{ij} es el tiempo que implica el viaje directo desde la ciudad i a la ciudad j. El problema consiste en encontrar un recorrido, partiendo de la ciudad 1, que: (a) visite cada ciudad exactamente una vez y vuelva a la ciudad 1 y (b) que el viaje tome el menor tiempo total.

Para formular este problema, se introducen variables en las que $x_{ij} = 1$ si j sigue inmediatamente de i sobre el recorrido, $x_{ij} = 0$ de otra manera. Por tanto

$$x \in \mathrm{B}^{|A|}$$

El requisito de que cada ciudad se visite en exactamente una vez se afirma como

$$\sum_{i(i,j) \in A} x_{ij} = 1 \quad , \qquad \text{para } j \in \mathrm{V},$$

y

$$\sum_{j(i,j) \in A} x_{ij} = 1 \quad , \qquad \text{para } i \in \mathrm{V}.$$

Estas restricciones no son suficientes para definir los recorridos, puesto que hay que satisfacer los subrecorridos también; por ejemplo para $n = 6$, $x_{12} = x_{23} = x_{31} = x_{45} = x_{56} = x_{64} = 1$, satisface dichas restricciones, pero no corresponden a un recorrido (ver Figura 2).

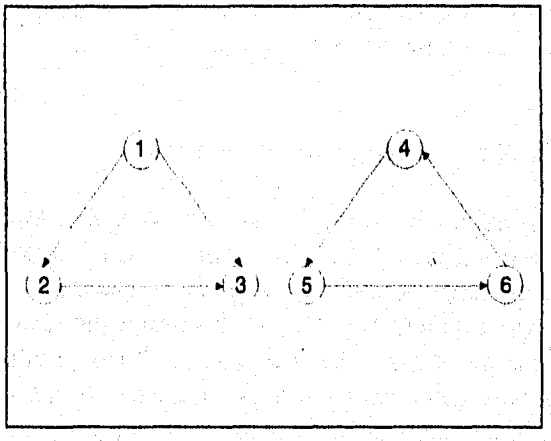

figura 2.

Una de las formas de eliminar subrecorridos es hacer que en cualquier recorrido haya un arco que pase de {1,2,3} a {4,5, 6} y un arco que pase de {4,5,6} a {1,2,3}. En general, para cualquier U ⊂ V con $2 \leq |U| \leq |V| - 2$, las restricciones

$$\sum_{(i,j)\in A: i\in U, j\in V\setminus U} x_{ij} \geq 1$$

son satisfechas por todos de los recorridos, pero cada subrecorrido infringe por lo menos a uno de ellos. Por lo tanto el problema del agente viajero puede formularse como

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

en que x satisface

$$x \in B^{|A|}$$

y

$$\sum_{(i,j)\in A: i\in U, j\in V\setminus U} x_{ij} \geq 1$$

Una alternativa al este último conjunto de restricciones es

$$\sum_{(i,j)\in A:i\in U,j\in U} x_{ij} \leq |U| - 1 \quad \text{para } 2 \leq |U| \leq |V| - 2$$

que también excluye todos los subrecorridos pero no los recorridos. Sin embargo, aplicando estas relaciones, el número de esas restricciones es aproximadamente 2^V. Este enorme número de restricciones es motivo para buscar una fórmula más compacta.

1.3 Problemas no lineales y restricciones disyuntivas.

Ahora se presentan dos aplicaciones importantes de variables binarias en problemas de optimización. La primera se refiere a la representación de las funciones objetivo no lineales de la forma $\Sigma_j f_j(y_j)$, utilizando funciones lineales y variables binarias. La segunda se refiere a modelos con restricciones disyuntivas. En el planteamiento usual de un problema de optimización, se presupone que todas las restricciones se deben de satisfacer. Sin embargo en algunas aplicaciones, solo una de un par (o, más generalmente, k de m) de restricciones se debe satisfacer. En este caso, se dice que las restricciones son disyuntivas.

1.3.1 Funciones lineales separables.

Una función de la forma $f(y_1, \ldots, y_p) = \sum_{j=1}^{p} f_j(y_j)$ se dice que es una función separable.

Aquí se considera la función objetivo separable y se supone que $f_j(y_j)$ es lineal por partes para cada j (ver Figura 3). Se observa que una función continua arbitraria de una variable puede ser aproximada por una función lineal por partes, que representa una aproximación controlada por el tamaño de los segmentos lineales.

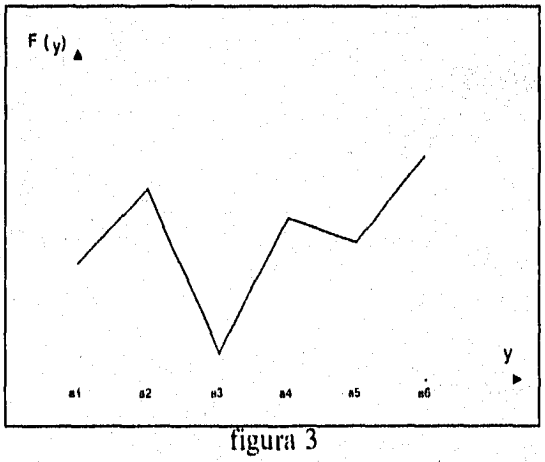

figura 3

Suponiendo que se tiene una función lineal por partes $f(y)$ especificada por los puntos $\{a_i, f(a_i)\}$

para i = 1,. . . ,r. Entonces, cualquier $a_1 \leq y \leq a_r$, puede escribirse como.

$$y = \sum_{i=1}^{r} \lambda_i a_i \ , \ \sum_{i=1}^{r} \lambda_i = 1 \ , \ \lambda = (\lambda_1,. \ . \ . \ ,\lambda_r) \in R^r.$$

Los λ_i no son únicos, pero si $a_i \leq y \leq a_{i+1}$ y λ se elige de modo que $y = \lambda_i a_i + \lambda_{i+1} a_{i+1}$ y $\lambda_i + \lambda_{i+1} = 1$, se obtiene $f(y) = \lambda_i f(a_i) + \lambda_{i+1} f(a_{i+1})$. En otras palabras

$$f(y) = \sum_{i=1}^{r} \lambda_i f(a_i) \ , \ \sum_{i=1}^{r} \lambda_i = 1 \ , \ \lambda \in R^r.$$

si más de dos de los λ_i's son positivos y si λ_j y λ_k son positivos, entonces $k = j - 1$ ó $j + 1$. Esta condición puede modelarse utilizando variables binarias x_i para $i = 1,. \ . \ . \ ,r - 1$ (donde $x_i = 1$ si $a_i \leq y \leq a_{i+1}$ y $x_i = 0$ de otra manera) y las restricciones

$$\lambda_1 \leq x_1$$

$$\lambda_i \leq x_{i-1} + x_i \qquad \text{para } i = 2,. \ . \ . \ ,r - 1$$

$$\lambda_r \leq x_{r-1}$$

$$\sum_{i=1}^{r-1} x_i = 1$$

$$x \in B^{r-1}$$

se observa que si $x_j = 1$, entonces $\lambda_i = 0$ para $i \neq j$ ò $j + 1$.

Las funciones lineales por partes que son convexas (cóncavas) pueden minimizarse (maximizarse) mediante programación lineal, ya que la pendiente de los segmentos son crecientes (decrecientes) (ver Figura 4). Pero en general las funciones lineales por partes no son ni cóncavas ni convexas, por tanto las variables binarias son necesarias para seleccionar el segmento correcto para un valor determinado de y.

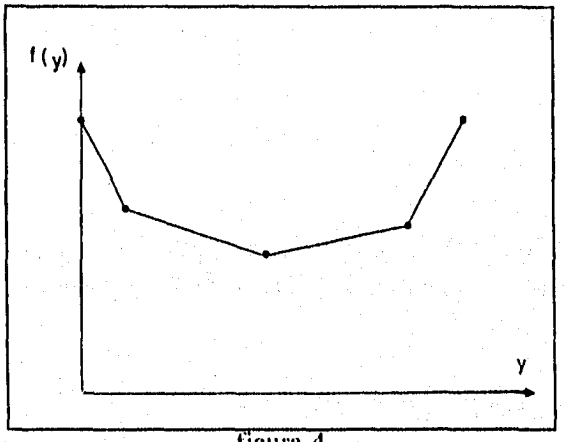

figura 4

1.3.2 Problemas con restricciones disyuntivas.

Las restricciones disyuntivas provienen naturalmente en muchos modelos. Una ilustración simple es cuando se requiere definir una variable igual al valor mínimo de dos variables, esto es, $y = \min(u_1, u_2)$. Esto puede hacerse con las dos desigualdades

$$y \leq u_1 \text{ e } y \leq u_2$$

junto con una de dos desigualdades

$$y \geq u_1 \text{ o } y \geq u_2$$

Un conjunto disyuntivo típico de restricciones afirma que un punto debe satisfacer por lo menos k de m de conjuntos de restricciones lineales. El caso de $k = 1$, $m = 2$ se muestra en la Figura 5, donde la región factible esta sombreada.

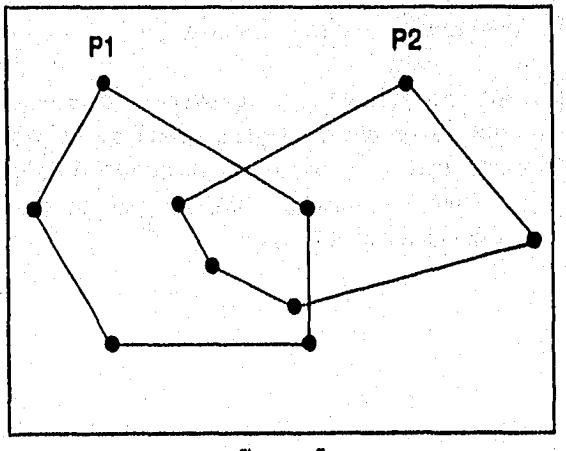

figura 5

Suponiendo $P^i = \{y \in R^p: A^i y \leq b^i, y \leq d\}$ para $i = 1,\ldots,m$, se observa que hay un vector ω tal que, para todo i, $A^i y \leq b^i + \omega$ es satisfecho por cualquier y, $0 \leq y \leq d$. Por tanto hay un y contenido en por lo menos k de los conjuntos P^i si y solo si el conjunto

$$A^i y \leq b^i + \omega(1-x_i) \quad \text{para } i = 1,\ldots,m$$

$$\sum_{i=1}^{m} x_i \geq k$$

$$y \leq d$$

$$x \in M^m,\ y \in R^p$$

es no vacío. Esto sigue desde $x_i = 1$ dadas las restricciones $A^i y \leq b^i$ mientras $x_i = 0$ dadas las restricciones redundantes $A^i y \leq b^i + \omega$.

Cuando $k = 1$, una formulación alternativa es

$$A^i y^i \leq x_i b^i \quad \text{para } i = 1,\ldots,m$$

$$y^i \leq x_i d \quad \text{para } i = 1,\ldots,m$$

$$\sum_{i=1}^{m} x_i = 1$$

$$\sum_{i=1}^{m} y^i = y$$

$$x \in B^m,\ y \in R^p,\ y^i \in R^p \text{ para } i = 1,\ldots,m.$$

Ahora se puede afirmar que $\bigcup_{i=1}^{m} P^i \neq \varnothing$ si y solo si la formulación anterior es no vacía.

Primero, dado que $y \in \bigcup_{i=1}^{m} P^i$, se supone, sin pérdida de generalidad, que esa $y \in P^1$.

Entonces una solución de la formulación anterior es $x_1 = 1$, $x_i = 0$ en otro caso, $y^1 = y$, y $y^i = 0$ de otro caso. Por otra parte, suponer que dicha formulación tiene solución y, sin pérdida de generalidad, se supone que $x_1 = 1$ y $x_i = 0$ en otro caso. Entonces se obtiene $y^i = 0$ para

$i = 2,. . . ,m$ e $y = y^l$. Así $y \in P^l$ y $U_{i=1}^{m} P^i \neq \emptyset$.

Los modelos anteriores presentan una formulación bastante diferente sobre el mismo problema. Sin embargo esta elección de fórmulas es típica.

Las restricciones disyuntivas son comunes de encontrar en problemas de asignación donde varios trabajos tienen que ser procesados en una máquina y donde el orden en que estos se deben de procesar no esta especificado. Por tanto se obtienen restricciones disyuntivas en las que el trabajo k precede al trabajo j en la maquina i y viceversa.

Se supone que hay n trabajos y m máquinas y cada trabajo debe procesarse en cada máquina. Para cada trabajo, el orden de la máquina es fijo, esto es, el trabajo j debe primero ser procesado en la maquina $j(1)$ y luego en la máquina $j(2)$, y así sucesivamente. Una máquina solo puede procesar un trabajo a la vez, y una vez que un trabajo se inicia sobre cualquier máquina este debe ser procesado hasta completarse. El objetivo es minimizar la suma de los tiempos que lleva realizar todos los trabajos. Los datos que especifican una instancia del problema son: *(a)* m, n, y p_{ij} para $j = 1,. . . ,n$ e $i = 1,. . . ,m$, que es el procesamiento del tiempo del trabajo j en la máquina i; y (b) el orden de las máquinas, $j(1),. . . j(m)$, para cada trabajo j.

Sea t_{ij} el tiempo inicial del trabajo j en la maquina i. Puesto que las $(r + 1)$ operaciones en el trabajo j no pueden iniciarse sino hasta que la r-ésima operación ha sido completada, se tienen las restricciones

$$t_{j(r+1),j} \geq t_{j(r),j} + p_{j(r),j} \quad \text{para } r = 1,. . . ,m - 1 \text{ para toda } j.$$

Para representar las restricciones disyuntivas para los trabajos j y k en la máquina i, sea $x_{ijk} = 1$ si el trabajo i precede al trabajo k en la maquina i y $x_{ijk} = 0$ en otro caso, donde $j < k$. *Así*

$$t_{ik} \geq t_{ij} + p_{ij} \qquad \text{si } x_{ijk} = 1$$

y

$$t_{ij} \geq t_{ik} + p_{ik} \qquad \text{si } x_{ijk} = 0$$

Dada una cota superior ω en $t_{ij} - t_{ik} + p_{ij}$ para todo i, j, y k se obtienen las restricciones disyuntivas

$$t_{ij} - t_{ik} \leq -p_{ij} + \omega(1-x_{ijk})$$

$$t_{ik} - t_{ij} \leq -p_{ik} + \omega x_{ijk} \quad \text{para toda } i,j \text{ y } k.$$

Puesto que el problema es minimizar $\sum_{j=1}^{n} t_{j(m),j}$ sujeto a

$$t_{j(r+1),j} \geq t_{j(r),j} \text{ para } r = 1, \ldots, m-1 \text{ para toda } j.$$

$$t_{ij} - t_{ik} \leq -p_{ij} + \omega(1-x_{ijk}).$$

$$t_{ik} - t_{ij} \leq -p_{ik} + \omega x_{ijk} \text{ para toda } i,j \text{ y } k.$$

con $t_{ij} \geq 0$ para toda i y j y $x_{ijk} \in \{0,1\}$ para toda i,j y k.

Este modelo requiere $m\binom{n}{2}$ variables binarias. En contraste con los modelos de programación entera introducidos previamente, este modelo no ha sido resuelto para valores de m y n que son de interés práctico. Esta formulación es parcialmente engorrosa, debido al gran número de variables binarias necesarias para representar el gran número de disyunciones.

La solución de este tipo de problemas tiene que ver mucho con la forma en que se estructure el modelo del correspondiente. Por tanto, una buena formulación es esencial para resolver eficientemente los problemas de programación entera, como se explica a continuación.

1.4 La elección en la formulación del modelo.

Aunque el planteamiento del modelo puede hacerse conociendo la estructura del problema, el objetivo principal es resolver el problema obteniendo la solución óptima o una solución aproximada al óptimo. La mayor parte de los problemas de programación entera pueden formularse de varias maneras[2]. Sin embargo en contraste con la programación lineal, en programación entera la formulación de un buen modelo es fundamental para resolver el problema. De hecho la buena formulación del modelo se relaciona estrechamente con el algoritmo que habrá de aplicarse.

Un modelo esta determinado por sus variables, su función objetivo y las restricciones. Normalmente la definición de las variables representa el primer paso directo en la formulación del modelo. Frecuentemente las variables se eligen simplemente para la definición de una solución. Esta es una solución que especifica los valores de ciertas incógnitas y se define una variable por cada incógnita. Una vez que las variables y una función objetivo han sido definidas en un problema entero, puede hacerse una representación implícita del problema. Por ejemplo, se tiene

---

[2] Para una mayor profundización en el planteamiento de programas enteros puede verse a TAHA, M. *Integer programing: Teory application and computations*. Academic Press, Orlando, Florida. 1975.

$$\text{Max } \{cx:\ x \in S \subset Z_+^n\},$$

donde S representa el conjunto de puntos factibles en Z_+^n .

Ahora se puede decir que

$$\text{Max } \{cx: Ax \leq b,\ x \in Z_+^n\},$$

es una formulación **PE** valida si $S = \{x \in Z_+^n : Ax \leq b\}$,

En general cuando hay una formulación valida, hay muchas elecciones de (A, b), y es normalmente fácil encontrar algún (A, b) que proporciona una. Pero una selección obvia puede no ser buena cuando se considera solución del problema, por lo que el aspecto más importante de la formulación del modelo es la selección de (A, b).

La mayoría de los algoritmos de programación entera requieren de un límite superior en el valor de la función objetivo, y la eficiencia del algoritmo depende mucho de la agudeza de el límite. Un límite superior esta determinado por la solución del programa lineal.

$$Z_{PL} = \{\text{Max } cx: Ax \leq b,\ x \in R_+^n\},$$

puesto que $P = \{x \in R_+^n ; Ax \leq b\} \supseteq S$. Ahora dadas dos formulaciones válidas, definidas por (A^i, b^i) para $i = 1, 2$, sea $P^i = \{x \in R_+^n : A^i x \leq b^i\}$ y $Z_{PL}^i = \text{Max } \{cx: x \in P^i\}$. Se observa que si $P^1 \subseteq P^2$, entonces $Z_{PL}^1 \leq Z_{PL}^2$. Por tanto, se consigue el mejor límite de la formulación basada en (A^1, b^1) y se dice que esta es la mejor formulación.

Un ejemplo de que una formulación sea mejor que otra, en el sentido descrito, se obtiene en el problema de ubicación de instalaciones ilimitado donde una formulación con menos restricciones se consigue reemplazando $y_{ij} - x_j \leq 0$ para $i \in I$ y $j \in N$ por

$$\sum_{i \in I} y_{ij} - mx_j \leq 0 \qquad \text{para todo } j \in N.$$

En este caso, cuando $x_j = 0$ se dice que ningún cliente puede ser atendido por la instalación j;

y cuando $x_j = 1$ no hay restricción en el número de clientes que puede ser atendido por la instalación j.

Se insiste en este punto por que se debe tomar en cuenta que el tiempo de procesamiento se incrementa y la factibilidad computacional decrece cuando el número de restricciones se incrementa. Pero tratar de encontrar una formulación con un número pequeño de restricciones es casi siempre una estrategia muy mala. En efecto, uno de los principales algoritmos existentes implica la adición sistemática de restricciones. Dicho algoritmo es el de planos de corte[3].

El énfasis en la selección de restricciones para la obtención de una buena formulación, suponiendo que las variables ya han sido definidas, es importante puesto que para la mayoría de los problemas esta es la parte de la formulación donde hay la mayor libertad de elección. Hay, sin embrago, problemas en los cuales la calidad de la formulación depende de la selección de variables.

Por ejemplo, en la formulación de problemas de flujos de redes, se definen las variables que son los flujos de los arcos. En ciertas situaciones es más ventajoso definir las variables que representan los flujos en cada dirección, entre dos nodos. Tal formulación involucra mucho más variables pero elimina la necesidad de algunas restricciones de conservación de flujos y puede ser preferible para encontrar soluciones integrales.

1.4.1 Preprocesamiento

Cuando se desarrolla un algoritmo en algún programa de cómputo, muchas veces es necesario realizar algunas condiciones previas. En este caso, el *preprocesamiento* se refiere a las operaciones elementales que pueden ejecutarse para mejorar o simplificar la formulación de un problema estrechando los límites de las variables, los valores fijos y así sucesivamente. El preprocesamiento pude pensarse como una fase entre la formulación y la solución. Este puede mejorar la velocidad de un algoritmo sofisticado que puede, por ejemplo, ser incapaz de reconocer el hecho de que alguna variable pueda ser fijada y luego eliminada del modelo. Ocasionalmente un problema pequeño puede resolverse en la fase de preprocesamiento o combinando preprocesamiento con enumeración. Aunque este enfoque ha sido considerado como una solución técnica en el desarrollo prematuro de la programación entera, bajo el nombre de enumeración implícita, este no es papel importante de estas técnicas simples. Su principal propuesta es preparar una rápida formulación y automatizarla por un algoritmo más sofisticado. Desafortunadamente, esto ha llevado mucho tiempo para los investigadores reconocer el hecho de que hay generalmente una necesidad por ambas fases en la solución de problemas prácticos.

---

[3] Dicho algoritmo se presenta en el capítulo II de este trabajo.

1.4.2 Estrechamiento de Cotas

Una restricción común en los **PEM** es $y_j \leq u_j x_j$, donde u_j es una cota superior en y_j y x_j es una variable binaria. Puesto que $x_j \in \{0,1\}$, la reducción de los límites superiores no esta mal. Pero si se reemplaza $xj \in \{0,1\}$ por $0 \leq x_j \leq 1$, llega a ser importante una reducción de la cota. Supóngase, por ejemplo, que el mayor valor factible de y_i es $u'_j < u_j$ y que hay un costo $c_i > 0$ asociado con x_j. Si $y_j = u'_j$ en una solución óptima y se utiliza la restricción $y_j \leq u_j x_j$, puede obtenerse $x_j = u'_j/u_j < 1$. Por otra parte, si se utiliza la restricción $y_j \leq u'_j x_j$, se obtiene $x_j = 1$.

En algunos casos, las buenas cotas pueden ser determinadas analíticamente. Por ejemplo, en el problema por lotes, más que usar una cota común para cada y_t, es más eficiente usar las cotas $y_t \leq \left(\sum_{t=1}^{T} d_t \right) x_t$. En general las cotas estrechas pueden ser determinadas resolviendo un programa lineal con el objetivo de maximizar y_t. Haciendo esto para cada una de las variables con una restricción con cota superior puede ser prohibitivo, en cuanto al tiempo consumido en su procesamiento, por lo que una buena táctica es aproximarse a la cota superior heurísticamente.

1.5 Complejidad de los problemas.

En investigación de operaciones el grado de complejidad de un problema tiene que ver con el tiempo y posibilidad de alcanzar la solución óptima. Como se observa en la amplia literatura existente, se han desarrollado muchos procedimientos para encontrarlas en ciertos tipos de problemas, pero es necesario reconocer que estas soluciones son óptimas sólo respecto al modelo que se este utilizando. Como el modelo necesariamente es una idealización y no una representación del problema real, no puede existir una garantía de que la solución óptima del modelo resulte ser la mejor solución posible que pueda llevarse a la práctica para el problema real. Esto es cierto si se toman en cuenta los muchos imponderables e incertidumbres asociadas a casi todos los problemas reales, pero si el modelo está bien formulado y verificado, la solución que resulta debe tener una buena aproximación del curso de acción ideal para el problema real.

Una clasificación de los problemas de optimización basada en su grado de dificultad establece las siguientes clases de problemas[4]:

Problemas Indecidibles. Son aquellos problemas para los cuales no se puede escribir un algoritmo.

Problemas intratables. Son aquellos problemas para los cuales no se pueden desarrollar algoritmos polinomiales. Este caso de problemas se demuestra son difíciles y sólo se pueden

---

[4] Flores de la Mota, Idalia. *Apuntes de programación entera.* Facultad de Ingeniería, UNAM. México.

resolver con algoritmos exponenciales, para los cuales el tiempo requerido para su solución solo puede ser acotado por arriba mediante una función exponencial del tamaño del problema.

Problemas no polinomiales determinísticos ***(NP).*** Esta clase incluye problemas que se pueden resolver en tiempo polinomial si se logra adivinar correctamente que ruta computacional se puede seguir. Esta clase incluye todos los problemas que tienen algoritmos exponenciales pero que no se ha probado que se puedan resolver con algoritmos de tiempo polinomial.

Problemas Polinomiales **(P).** 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. Se entiende por tamaño del problema la longitud de un código, por ejemplo binario, de los datos del problema. Esta clase incluye todos los problemas que tienen algoritmos de tiempo polinomial.

Por ejemplo, una gráfica G = [X,A] con N = X nodos y M = A arcos, una ruta más corta se encuentra a lo más en un tiempo $O(N^3)$, un árbol de peso mínimo en un tiempo $O(N^2)$, sin embargo no todos los problemas de tipo combinatorios son polinomiales.

Problemas no polinomiales completos **(NP-completos).** Estos son un subconjunto de problemas **NP** que son los más difíciles en el sentido de que cualquier problema **NP** se puede reducir a cualquier problema de este subconjunto y transformarlo para poder resolverlos en tiempo polinomial. Por ejemplo, problemas como el del agente viajero o el de la mochila se conocen como problemas no polinomiales completos o NP-completos.

1.6 Complejidad computacional.

En los últimos años muchos autores sobre investigación de operaciones se han dedicado a la tarea de describir una teoría que establezca los factores que determinan la complejidad computacional de los problemas de optimización combinatoria, en donde se definan el porque y cuan difícil puede ser resolver un problema de este tipo.

Una práctica común en programación lineal consiste en relacionar el tiempo de procesamiento de cómputo con el tamaño de problema. Tradicionalmente, el tamaño de un problema de optimización ha sido descrito por su número de variables y de restricciones. Estos dos parámetros, sin embargo, no pueden ser adecuados.

Un programa de computadora consiste en una serie de instrucciones dictadas en forma sistemática y ordenada, orientadas a la realización de una serie de operaciones lógicas, aritméticas y de decisión, cuyo fin es una salida que corresponda al procesamiento que la computadora hace de dichas instrucciones a partir de ciertos datos de entrada expresados en forma correcta

Para que una computadora resuelva un problema es preciso indicarle qué operaciones debe

realizar. Es decir, se le debe de construir el procedimiento para resolver el problema. Dicho procedimiento se llama algoritmo.

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

Ahora bien, una serie de preguntas obligadas sobre este tema serían las siguientes: Si existen varios algoritmos para resolver un problema ¿cuál de ellos es el "mejor" en el sentido de que necesite menos recursos informáticos? ¿Cuáles son los mínimos recursos informáticos necesarios para llevar a cabo un proceso determinado? (es decir, qué recursos utilizará el mejor algoritmo posible para realizar dicho proceso?) ¿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?

Hay algoritmos cuyo número de pasos depende explícitamente de la magnitud de los datos numéricos. Por ejemplo, hay un algoritmo para el problema entero de la mochila cuyo número de procesos de cómputo es proporcional al número de variables que están del lado derecho de la restricción. En el método elipsoidal para programación lineal, el número de procesos de cómputo depende de el volumen de la elipsoide inicial, que a la vez depende de la magnitud de los datos numéricos. También, el valor de los números implicados en los cálculos elementales, tales como sumas y multiplicaciones, puede incidir en el tiempo de procesamiento. Además, es frecuentemente razonable suponer que estas operaciones son hechas en una constante unidad de tiempo.

Una vez que se ha establecido un modelo que describe un problema, los datos de sus instancias, es posible considerar el tiempo de cómputo. Si se requiere que la medida de tiempo de procesamiento sea independiente de las características particulares de cierta computadoras, simplemente se cuenta el número de operaciones elementales tales como adiciones, multiplicaciones, comparaciones, etcétera; esto es, se supone que cada operación elemental se hace en tiempo individual. Este es un supuesto razonable mientras el tamaño de los números no crezca demasiado rápido conforme los cálculos progresan.

En este caso, tomando como base la programación entera-mixta es posible considerar las incidencias en torno a complejidad computacional relativas a los programas combinatorios

Como se mencionó anteriormente, la programación entera-mixta se refiere al problema genéricamente escrito como

$$\text{(PEM)} \quad \text{Max } \{cx + hy \; Ax + Gy \leq b,\; x \in Z_+^n \;,\; y \in R_+^p \;\}$$

donde m es cualquier entero positivo, p y n son enteros no negativos con $p + n \geq 1$, y c, h, A, G, b son matrices con coeficientes enteros; las dimensiones de estas matrices son como sigue: c es 1 x n, h es 1 x p, A es m x n, G es m x p, y b es m x 1. Es importante suponer que estas matrices tienen coeficientes racionales, pero el supuesto de coeficientes enteros no es menos general y es más conveniente.

Un problema consiste de un número infinito de instancias. Una instancia se refiere a los valores numéricos que se asigna a los parámetros de problema. En el caso del programa entero-mixto, los datos que especifican una instancia son los enteros m, n, y p así como también c, h, A, G, y b son matrices enteras de dimensión apropiada.

Cuando se habla de facilitar el análisis es deseable delinear los casos especiales de problemas de programación entero-mixto. Esto se hace principalmente para restringir los parámetros de manera natural. Por tanto se puede establecer que el problema de programación entero puro es un caso especial de programa entero mixto en el que $p = 0$, y las matrices h y G no se consideran. La programación lineal también es un caso especial de ésta, en que $n = 0$, y las matrices c y A no se consideran.

Cada instancia de un programa lineal entero puro es también un caso de un programa entero-mixto. Por tanto un algoritmo que puede resolver todas las instancias de los programas enteros mixtos pueden, por definición, resolver todas las instancias de casos especiales de programación lineal entera pura. Una conclusión obvia es que la programación entera mixta es al menos tan difícil como la programación lineal entera pura.

Sin embargo, intentar resolver cualquier problema de programación lineal a través de un algoritmo genérico propuesto para los programas enteros-mixtos, llevaría hacia procesos demasiado largos que pueden evitarse aplicando una técnica adecuada para cada problema.

Por otra parte, hay tantos casos especiales de programación entera-mixta, que sería imposible considerarlos de forma separada, por lo que el principal punto consistiría en encontrar semejanzas naturales entre los diversos casos y establecer el algoritmo que lleve a la solución genérica de problemas de tipo combinatorio. Este es un punto medular en lo que a solución de programas y complejidad computacional se refiere.

1.6.1 Factores que determinan la complejidad computacional de los algoritmos de tipo combinatorio.

Un programa es la representación de un algoritmo en un lenguaje de programación que se puede interpretar y ejecutar por una computadora. Ahora bien ¿Hasta que punto la complejidad computacional de un programa que resuelva un problema combinatorio depende de la característica del algoritmo que se emplea?

Una posible alternativa para contestar dicha cuestión consiste en representar dos algoritmos que

resuelven un mismo problema mediante un lenguaje de programación, ejecutarlos en una computadora y medir el tiempo que lleva a cada uno de ellos obtener la solución del problema.

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

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. Por tanto, se supone desde ahora que todas las operaciones básicas se ejecutan en una misma 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 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.

El tiempo requerido por un algoritmo para su ejecución, depende fundamentalmente de los siguientes factores:

1. El lenguaje de programación elegido

2. El programa que representa el algoritmo

3. La computadora que lo ejecuta

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

1.6.2 El peor caso y caso probabilístico

Cuando se hacen comparaciones sobre el tiempo de ejecución de los algoritmos, aquel que ocupa el mayor posible de entre todos los casos que se pueden presentar se le llama *el peor caso.*

En ocasiones no solo se considera el peor caso, sino también se tiene interés en el 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 siendo de gran interés en la evaluación

de los algoritmos misma que resulta en ocasiones sumamente complejo.

Este análisis se encarga del estudio del *tiempo* y *espacio* requerido por un algoritmo para su ejecución . Ambos parámetros 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 peor caso, pues el análisis se hace intratable matemáticamente. Así pues, se utilizará el tiempo de ejecución del peor caso como medida principal de 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 $\mathbf{T}(n)$ y $\mathbf{E}(n)$ 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 contante conocido.

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

1.6.3 Tiempo de ejecución de un programa

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 $O(\max [f(n), g(n)])$.
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:

$$T_1(n) + T_2(n) \leq (c_1 + c_2) \max[f(n), g(n)]$$

por lo tanto $T_1(n) + T_2(n)$ es O(max $[f(n), g(n)]$).

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 incommensurables (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:

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

En tales casos, la regla de la suma debe aplicarse directamente, en el ejemplo, el tiempo de ejecución es O(max $[f(n), g(n)]$), 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($f(n)g(n)$). 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).

1.6.4 Análisis asintótico de funciones

Un tipo de estudio importante consiste en comparar funciones reales de variable natural f: N → R y decir cuál tiene mejor comportamiento asintótico, es decir, cuál es menor cuando la variable independiente es suficientemente grande.

Si se sabe hacer esto con dos funciones se puede utilizar las funciones de tiempo de dos algoritmos y determinar cuál de ellos tiene mejor comportamiento asintótico.

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

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 ≥ 0 y m ≥ 0 tales que se verifica la desigualdad: $| f(n) | \leq k | g(n) |$ para todo entero $n \geq m$.

Se observa 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 esta situación, si f y g son funciones de tiempo de dos algoritmos F y G respectivamente, resulta que el algoritmo f nunca tardará más 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, $|n| \geq |n^3|$ 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.

Ahora bien, 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 de orden *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*.

1.6.5 Velocidad de crecimiento.

Partiendo 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 menor 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 $5n^2$ 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 se procesen los programas. Para entradas de tamaño $N < 20$, el programa con tiempo de ejecución $5n^3$ será más 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 de $5n^3/100n^2 = n/20$, 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.

1.7 Conclusiones.

Un problema combinatorio es aquel que asigna valores numéricos discretos a algún conjunto de restricciones bajo las cuales se pretende minimizar (maximizar) alguna función objetivo. Dentro de este tipo de problemas se puede ubicar a los problemas de asignación de trabajos, ubicación de plantas, secuenciación de rutas, flujo de redes con cargo fijo, así como los problemas de la mochila y el del agente viajero, entre otros.

Problemas como el del agente viajero o el de la mochila se conocen como problemas 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.

Por otra parte, 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 llaman de 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 del 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.

En investigación de operaciones se han presentado diversos algoritmos no determinísticos que en la práctica, han demostrado que son eficientes resolver lo problemas de optimización combinatorios. La mayoría de estos algoritmos se basan en los procedimientos de programación lineal y en la estructura del método simplex, por lo que su mecanismo es iterativo y, dependiendo del tamaño del problema, alcanzaran la respuesta en tiempo polinomial o exponencial.

Los algoritmos de este tipo son los que se basan en los métodos de **planos de corte, de ramificación y acotamiento** y de **enumeración implícita.** La eficiencia de estos métodos está determinada por su convergencia a la solución óptima en problemas de programación entera y mixta. Sin embargo, su procedimiento de búsqueda implica la generación nuevas variables, lo que determina un tiempo de procesamiento cada vez mayor conforme el tamaño del problema se incrementa, llegando a ser este factor, el tiempo, lo que establece su imposibilidad de aplicación.

En el siguiente capítulo se hace una breve descripción de dichos algoritmos en sus versiones más generales, describiendo sus procedimientos y las ventajas y desventajas de su aplicación en problemas de optimización combinatoria.

CAPITULO II

METODOS DE SOLUCION DE PROBLEMAS COMBINATORIOS

2.1 Introducción

Desde sus origenes, el más importante y notable problema de cómputo de la programación lineal ha sido el encontrar la solución óptima a un programa lineal en las que todas o algunas de la variables estén restringidas a valores enteros. La necesidad que hay de procedimientos computacionales para resolver tales programas se agudiza por el gran número de problemas del campo de análisis combinatorio.

Los procedimientos para resolver programas enteros son clasificados en la mayoría de los textos de investigación de operaciones en dos categorias computacionales: *planos de corte y enumeración*. La idea básica de un método de plano de corte es alterar el conjunto convexo de solución del problema de programación lineal continuo relacionado, es decir, el problema de programación lineal que resulte de ignorar las restricciones enteras, de manera que el punto extremo óptimo al problema continuo sea un valor entero. Esto se consigue al añadir sistemáticamente restricciones adicionales que eliminan partes del conjunto convexo que no contienen algún punto entero posible y resuelven los problemas resultantes por el algoritmo simplex.

Los métodos enumerativos o de búsqueda están diseñados para investigar sistemáticamente un subconjunto restringido del gran conjunto de soluciones enteras. Estos métodos se pueden usar para resolver, tanto problemas enteros como problemas entero mixtos e incluyen métodos de búsqueda de árboles, de ramificación y acotamiento y enumeración implícita o algoritmos aditivos. Los procedimientos de bifurcación y acotamiento usan la información obtenida de soluciones sucesivas de problemas continuos relacionados, para generar nuevos problemas con límites en variables seleccionadas y en la función objetivo, de manera que se restrinja el número total de problemas continuos que se deben resolver.

La enumeración implicita fue diseñada básicamente para resolver problemas con variables binarias, o sea, variables restringidas a los valores cero o uno. Como hay 2^n soluciones posibles, se requiere un procedimiento para enumerar sistemáticamente parte de las soluciones al problema binario y examinarlo de tal forma, que asegure que al enumerar el número (realmente pequeño) de soluciones, se examinan implicitamente todos los elementos del conjunto de la solución. El procedimiento no emplea ninguna de las técnicas de la programación lineal sino que sustituye nuevas soluciones 0 - 1 basadas en reglas sistemáticas para mejorar una solución que descanza en la información implicita de las restricciones originales del problema.

En este capítulo se realiza una descripción de los principales algoritmos que se presentan en los textos que sobre el tema existen.

2.2 Métodos de planos de corte

Los métodos llamados planos de corte fueron los primeros que se utilizaron para hacer frente a los problemas de programación entera y fueron presentados originalmente por Gomory.

Estos métodos son eficientes en la solución de problemas que no son muy grandes. Sin embargo en el caso de problemas de tamaño medio resultan ser muy eficientes. Estos métodos generan en cada iteración una restricción y una variable extra. Su ventaja es que ilustran lo que se pretende hacer con la región de factibilidad del problema entero, para lograr la solución del mismo.

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

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

$$\begin{aligned} &\text{minimizar } C^t x \\ &\text{sujeto a} \\ &x \in S \end{aligned} \qquad (1)$$

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

$$\begin{aligned} &\text{minimizar } f(y) \\ &\text{sujeto a} \\ &y \in R \end{aligned} \qquad (2)$$

donde $R \in E^{n-1}$ es un conjunto convexo y f, una función convexa, se puede convertir fácilmente a la forma (1) escribiendo (2) de manera equivalente como:

$$\begin{aligned} &\text{minimizar } r \\ &\text{sujeto a} \\ &f(y) - r \leq 0 \end{aligned} \qquad (3)$$

lo cual con $x = (r, y)$ es un caso especial de (1).

La forma general de un algoritmo de plano de corte para el problema (1) es la siguiente: dado un politope P_k S

PASO 1 Minimice Cx, 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

$$S \in \{ x \mid a_k^t x \leq b_k \}$$

$$x_k \in \{ x \mid a_k^t x > b_k \}$$

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 2.1

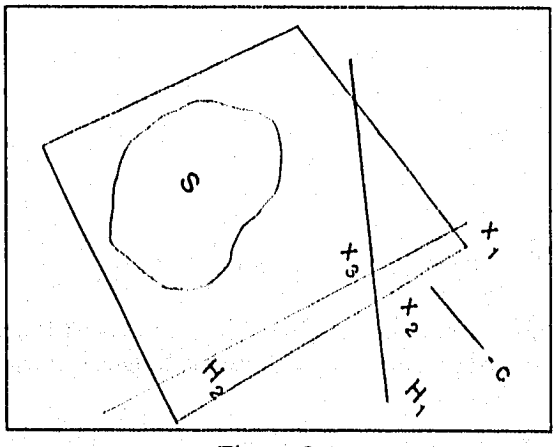

Figura 2.1

Una especificación de los algoritmos de planos de corte radica en la manera de seleccionar el hiperplano que separa el punto actual x_k del conjunto de restricciones de S. Por supuesto, esta selección es el aspecto más importante del algoritmo, pues la profundidad del corte asociado al hiperplano de separación, la distancia del hiperplano al punto actual, que determina cuanta mejora hay en la aproximación al conjunto de restricción y, por tanto, la rapidez de convergencia del método.

Otro factor importante viene dado en la manera de actualizar el politope una vez determinado el hiperplano nuevo. El procedimiento más directo consiste simplemente en añadir la desigualdad lineal asociada a este hiperplano a aquellas determinadas antes. Esto proporciona la mejor aproximación actualizada posible al conjunto restricción, pero tiende a producir, después de un gran número de iteraciones, una cantidad enorme de desigualdades que expresan la aproximación. Así, en algunos algoritmos,las desigualdades primitivas, que no son obligatorias en el punto actual, dejan de tenerse en consideración.

El planteamiento que fundamenta la estructura del método de planos de corte, se basa en el siguiente desarrollo:

Se entiende por $\lceil X \rceil$ al entero más grande menor que el número X, es decir

$$\lceil X \rceil = \text{Máx } Y \leq X, \; Y \text{ entero.} \qquad (4)$$

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

Se considera el siguiente problema entero [*PE*]

$$\text{Máx } \mathbf{cX}$$

sujeto a

$$\mathbf{AX} = \mathbf{b}$$
$$\mathbf{X} \geq \mathbf{0}, \text{ entero.} \tag{5}$$

El problema lineal correspondiente a (5) es

$$\text{Máx } \mathbf{cX}$$

sujeto a (6)

$$\mathbf{AX} = \mathbf{b}$$

$$\mathbf{X} \geq \mathbf{0}.$$

Una restricción típica de (6) es

$$a_{i1}X_1 + a_{i2}X_2 + \ldots + a_{in}X_n + X_{n+i} = b_i,$$

donde X_1, X_2, X_n son las variables no básicas, $X_{n+1},\ldots,X_{n+m}$ son las variables básicas y b_i es el valor de la variable básica correspondiente. Esta restricción puede escribirse como

$$\sum_{j=1}^{n} \lceil a_{ij} \rceil X_j + \sum_{j=1}^{n} (a_{ij} - \lceil a_{ij} \rceil)X_j + X_{n+i} = \lceil b_i \rceil + (b_i - \lceil b_i \rceil)$$

Poniendo a las partes enteras de un lado de la expresión y a las fracciones del otro, se tiene

$$\sum_{j=1}^{n} \lceil a_{ij} \rceil X_j + X_{n+i} - \lceil b_i \rceil = (b_i - \lceil b_i \rceil) - \sum_{j=1}^{n} (a_{ij} - \lceil a_{ij} \rceil)X_j$$

como la parte izquierda de la igualdad es entera, la parte derecha también debe serlo.

Teorema 1. Dado (6) se tiene que

$$\sum_{j=1}^{n} (a_{ij} - \lceil a_{ij} \rceil)X_j \geq (b_i - \lceil b_i \rceil)$$

Prueba. Dada un restricción típica de (6):

$$\sum_{j=1}^{n} (a_{ij}X_j + \lceil X_{n+i} \rceil) = b_i$$

se le puede multiplicar por $h \neq 0$ y por $\lceil h \rceil \neq 0$, h arbitrario, quedando respectivamente

$$\sum_{j=1}^{n} ha_{ij}X_j + hX_{n+i} = hb_i \tag{7}$$

$$\sum_{j=1}^{n} \lceil h \rceil a_{ij}X_j + \lceil h \rceil X_{n+i} = \lceil h \rceil b_i \tag{8}$$

De la expresión (7) y de la definición (4) se tiene

$$\sum_{j=1}^{n} \lceil ha_{ij} \rceil X_j + \lceil h \rceil X_{n+i} \leq hb_i \tag{9}$$

por lo tanto, como la parte izquierda de la desigualdad es entera, se tiene

$$\sum_{j=1}^{n} \lceil ha_{ij} \rceil X_j + \lceil h \rceil X_{n+i} \leq \lceil hb_i \rceil \tag{10}$$

(10) menos (8) da

$$\sum_{j=1}^{n} \lceil ha_{ij} \rceil X_j - \sum_{j=1}^{n} \lceil h \rceil a_{ij}X_j + \lceil h \rceil X_{n+i} - \lceil h \rceil X_{n+i} \leq \lceil hb_i \rceil - \lceil h \rceil b_i,$$

$$\sum_{j=1}^{n} (\lceil ha_{ij} \rceil - \lceil h \rceil a_{ij})X_j \leq \lceil hb_i \rceil - \lceil h \rceil b_i$$

o

$$\sum_{j=1}^{n} (\lceil h \rceil a_{ij} - \lceil ha_{ij} \rceil)X_j \geq \lceil h \rceil b_i - \lceil hb_i \rceil \tag{11}$$

Como h es arbitrario, se hace $h = 1$, quedando (11) como

$$\sum_{j=1}^{n} (a_{ij} - \lceil a_{ij} \rceil)X_j \geq (b_i - \lceil b_i \rceil)$$

con lo que el teorema queda probado.

Los métodos de *planos de cortes* están basados fundamentalmente en la restricción

$$\sum_{j=1}^{n} (a_{ij} - \lceil a_{ij} \rceil) X_j \geq (b_i - \lceil b_i \rceil) \qquad i = 1, \ldots, m$$

que constituye un *corte*.

Los *métodos de corte*, que se desarrollan principalmente para problemas *lineales* enteros, comienzan con el óptimo continuo. Sumando sistemáticamente restricciones secundarias especiales, que representan básicamente condiciones necesarias de integridad, el espacio de soluciones continuo se modifica de manera gradual hasta que su punto extremo óptimo continuo satisface las condiciones enteras. El nombre métodos de corte surge del hecho que las restricciones secundarias sumadas cortan (o eliminan) efectivamente ciertas partes del espacio de soluciones que no contienen puntos enteros factibles.

2.2.1 Algoritmo fraccional de Gomory

Uno de los métodos de corte tradicionales es el algoritmo fraccional de Gomory, cuyo procedimiento se puede generalizar en los siguientes pasos:

Paso 1. Se resuelve el problema entero como un problema lineal, olvidándose por el momento de las condiciones de integralidad.

Paso 2. Si el resultado óptimo del paso 1 o del paso 3 es entero, parar. De otra manera continuar en el paso 3.

Paso 3. Seleccionar el máximo $(_{bi} - \lceil _{bi} \rceil)$ fraccionario y generar un corte

$$\sum_{j=1}^{n} (a_{ij} - \lceil a_{ij} \rceil) X_j \geq (b_i - \lceil b_i \rceil)$$

Añadir este corte como una restricción adicional, junto con su variable superflua. Resolver el problema por el *método dual simplex* y regresar al paso 2.

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[1]. 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, por lo que únicamente cambia el valor de las variables no básicas. También, cuando una variable se convierte en no básica, se restaura en la fila correspondiente una restricción de no negatividad.

El algoritmo se conoce como el *método fraccional* porque todos los coeficientes diferentes de cero del corte generado, son menores que uno.

2.2.2 Algoritmo entero puro de Gomory.

Este método es una variación del anterior y pertenece a un proceso de optimización con punto de partida que es dual factible. En este caso *todos los coeficientes de la matriz A deben ser enteros, aspecto que no se presenta en el algoritmo anterior.* El método asegura que esta condición de integridad de los coeficientes permanece constante durante todas las iteraciones. Los pasos a seguir son los siguientes:

Paso 1. Se escoge la b_i más negativa. Se designa a esa fila con r. Si el *método dual simplex* genera un pivote -1, se hace la iteración y se repite el paso 1. Si no, se continúa en el paso 2.

Paso 2. Se escoge aquella columna no-básica, con $a_{rj} < 0$ que lexicográficamente sea la menor. Se designa a la columna por 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 $u_{ij}\left[\frac{a_{pj}}{a_{kp}}\right]$, 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 $\lambda = \text{Máx}\ u_{ij}\left[\frac{a_{rj}}{u_j}\right]$ para $a_{rj} < 0$ y $u_j \neq \infty$.

---

1. El *algoritmo fraccional*, puede indicar, a primera vista, que el tamaño de la tabla simplex puede llegar a ser muy grande en cuanto nuevos cortes se agregan al problema. Esto no es cierto. En efecto, el número total de restricciones en el problema *aumentado* no puede exceder el número de variables en el problema original; o sea, $(m + n)$. Esto se deduce puesto que si el problema aumentado incluye más de $(m + n)$ restricciones, una de las variables de holgura asociadas a los cortes fraccionales debe ser básica. En este caso, las ecuaciones asociadas llegan a se redundantes y pueden quitarse completamente de la tabla.

Paso 5. Se deriva un corte

$$\sum_{j=1}^{n} \left[\frac{a_{rj}}{\lambda} \right] X_j \leq \left[\frac{X_{b_i}}{\lambda} \right],$$

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, al elemento del tableau en la fila i columna j, se le designa a_{ij}.

El algoritmo fraccional tiene dos desventajas:

Primera. Los errores de redondeo que resultan en cálculos automáticos pueden proporcionar la solución entera óptima equivocada, en especial al aumentar el tamaño del problema.

Segunda. La solución del problema permanece infactible en el sentido de que ninguna solución *entera* puede obtenerse hasta que se alcanza la solución entera óptima. Esto significa que no habrá *buenas* soluciones enteras almacenadas si los cálculos se paran prematuramente antes de obtener la solución óptima entera.

La primera dificultad se evita con el desarrollo de un *algoritmo todos los enteros-enteros*. El algoritmo comienza con una tabla totalmente entera (esto es, todos los coeficientes son enteros) adecuada para la aplicación del algoritmo dual simplex.

Una restricción especial adicional se constituye entonces, de modo que su adición a la tabla preserve que sean enteros todos los coeficientes. Sin embargo, el hecho de que la solución permanezca infactible hasta que se logra la solución óptima entera, todavía presenta una desventaja.

La segunda dificultad que se considera al desarrollar algoritmos de planos de corte, es que comienzan como enteros factibles pero no óptimos. Las iteraciones continúan siendo factibles y enteras hasta que se alcanza a la solución óptima. En este aspecto, este algoritmo es *primal factible* al compararse con los algoritmos de Gomory que son *duales factibles*. Sin embargo, los algoritmos primales no parecen ser computacionalmente promisorios.

2.2.3 Algoritmo entero-mixto de Gomory.

Este algoritmo, también basado en el método de *planos de corte*, sirve para resolver problemas enteros-mixtos. El punto inicial de la optimización, es factible primal-dual. Dada una restricción en cualquier iteración.

$$a_{i1}X_1 + a_{i2}X_2 + \ldots + a_{in}X_n + X_{n+i} = b_i$$

está puede re-escribirse como

$$\sum_{j=1}^{n} \left(a_{ij}^{+} + a_{ij}^{-}\right) X_j + X_{n+i} = (b_i - \lceil b_i \rceil + \lceil b_i \rceil)$$

donde

$$a_{ij}^{+} = \begin{cases} a_{ij}, & si\ a_{ij} \geq 0, \\ 0, & si\ a_{ij} < 0, \end{cases} \qquad a_{ij}^{-} = \begin{cases} 0, & si\ a_{ij} \geq 0, \\ a_{ij}, & si\ a_{ij} < 0, \end{cases}$$

Se supone que X_{n+i} es una variable entera, pero b_i no lo es. Entonces haciendo

$$h = b_i - \lceil b_i \rceil$$

$$z = \lceil b_i \rceil$$

se tiene

$$\sum_{j=1}^{n} \left(a_{ij}^{+} + a_{ij}^{-}\right) X_j + X_{n+i} = h + z \qquad (12)$$

Si la parte izquierda de (12) es positiva, sería igual a h, $h + 1$, $h + 2$, ... y la siguiente expresión sería cierta[2].

$$\sum_{j=1}^{n} \left(a_{ij}^{+} + a_{ij}^{-}\right) X_j \geq h$$

Pero como se tiene que

$$\sum_{j=1}^{n} a_{ij}^{+} X_j \geq \sum_{j=1}^{n} \left(a_{ij}^{+} + a_{ij}^{-}\right) X_j \qquad (13)$$

entonces

$$\sum_{j=1}^{n} a_{ij}^{+} X_j \geq h$$

Por otro lado, si la parte izquierda de (12) es negativa, ésta sería igual a $-1 + h$, $-2 + h$, $+\ldots$ y la siguiente expresión sería cierta

[2]La expresión es cierta porque de ambos lados se han quitado cantidades enteras pero $X_{n+i} \leq Z = \lceil b_i \rceil$

$$\sum_{j=1}^{n} \left(a_{ij}^{+} + a_{ij}^{-}\right) X_j \leq -1 + h$$

Pero

$$\sum_{j=1}^{n} \left(a_{ij}^{-}\right) X_j \leq \sum_{j=1}^{n} \left(a_{ij}^{+} + a_{ij}^{-}\right) X_j$$

por lo que

$$\sum_{j=1}^{n} \left(a_{ij}^{-}\right) X_j \leq -1 + h$$

o sea, dividiendo esta última expresión por $(h - 1)$, que es negativo, se tiene

$$\sum_{j=1}^{n} \frac{\left(a_{ij}^{-}\right)}{h-1} X_j \geq 1$$

Multiplicado por h

$$\sum_{j=1}^{n} h\frac{\left(a_{ij}^{-}\right)}{h-1} X_j \geq h$$

o

$$\sum_{j=1}^{n} h \frac{\left(-a_{ij}^{-}\right)}{1-h} X_j \geq h \qquad (14)$$

Resumiendo (13) y (14) se tienen:

$$\sum_{j=1}^{n} \left(a_{ij}^{+}\right) X_j + \sum_{j=1}^{n} \frac{h}{1-h} \left(-a_{ij}^{-}\right) X_j \geq h \qquad (15)$$

La expresión (15) constituye un *corte*. Para hacer este corte más efectivo hay que tomar en cuenta dos consideraciones:

a) que algunas variables en la restrieción, diferentes de X_{n+1} pueden ser enteras, y

b) se quiere que los coeficientes de esas variables enteras diferentes de X_{n+1}, sean lo más pequeñas posible. El coeficiente a_{ij}^{+} más pequeño en (10) es $(a_{ij} - [a_{ij}])$, mientras que el coeficiente a_{ij}^{+} más grande que se puede obtener en (11) es $(1 - a_{ij} + [a_{ij}])$. Por lo tanto, el corte (12) se puede transformar en un corte más efectivo, dado por:

$$\sum_{j=1}^{n} (a_{ij}^{+})\ X_j + \sum_{j=1}^{n} \frac{h}{1-h}\,(-a_{ij}^{-})\ X_j \qquad + \qquad \sum_{j=1}^{n} (a_{ij} - [a_{ij}])\ X_j \quad +$$

Para variables no enteras

Para variables no enteras con $(a_{ij} - [a_{ij}]) \leq h$

$$+ \sum_{j=1}^{n} \frac{h}{(1-h)}\,(1 - a_{ij} - [a_{ij}])\ X_j \geq h$$

Para variables no enteras con $(a_{ij} - [a_{ij}]) > h$

El algoritmo *mixto-entero de Gomory* procede de la misma manera que su algoritmo *fraccionario*, siendo la estructura del corte el que diferencia a ambos algoritmos.

2.2.4 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 no es especialmente útil en el análisis del método, revela la conexión básica entre los métodos de planos de corte y el dual. La base de este punto de vista es el hecho de que S puede escribirse como la intersección de todos los espacios que lo contienen, así

$$S = \{ x : a_i' x \leq b_i, \in I \}$$

donde I es un conjunto de índices (infinito) correspondiente a los semiespacios que contienen a S. Con S considerado de ese modo el problema (2.1) se puede tomar como un problema de programación lineal finito.

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

maximizar $\sum a_i b_i \qquad i \in I$

sujeto a

$$\sum \lambda_i a_i = c\lambda$$
$$\lambda_i \geq 0,$$

Seleccionando un subconjunto finito S, por ejemplo I, y formando

$$P = \{ x : a^T_i x \leq b_i, \in I \}$$

resulta un politope que contiene S. Al minimizar $c^T x$ sobre este politope, se generan un punto y un subconjunto correspondiente de restricciones activas I_A. Entonces el problema dual con la restricción adicional $\lambda_i = 0$ 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 óptima del dual. Por esta razón, se puede considerar que el método de plano cortante busca la optimalidad del dual (de dimensión infinita).

La diferencia entre los métodos duales y los primales en los algoritmos que utilizan planos de corte, es la siguiente. Los métodos duales no generan una solución factible del problema sino hasta que se llega a la solución óptima. Al respecto, la mayor parte de los autores coincide en que para ciertos problemas, después de una cantidad bastante grande de iteraciones, lo mejor que se ha logrado, es definir una cota de la función objetivo del problema. En cambio, los métodos primarios, pueden producir desde la primera iteración una solución factible. El problema estriba en que esta solución converge de forma lenta a la solución óptima.

La desventaja de los métodos de planos de corte, 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.

2.3 Métodos de ramificación y acotamiento.

El *método de ramificación y acotamiento* redondea y acota variables enteras, resultantes de la solución de los problemas lineales correspondientes. Este proceso de acotamiento y redondeo se hace de una manera secuencial lógica heurística, que permite eliminar con anticipación un buen número de soluciones factibles alejadas del óptimo a medida que se itera. De tal forma que si una variable entera X_j, $j = 1, \ldots, n$, está acotada entre un límite inferior entero d_j, $j = 1, \ldots, n$ y un límite superior entero u_j, $j = 1, \ldots, n$, el proceso de bifurcación y acotación sólo analiza un número muy pequeño de todas las posibles soluciones. Para tener una idea de la gran cantidad de posibles soluciones, se debe tener presente que sólo la variable X_j puede tomar cualquiera de los siguientes valores enteros: d_j, d_{j+1}, d_{j+2}, ..., u_{j-2}, u_{j-1}, u_j. Si se tienen n variables enteras, el número de posibles combinaciones que se pueden obtener es muy grande.

Esta técnica también resuelve el problema entero considerando su versión continua. Pero a diferencia de los métodos de corte, el método de ramificar y acotar se aplica directamente a ambos tipos de problemas, el *puro* y el *mixto*.

La idea general del método es resolver primero el problema como un modelo continuo (programa lineal). Se supone que $\lceil x_r \rceil$ es una variable entera restringida cuyo valor óptimo continuo x_r^* es fraccional. El intervalo

$$x_r^* < \lceil x_r \rceil < x_r^* + 1$$

no puede incluir ninguna solución entera factible. Consecuentemente, un valor entero factible de x_r debe satisfacer *una* de dos condiciones, a saber:

$$\lceil x_r \rceil \leq x_r^* \text{ o bien } \lceil x_r \rceil \geq x_r^* + 1$$

Estas dos condiciones, cuando se aplican al modelo proporcionan dos problemas mutuamente excluyentes, que se crean imponiendo las restricciones $x_r \leq \lceil x_r^* \rceil$ y $x_r \geq \lceil x_r^* \rceil + 1$ al espacio de soluciones original. En este caso se dice que el problema original se ha ramificado o partido en dos subproblemas. Realmente el procedimiento de ramificación desecha partes del espacio continuo que no incluyen puntos enteros factibles, reforzando las condiciones *necesarias* para que se tengan los enteros.

Ahora, cada subproblema puede resolverse como un programa lineal utilizando la misma función objetivo del problema original. Si su *óptimo* es factible con respecto al problema entero, su solución se registra como la mejor disponible. En este caso será innecesario *ramificar* adicionalmente este subproblema ya que no puede proporcionar una mejor solución entera. De otra manera, el subproblema debe partirse en dos subproblemas imponiendo de nuevo las condiciones enteras sobre una de sus variables enteras, que por lo general tiene un valor óptimo fraccional.

Naturalmente, cuando se encuentra una *mejor* solución entera factible para cualquier subproblemas, ésa deberá reemplazar la que se tenía. El procedimiento de ramificar continua, cuando sea aplicable, hasta que cada subproblema termine con una solución entera o cuando existe evidencia de que no puede haber una mejor. En este caso la solución factible actual, si existe alguna, es la óptima.

La eficiencia de cómputo puede aumentarse introduciendo el concepto de *acotamiento*. Este concepto indica que, si la solución óptima continua de un subproblema proporciona un valor de la función objetivo peor que el asociado a la mejor solución entera disponible, no vale la pena explorar adicionalmente el subproblema. En este caso se dice que el subproblema esta *exhausto* y, por consiguiente, puede desecharse.

En otras palabras, una vez que se encuentra una solución factible entera, su valor de la función objetivo asociado, puede ser utilizado como una *cota* (superior en el caso de minimización e inferior en el caso de maximización) para descartar subproblemas inferiores.

No puede enfatizarse demasiado la importancia de adquirir una *buena* cota en las primeras etapas del cálculo. En términos del procedimiento anterior esto parece depender directamente del orden en el que se *generan* y *examinan* los subproblemas.

Los problemas específicos generados dependen de la variable seleccionada para efectuar la ramificación. Desafortunadamente, no existe la forma *mejor* definida; para seleccionar la variable ramificadora o la sucesión específica con que debe examinarse el subproblema. Pero existen reglas empíricas que mejoran el proceso. Estas reglas usualmente están implantadas en la mayoría de los códigos de ramificar y acotar.

2.3.1 Aceleración de los métodos de ramificación y acotamiento.

Los *métodos de ramificación y acotamiento*, pueden acelerarse, es decir, hacerlos converger más rápidamente a la solución óptima, si se tiene un poco más de cuidado en la selección de la variable entera que genera una ramificación.

Se entiende por $[X]$ al número entero z más grande, menor o igual a X, y por $<X>$ al número entero z más pequeño, mayor o igual a X. Por ejemplo:

$[6.5]=6$, $[-6.5]=-7$, $[0.3]=0$, $[-0.3]=1$.
$<6.5>=7$, $<-6.5>=-6$, $<0.3>=1$, $<-0.3>=0$.

Los costos penales de Driebeek sirven para seleccionar al mejor candidato. Dada una variable básica $X_r = X_{Br}$ cuyo resultado final debe ser entero, y por el momento es todavía fraccionario, se tiene que el costo penal de hacer $X_r = [X_{Br}]$ es

$$(X_{B_r} - [X_{B_r}]) \underset{j=1,\ldots,n}{Min} \left\{ \frac{z_j - c_j}{Y_{rj}} \middle| Y_{rj} > 0 \right.$$

mientras que el costo penal de hacer $X_r = <X_{Br}>$ es

$$(1 - X_{B_r} + [X_{B_r}]) \underset{j=1,\ldots,n}{Min} \left\{ \frac{z_j - c_j}{-Y_{rj}} \middle| Y_{rj} < 0 \right.$$

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 penales, uno asociado con el cambio $X_r = [X_{Br}]$ y el otro con $X_r = <X_{Br}>$. Para identificación, se denota al primer costo penal por $[CP]_r$ y al segundo por $<CP>_r$. La variable que se selecciona para ramificarse es aquella cuyo $[CP]_r$ ó $<CP>_r$ es el máximo entre todas las variables básicas X_r, que siendo aún fracciones, deben ser enteras en la solución óptima. este proceso acelera al método, pues al elegir una variable básica con costo penal alto, se evita aumentar el número de iteraciones al eliminar implícitamente soluciones peores a las actuales.

2.3.2 Cálculos en métodos de ramificar y acotar

Los paquetes prácticos de computadora basados sobre la técnica de ramificar y acotar difieren de la parte anterior principalmente en los detalles al seleccionar las variables en un nodo y la sucesión con que se generan los subproblemas basados en métodos heurísticos desarrollados por medio de la experimentación.

Una desventaja básica del algoritmo anterior consiste en que es necesario resolver un programa lineal completo en cada nodo. En problemas grandes, esto podría consumir mucho tiempo, particularmente cuando la única información necesaria en el nodo puede ser su valor

óptimo de la función objetivo. Este punto se aclara notando que una vez que se obtiene una *buena* cota, muchos nodos podrían descartarse del conocimiento de sus valores objetivos óptimos.

La idea anterior llevó al desarrollo de un procedimiento donde llega a ser innecesario resolver todos los subproblemas del Arbol de ramificar y acotar. La idea es *estimar* una cota *superior* (supóngase un problema de maximización) sobre el valor óptimo de la función objetivo en cada nodo. Si esta cota superior es más pequeña que el valor de la función objetivo asociado con la mejor solución entera disponible, entonces se descarta el nodo. La principal ventaja es la estimación rápida de estas cotas superiores con cálculos mínimos. La idea general es estimar las *penalizaciones* (esto es, el deterioro en el valor de la función objetivo) que resultan al introducir las condiciones $x_k \leq [\beta_k]$ y $x_k \geq [\beta_k] + 1$. Esto puede lograrse aumentando cada una de estas restricciones a la tabla óptima en el nodo (con el cual está asociada x_k). Entonces, *bajo la hipótesis de ningún cambio en la base*, la penalización requerida puede estimarse directamente a partir de los coeficientes de la función objetivo.

Aunque las penalizaciones son fáciles de calcular tienen la desventaja de que no necesariamente son proporcionales al deterioro real del valor objetivo. En otras palabras, no proveen una cota exacta y entonces, pueden no ser efectivas. Se han hecho varios intentos para reforzar estas penalizaciones. El más interesante de éstos es el que utiliza información de los métodos de corte. Sin embargo, aun estas penalizaciones *reforzadas* parecen ser inefectivas computacionalmente, en particular cuando el tamaño del problema aumenta. Parece que los paquetes comerciales han abandonado el uso de penalizaciones simples en favor de métodos heurísticos que han probado, a través de la experimentación con problemas grandes y complejos, ser muy efectivos (como los que se analizan en el capítulo siguiente). Independientemente de las desventajas del método de ramificar y acotar, puede establecerse que, a la fecha, estos métodos son los que académicamente se utilizan para resolver problemas enteros de gran tamaño que se presentan en la práctica. Decididamente, todos los paquetes comerciales disponibles están basados en el método de ramificar y acotar. Esto *no* significa, sin embargo, que *cada* problema entero pueda resolverse por un método de ramificar y acotar. Unicamente quiere decir que cuando la elección está entre un método de corte y uno de ramificar y acotar, generalmente se ha comprobado que el último es superior.

2.3.3 Convergencia del algoritmo de Ramificación y Acotamiento.

La operación de ramificación del algoritmo garantiza una solución óptima en un número finito de iteraciones. Como T, que representa todas las soluciones posibles del problema, 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 T para la cual cada subconjunto T_i, obedece a la definición de partición. Así todos los nodos finales posibles se pueden generar y obtenerse así la solución óptima. Las características de la ramificación prometen una enumeración completa para el algoritmo ramificación y acotamiento a menos que se pueda encontrar una solución óptima antes de hacerlo. Las características de acotamiento del algoritmo proporcionan la posibilidad de reconocer una solución óptima antes de completar la numeración.

El método de ramificación y acotamiento 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. 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.

2.4 Métodos de enumeración implícita

Los métodos de enumeración implícita son métodos heurísticos, basados en la lógica, que como los métodos de ramificación y acotamiento resuelven problemas enteros, sin tener que analizar todas las posibles alternativas. Así como el método de planos de corte resuelve un problema entero mediante la modificación de la región de factibilidad del problema lineal correspondiente, y el método de bifurcación y acotación mediante la ramificación de problemas que obligan a una variable fraccionada a tomar el valor entero inmediato mayor o menor de la fracción, el método de enumeración implícita resuelve problemas enteros mediante la aceptación o rechazo implícito de ciertas alternativas.

Por ejemplo, para el caso de enumeración implícita binaria, cualquier variable entera puede expresarse equivalentemente en términos de un cierto número de variables puras *cero-uno*. La forma más simple de lograr esto es como sigue. Sea $0 \leq x \leq n$ una variable entera donde n es una cota superior entera. Entonces, dadas $y_1, y_2, \ldots, y_n$ como variables cero-uno,

$$X = y_1 + y_2 + \ldots + y_n$$

es una representación binaria exacta de todos los valores factibles de x. Otra representación (más económica) donde el número de variables binarias es usualmente menor que n es

$$X = y_0 + 2y_1 + 2^2y_2 + \ldots + 2^ky_k$$

donde k es el entero más pequeño que satisface $2^{k+1} - 1 \geq n$.

El hecho de que cada problema entero pueda hacerse binario junto con la simplicidad de cómputo al tratar con variables cero-uno (cada variable tiene dos valores solamente) ha dirigido la atención en la explotación de estas propiedades a fin de desarrollar un algoritmo eficiente.

El algoritmo que se presentar a continuación es realmente una variante del método más general de ramificar y acotar. Es interesante observar que la versión original del algoritmo no se presenta en este contexto. Quizá la principal razón es que el algoritmo original y, por cierto, sus modificaciones nunca requieren la solución de problemas lineales. En efecto, las únicas operaciones aritméticas necesarias son sumas y restas. Este es el porqué algunas veces se conoce como el *algoritmo aditivo*.

2.4.1 Algoritmo aditivo (problema binario puro)

Para el propósito de este algoritmo, la versión continua del problema cero-uno debe comenzar como dual factible. Además, todas las restricciones deben ser del tipo ($\leq$), excluyendo, por tanto, ecuaciones explícitas. Este formato siempre puede lograrse de la manera siguiente. Sea el problema del tipo de minimización (no existe pérdida en la generalidad aquí) definido como

$$\text{minimizar} \quad z = \sum_{j=1}^{n} c_j x_j$$

sujeto a

$$\sum_{j=1}^{n} a_{ij} x_j + S_i = b_i \qquad i = 1,2,\ldots,m$$

$$x_j = 0 \text{ ó } 1, \qquad j = 1,2,\ldots,n$$

$$S_i \geq 0, \qquad i = 1,2,\ldots,m$$

donde S_i es la variable de holgura asociada a la restricción i.

La versión continua del problema anterior es dual factible si toda $c_j \geq 0$. Cualquier $c_j < 0$ puede convertirse al formato deseado complementando la variable x_j, esto es, sustituyendo $x_j = 1 - x'_j$, donde x'_j es una variable binaria, en la función objetivo y en las restricciones. Si además de la factibilidad dual, el problema es primal factible, nada más necesita hacerse ya que el mínimo, en términos de las nuevas variables, se logra asignando valores cero a todas las variables. Sin embargo, es primal infactible, y el algoritmo aditivo se utiliza para encontrar el óptimo.

La idea general del algoritmo aditivo es enumerar todas las 2^n soluciones posibles del problema. Sin embargo, reconoce que algunas soluciones pueden descartarse automáticamente sin ser investigadas en forma explícita. Entonces en el análisis final, únicamente una porción de las 2^n soluciones necesita investigarse explícitamente.

En términos del problema cero-uno anterior, la idea antes mencionada se implanta como sigue. Inicialmente, se supone que todas las variables están en el nivel cero. Esto es lógico ya que todas las $c_j \geq 0$. Como la solución correspondiente no es factible (esto es, algunas variables de holgura S_i pueden ser negativas), será necesario elevar algunas variables al nivel uno. El procedimiento pide elevar una, o quizá más variables, a la vez, siempre que exista la evidencia de que este paso estará moviendo las soluciones hacia la factibilidad, o sea, haciendo $S_i \geq 0$ para toda i.

Se ha desarrollado un número de pruebas para asegurar la selección adecuada de las variables que se van a elevar al nivel uno. Estas se presentan primero por medio de un ejemplo numérico y luego se formalizan matemáticamente.

Ahora se presenta la versión general del algoritmo aditivo, utilizando el concepto de soluciones parciales. También se generalizan las pruebas de exclusión usadas para examinar soluciones parciales y aumentar nuevas variables al nivel uno en el problema cero-uno.

Considérese el problema binario general siguiente.

$$\text{minimizar } z = \sum_{j=1}^{n} c_j x_j \qquad \text{para toda } c_j \geq 0$$

sujeto a

$$\sum_{j=1}^{n} a_{ij} x_j + S_i = b_i \qquad i = 1,2,\ldots,m$$

$$x_j = 0 \text{ ó bien, } 1, \qquad \text{para toda } j$$

$$S_i \geq 0, \qquad \text{para toda } i$$

Sea J_t la solución parcial en el nodo t (inicialmente, $J_0 \equiv \varnothing$, lo cual significa que todas las variables son libres) y se supone que z^t es el valor asociado de z mientras que z es la mejor cota superior actual (inicialmente $z = \infty$).

Prueba 1: Para cualquier variable libre x_r, si $a_{ir} \geq 0$ para toda i correspondiendo a $S_i^t < 0$, entonces, x_r no puede mejorar la infactibilidad del problema y debe descartarse como no promisoria.

Prueba 2: Para cualquier variable libre x_r, si

$$C_r + Z^t \geq z$$

entonces x_r no puede llevar a una solución mejorada y, por tanto, debe descartarse.

Prueba 3: Considere la i-ésima restricción

$$a_{i1}x_1 + a_{i2}X_2 + \ldots + a_{in}x_n + S_i = b_i$$

para la cual $S_i^t < 0$. Sea N_t el conjunto de variables *libres* no descartadas por las pruebas 1 y 2. Ninguna de las variables libres en N_t es promisoria si para al menos una $S_i^t < 0$, la condición siguiente está satisfecha.

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

Esto realmente establece que el conjunto de N_t no puede llevar a una solución factible y, por

tanto, debe descartarse. En este caso, se dice que J_t está exhausta.

Prueba 4: Si $N_t \neq \varnothing$ la variable de ramificación x_k se elige como aquélla correspondiente a

$$V_k^t = \max_{j \in N_t} (V_j^t)$$

donde

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

Si $V_k = 0$, $X_k = 1$ junto con J_t proporciona una solución factible *mejorada*. En este caso, J_{t+1} la cual está definida como J_t aumentada con {k} a la derecha, está exhausta. De otra manera, las pruebas anteriores se aplican de nuevo a J_{t+1} hasta que la enumeración se completa, esto es, hasta que *todos* los elementos de la solución parcial *exhausta* sean negativos.

2.4.2 Método aditivo de Balas para resolver problemas binarios (cero-uno) por enumeración implícita.

El problema a resolver es

$$Mín\ Z = \sum_{j=1}^{n} c_j X_j$$

sujeto a

$$\sum_{j=1}^{n} a_{ij} X_j \leq b_i, \quad i = 1, \ldots, m$$

$$X_j = 0 \text{ ó } 1, \quad j = 1, \ldots, n$$

Se supone lo siguiente:

a) que la función sea minimizada. En el caso de que sea un proceso de maximización, por las reglas de equivalencia, se convierte el problema en uno de minimización.

b) se requiere que $c_j \geq 0$. En caso de que $c_j < 0$, entonces la variable X_j se substituye por otra $\overline{X_j}$, donde $\overline{X_j} = 1 - X_j$. Es decir $\overline{X_j}$ es el complemento de X_j. Así por ejemplo, Mín $Z = 3X_1 - 2X_2$, quedaría Mín $Z = 3X_1 + 2\ \overline{X_2} - 2$, que para el caso de optimización es

equivalente al Mín $Z = 3X_1 + 2\ \overline{X_2}$, donde $\overline{X_2} = 1 - X_2$.

Con estas dos suposiciones se puede garantizar la existencia de una solución inicial que sea básica factible y dual. Los coeficientes c_j y a_{ij} no necesitan ser números enteros.

Sea para $j = 1,2,\ldots,n$

$$X_j^+ = \begin{cases} X_j, & si\ X_j > 0, \\ 0, & si\ X_j \leq 0, \end{cases} \qquad X_j^- = \begin{cases} 0, & si\ X_j > 0, \\ X_j, & si\ X_j \leq 0, \end{cases}$$

A un conjunto específico de variables que describen un problema se le denominará solución parcial, ya que de aquellas se generarán las soluciones completas del problema. De manera análoga, aquellas variables que no se especifican en la solución parcial, se les llamará variables libres o variables no especificadas. Se utiliza la convención que la componente que se encuentra a la extrema izquierda de una solución parcial, es la primera variable especificada y así con las demás posiciones. Una segunda solución parcial es continuación de la primera, si todos los elementos de esta última, aparecen en la misma posición que en la primera solución. Por ejemplo la solución parcial (X_4=1, X_3=0, X_{10}=1), es una continuación de ($X_4 = 1$, $X_3 = 0$).

Se denota *a una solución parcial cualquiera por*

$$(j_1++, j_2--, j_3-, j_4-, j_5+, j_6-, j_7++, \ldots, j_p-)$$

cuyo significado es el siguiente:

j_k++ significa que X_{jk} ha sido seleccionada con valor uno, de acuerdo con las reglas del método.

j_k-- significa que X_{jk} ha sido seleccionada con valor cero, de acuerdo con las reglas del método.

j_k+ significa que X_{jk} es igual a uno, como consecuencia de la continuación de una solución parcial previa.

j_k- significa que X_{jk} es igual a cero, como consecuencia de la continuación de una solución parcial previa.

j_k+ también puede significar que X_{jk} es uno, debido a que todas la posibles continuaciones de $(j_1 j_2, \ldots j_k--)$ han sido implícitamente analizadas.

j_k- también puede significar que X_{jk} es cero, debido a que todas la posibles continuaciones de $(j_1 j_2, \ldots j_k++)$ han sido implícitamente analizadas.

Así por ejemplo, la solución parcial (5++, 3-, 8+, 1--) significa que X_5 se igualó a uno como consecuencia de las reglas del método, y como consecuencia de esa elección se determinó (por deducción o por análisis implícito de todas las consecuencias de tener X_3=1), que X_3=0. De manera análoga, por tener X_5=1, X_3=0 se determina que X_8=1 y por último, por tener X_5=1, X_1=0, X_8=1 se determina que X_1=0.

Cuando en una solución parcial aparece un elemento sin signo, entonces este representa a cualquiera de los siguientes elementos: j_k++, j_k--, j_k+, j_k-.

El método generalizado de Balas requiere que todos los costos $c_j \geq 0$. Las restricciones de igualdad se trabajan tal cual, pero el resto de las desigualdades deben ser de la forma

$$\sum_{j=1}^{n} a_{ij}X_j \leq b_i, \quad i = 1,2,\ldots,m.$$

Se supone que existe una solución parcial $(j_1,j_2,\ldots,j_p)$, sin signos asociados. Inicialmente se hace $Z' = \infty$ o bien igual a M, un número positivo muy grande (por ejemplo $M = \sum_{j=1}^{n} c_j$).

Se tienen los siguientes pasos:

Paso 1. a) Si la solución parcial actual satisface las restricciones del problema, se completa la solución parcial asignando cero a todas las variables libres. Se Hace Z^* igual al valor de la función objetivo correspondiente a la solución parcial. Ir al paso 5.

b) Si la solución parcial actual no satisface las restricciones del problema, se hace Z igual al valor de la función objetivo correspondiente a la solución parcial y se va al paso 2.

Paso 2. Para las variables libres X_j tal que

a) $Z + c_j < Z^*$

y

b) $a_{ij} < 0$ por lo menos para una restricción,

$$b_i - \sum_{j=1}^{p} a_{ij_k}X_{j_k} < 0$$

examinando las relaciones

$$\sum_{j \in N} a_{ij}^{-} \leq b_i - \sum_{k=1}^{p} a_{ij_k}X_{j_k} \qquad (16)$$

donde **N** es el conjunto de variables libres que satisfacen (a) y (b), y

$$a_{ij}^{-} = \begin{cases} a_{ij}, & si\ a_{ij} < 0, \\ 0, & si\ a_{ij} \geq 0, \end{cases} \qquad a_{ij}^{+} = \begin{cases} a_{ij}, & si\ a_{ij} > 0 \\ 0, & si\ a_{ij} \leq 0 \end{cases}$$

Si ninguna de estas restricciones (16) se viola, entonces no existe una continuación factible. Ir al paso 5. De otra manera ir al paso 3.

Paso 3. a) Si todas las relaciones (16) se satisfacen como una estricta desigualdad, se determina cuál variable libre del conjunto N, al tener el valor igual

a uno, reduce considerablemente la infactibilidad total. Por infactibilidad total se entiende la suma, en valor absoluto de las cantidades en las cuales se viola cada restricción. Sea esta variable la $X_{j_{p+1}} \in N$. Entonces $X_{j_{p+1}} = 1$. Se ha maximizado la siguiente expresión.

$$\sum_{i=1}^{m} (b_i - \sum_{k=1}^{p+1} a_{ij_k} X_{j_k})$$

La nueva solución parcial es $(j_1, j_2, \ldots j_p, j_{p+1}++,)$. Ir al paso 1.

b) Si para alguna restricción (16), esta se comporta como una igualdad, se denota por **F** al conjunto de variables libres de esa restricción que tengan $a_{ij} < 0$. Se examina la relación

$$\sum_{j \in F} c_j < Z^* - Z \qquad (17)$$

Ir al paso 4.

Paso 4. a) Si (17) se satisface, entonces $X_{j=1}$, $j \in F$ es la única continuación factible que es posible (dada la solución parcial actual). La siguiente solución parcial que se debe analizar es:

$(j_1, \ldots, j_p, j_{p+1} +, \ldots, j_{p+q} +)$,

en que

$j_{p+1}, \ldots, j_{p+q} \in \mathbf{F}$

Regresar al paso 1.

b) Si (17) no se satisface, entonces no existe una posible continuación de la solución parcial. Ir al paso 5.

Paso 5. a) Dada la solución parcial actual encontrar el elemento j_k++ a la extrema derecha y eliminar a todos los elementos que se encuentren a su derecha. Se reemplaza este elemento (el j_k++) por j_k-. Esta es la nueva solución parcial. Regresar al paso 1.

b) Si no existe un elemento j_k++ en la solución parcial actual, le enumeración implícita se ha completado y la solución que se tiene en el paso 1 (a) es óptima. Si $Z^* = \infty$ o M, el problema original es inconsistente y por lo tanto no tiene solución.

2.4.3 Algoritmo aditivo generalizado de Balas.

El algoritmo anterior fue generalizado por Zionts[3] [1968] en el sentido de que puede trabajar restricciones de igualdad y acelera el proceso de convergencia a la solución óptima. Esta generalización agrega una restricción adicional de la forma.

$$\sum_{j=1} c_j X_j \leq Z^* - E$$

donde $0 \leq E \leq \text{Min}\{c_j\}$ y $Z^* = \infty$ o bien, $Z^* = \sum_{j+1}^{n} c_j$. Esta restricción tiene por objeto incluir a la función objetivo dentro del sistema de revisión que utilizan las reglas de algoritmo de Balas con el objeto de acelerar la convergencia del mismo. Este tipo de restricciones, llamadas de reemplazo se discuten a continuación.

El algoritmo aditivo generalizado es parecido al anterior, en donde sólo los pasos 2 y 3 se cambian y el 4 se suprime (pasos 1 y 5 son idénticos). Se dan los cambios a continuación:

Paso 2. Se generan cotas inferiores y superiores para cada variable cero-uno en cada restricción del problema, de la siguiente manera. Sea:

$$a_{ij}^{+} = \begin{cases} a_{ij}, & si\ a_{ij} \geq 0, \\ 0, & si\ a_{ij} < 0, \end{cases} \qquad a_{ij}^{-} = \begin{cases} 0, & si\ a_{ij} \geq 0 \\ a_{ij}, & si\ a_{ij} < 0 \end{cases}$$

Entonces para a_{ij}^{+} una cota inferior h_j de X_j se calcula por medio de

$$h_j = \min\left\{0, \left\lceil \frac{b_i}{a_{ij}^{+}} - \frac{1}{a_{ij}^{+}} \sum_{k \neq j} a_{ik}^{+} u_k - \frac{1}{a_{ij}^{+}} \sum_{k \neq j} a_{ik}^{-} h_k \right\rceil \right\} \quad j = 1,..,n,$$

donde h_k y u_k son la cota inferior y superior, respectivamente, de X_k. Una cota superior u_j de X_j se calcula de

$$u_j = \left\lfloor \frac{b_i}{a_{ij}^{+}} - \frac{1}{a_{ij}^{+}} \sum_{k \neq j} a_{ik}^{+} h_k - \frac{1}{a_{ij}^{+}} \sum_{k \neq j} a_{ik}^{-} u_k \right\rfloor \quad j = 1,..,n,$$

para $a_{ij}^{-} < 0$, la cota inferior h_j de X_j se calcula de

---

[3] Zionts S., *On an algorithm for the solution of mixed integer programming problems*. Management Science, Vol. 15, 1968. pp. 113 - 116.

$$h_j = \min \left\{ 0, \left\langle \frac{b_i}{a_{ij}^+} - \frac{1}{a_{ij}^+} \sum_{k \neq j} a_{ik}^+ h_k - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik}^- u_k \right\rangle \right\},$$

y la cota superior u_j de X_j se calcula de

$$u_j = \left[\frac{b_i}{a_{ij}^-} - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik}^+ u_k - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik}^- h_k \right].$$

En todas las formulas anteriores se tiene $h_k = 0$ y $u_k = 1$ para $k = 1, 2, \ldots, n$, por tratarse de variables binarias.

En los cálculos anteriores, los símbolos [] y $< >$ denotan lo siguiente:

$$[X] = \text{Máx } Y \leq X, \ Y \text{ entero.}$$
$$<X> = \text{Mín } Y \geq X, \ Y \text{ entero.}$$

a) Si una cota inferior h_k es mayor que cero (pero menor o igual a uno), se hace $X_k = 1$ en todas las continuaciones de la solución parcial actual. Por lo tanto se agrega $k+$ a la solución parcial actual. Regresar al paso 1.

b) Si $h_k > 1$, entonces no existe una continuación factible de la solución parcial actual. Ir al paso 5.

c) Si $0 \leq u_k < 1$, entonces $X_k = 0$ para todas las continuaciones de la solución parcial actual. Se agrega $k-$ a la solución parcial actual. Regresar al paso 1.

d) Si $u_k < 0$, entonces no existe una continuación factible para la solución parcial actual. Ir al paso 5.

e) Si todas las cotas inferiores son a lo máximo cero y todas las superiores son por lo menos uno, entonces no existen cotas más restringidas. Ir al paso 3.

Paso 3. Se determina cuál variable libre daría la mayor reducción de la infactibilidad total (que es la suma de valores absolutos de las cantidades en las cuales se están violando las restricciones con la solución parcial actual). En otras palabras, si se escoge j_{p+1}, se hace $X_{j_{p+1}} = 1$ y se minimiza la siguiente expresión

$$\sum_{i \notin E} \left| (b_i - \sum_{k=1}^{p+1} a_{ij_k} X_{j_k}) \right| + \sum_{i \in E} \left| \sum_{k=1}^{p+1} a_{ij_k} X_{j_k} - b_i \right|$$

donde **E** es el conjunto de restricciones de igualdad. La nueva solución parcial es $(j_1, \ldots, j_p, j_{p+1}++)$. Regresar al paso 1.

Así en el ejemplo anterior (en forma abreviada) se encontraría que en el paso 2 del algoritmo *aditivo generalizado de Balas*, la cota inferior de X_j es igual a uno (es decir $h_j = 1$) en la tercera restricción. Esto origina la solución parcial (3+). Con esta configuración se encuentra, aplicando el paso 2, que de la segunda restricción, la cota inferior h_2 de X_2 es uno, lo que genera la segunda solución parcial de (3+,2+). Como esta solución es factible, se tiene que $Z = 17$. De aquí se demuestra que (3+, 2-) no es factible y que (3-) tampoco lo es, por lo que (3+,2+) es óptimo.

2.4.4 Restricciones de reemplazo

Se supone que durante la aplicación del algoritmo *aditivo de balas* se encuentra una primera solución parcial factible que genera un valor de la función objetivo de Z^*. Con objeto de eliminar implícitamente otras soluciones parciales factibles, que pudieran generar valores de la función objetivo mayores a Z^*, se agrega la siguiente restricción

$$\sum_{j=1}^{n} c_j X_j \leq Z^*$$

Esta restricción tiende a acelerar el proceso, ya que evita buscar soluciones factibles que rinden peores resultados a los ya obtenidos.

Una *restricción de reemplazo* es una restricción del tipo $\mathbf{Y^T AX \leq Y^T b}$, donde $\mathbf{AX \leq b}$ son las restricciones originales y $\mathbf{Y \geq 0}$ es un vector de orden apropiado.

Dadas dos restricciones de reemplazo $\mathbf{a_0X \leq b_0}$ y $\mathbf{a_1X \leq b_1}$ en un proceso de minimización, se dice que la primera *domina* a la segunda si el valor asociado a la función objetivo utilizando sólo $\mathbf{a_0X \leq b_0}$, es mayor que el valor de la función objetivo utilizando sólo $\mathbf{a_1X \leq b_1}$.

Dado un problema cero-uno (binario).

$$\text{Mín } Z = \mathbf{cX}$$

sujeto a

$$\mathbf{AX \leq b}$$

con $\mathbf{X}$ un vector binario de ceros y unos, se entiende por su *continuación análoga*, al siguiente programa lineal.

$$\text{Mín } Z = \mathbf{cX}$$

sujeto a

$$\begin{aligned} \mathbf{AX} &\leq \mathbf{b} \\ \mathbf{0} \leq \mathbf{X} &\leq \mathbf{1} \end{aligned} \qquad (17)$$

Teorema 2. Dado (17), la solución óptima dual genera multiplicadores (variables duales) que

se pueden utilizar para construir a la restricción de reemplazo más dominante.

Prueba. Dado (17).

$$\text{Máx } -\mathbf{cX}$$

sujeto a

$$\mathbf{AX} \leq \mathbf{b}$$
$$\mathbf{X} \leq \mathbf{1}$$
$$\mathbf{X} \geq \mathbf{0},$$

su estructura dual es

$$\text{Mín } \mathbf{b}^t\mathbf{Y} + \mathbf{l}^t\mathbf{W}$$

sujeto a

$$\mathbf{A}^t\mathbf{Y} + \mathbf{W} \geq -\mathbf{c}^t \qquad (18)$$
$$\mathbf{Y} \geq \mathbf{0},\ \mathbf{W} \geq \mathbf{0}.$$

Sea la solución óptima del dual los vectores $\mathbf{Y}^*$, $\mathbf{W}^*$, que generan un valor óptimo de la función objetivo dual y de la función objetivo primal de

$$\mathbf{b}^t\mathbf{Y}^* + \mathbf{l}^t\mathbf{W}^* = Z^*.$$

La restricción de reemplazo correspondiente, de acuerdo a la definición es

$$\mathbf{Y}^{*t}\mathbf{AX} + \mathbf{W}^{*t}\mathbf{X} \leq \mathbf{Y}^{*t}\mathbf{b} + \mathbf{W}^{*t}\mathbf{l} = Z^*.$$

Se debe probar a continuación que la solución óptima de

$$\text{Máx } -\mathbf{cX}$$

sujeto a

$$(\mathbf{Y}^t\mathbf{A} + \mathbf{W}^t)\mathbf{X} \leq (\mathbf{Y}^t\mathbf{b} + \mathbf{W}^t\mathbf{l}) \qquad (19)$$
$$\mathbf{X} \geq \mathbf{0}$$

es mínima para $\mathbf{Y} = \mathbf{Y}^*$ y $\mathbf{W} = \mathbf{W}^*$.

Si $\mathbf{X}^*$ es óptima para (17), por el teorema fundamental de la dualidad se tiene que

$$-\mathbf{cX}^* = (\mathbf{Y}^{*t}\mathbf{A} + \mathbf{W}^{*t})\mathbf{X}^* = \mathbf{Y}^{*t}\mathbf{b} + \mathbf{W}^{*t}\mathbf{l} = Z^*.$$

Por lo tanto, se puede escribir para cualquier vector $\mathbf{X}$ lo siguiente:

$$-\mathbf{cX} \leq (\mathbf{Y}^{*t}\mathbf{A} + \mathbf{W}^{*t})\mathbf{X} \leq \mathbf{Y}^{*t}\mathbf{b} + \mathbf{W}^{*t}\mathbf{l}.$$

La solución óptima de (17) dados $\mathbf{Y}^*$ y $\mathbf{W}^*$ es $\mathbf{X}^*$. Para cualquier otra restricción de reemplazo generada por los multiplicadores duales $\mathbf{Y}$, $\mathbf{W}$, cualquier solución factible a (17), incluyendo $\mathbf{X}^*$, es factible. Además, algunas soluciones que no son factibles para (17), lo pueden ser para (19). Por lo tanto, cualquier solución óptima para (19), puede ser en este último caso mayor que $-\mathbf{cX}^*$, pero nunca menor. El teorema queda probado.

2.4.5 Efectividad de los cálculos por enumeración implícita

La efectividad del algoritmo aditivo es altamente dependiente de las fuerzas de las pruebas (de exclusión). Desafortunadamente, estas pruebas no son suficientes para producir un algoritmo computacionalmente eficiente. Los paquetes de computadora que tienen éxito se basan en pruebas mucho más fuertes. Quizá la más efectiva de ellas es la llamada *restricción delegada* (o sustituta). Se observa que el algoritmo aditivo examina las restricciones una a la vez. La idea es entonces desarrollar una restricción que *combine* todas las restricciones originales del problema en una restricción y no elimine ninguno de los puntos (enteros) factibles originales del problema. Se desarrolla la nueva restricción de manera que tenga el potencial de revelar información que no puede transmitirse por ninguna de las restricciones originales consideradas separadamente.

Las experiencias de cómputo reportadas indican que el uso de la restricción delegada mejora efectivamente el tiempo de cómputo. Sin embargo, debido a que la enumeración implícita investiga (implícita a explícitamente) todos los puntos binarios, el tiempo de solución varia casi exponencialmente con el número de variables n. Esto limita el número de variables que puede ser manejado por este método. Aunque se han reportado casos con éxito para problemas grandes (teniendo estructuras especiales), es seguro concluir que, en general, únicamente pueden resolverse problemas hasta con 100 variables en una cantidad razonable de tiempo de cómputo.

Una observación particular acerca de la enumeración implícita es que el tiempo de cómputo depende de los datos. El ordenamiento especifico de las variables y restricciones puede tener un efecto directo sobre la eficiencia del algoritmo. Por ejemplo, las restricciones deberán estar ordenadas con la más restrictiva en la parte superior, mientras que las variables podrían ordenarse según el orden ascendente de sus coeficientes en la función objetivo (coeficientes no negativos). Ambas condiciones son favorables para producir el examen más rápido de las soluciones parciales.

La conclusión es que el método de enumeración implícita todavía no proporciona la solución perfecta al problema de cómputo, por lo que los métodos de ramificar y acotar continuarán siendo los más eficaces, particularmente cuando se intenta la resolución de problemas prácticos de gran tamaño.

2.5 Conclusiones.

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. Las técnicas de planos de corte, de ramificación y acotamiento y de enumeración implícita permiten resolver problemas *difíciles* usando los métodos existentes para la resolución de problemas *fáciles*.

Particularmente, estos algoritmos se han formulado para resolver una amplia variedad de problemas de optimización combinatoria, entendiendo por problema combinatorio a aquel que asigna valores numéricos discretos a algún conjunto de restricciones y minimice alguna

función objetivo. Dos de tales problemas, el del agente viajero y el de la mochila, en los que la función objetivo puede ser no lineal, discontinua o no necesariamente definida matemáticamente, se pueden resolver con ramificación y acotamiento o enumeración implícita.

Dichos problemas, el del agente viajero o el de la mochila, se conocen como problemas no polinomiales completos o NP-completos, por lo que al aplicar alguna técnica de ramificar y acotar, o de enumeración implícita el algoritmo que utiliza adopta un tiempo de computadora exponencial.

Aunque se han desarrollado varios algoritmos finitos para los problemas combinatorios, ninguno de estos métodos es uniformemente eficiente desde el punto de vista computacional, principalmente cuando aumenta el tamaño del problema.

A diferencia de los programas lineales, donde problemas muy grandes se han resuelto en un tiempo razonable, los algoritmos que se aplican a problemas combinatorios han sido erráticos en el comportamiento de resolución.

El procesamiento de un algoritmo para este tipo de problemas, utilizando programas de cómputo normalmente genera errores de redondeo. En otros casos, por la naturaleza misma del algoritmo la convergencia hacia la solución correcta es extremádamente lenta. Ello ha motivado la creación de nuevos algoritmos basados en procedimientos *heurísticos*, los cuales en los últimos años han cobrado un importante auge.

En el siguiente capítulo se analiza en que consiste el procedimiento heurístico, su importancia en el desarrollo de nuevas técnicas de optimización, algunos ejemplos y sus actuales perspectivas.

CAPITULO III

TECNICAS BASADAS EN PROCEDIMIENTOS HEURISTICOS.

3.1 Introducción

Normalmente se presentan en la práctica problemas de programación que pueden ser tan complejos que los modelos que se construyan para abordarlos no se puedan resolver mediante los algoritmos tradicionales que se revisaron en el capítulo anterior. Esto se presenta cuando el problema es muy grande o demasiado complejo en el aspecto lógico (por ejemplo que requiera de muchas variables 0 - 1 en su formulación), y se considera que la imposición de supuestos simplificadores o aproximaciones que podían hacer más manejable el problema, provoca que la estructura del mismo cambie.

Resolver el problema dentro de este contexto suele ser tan complicado que, aunque exista una solución rigurosa (punto óptimo que maximice o minimice una función objetivo y que satisfaga toda una serie de restricciones) es difícil y quizá imposible describirla con la tecnología existente y sabiendo como hacerlo. En tales casos, se podría emplear un *algoritmo heurístico.*

Un algoritmo heurístico es aquel que produce con eficiencia buenas soluciones aproximadas para un problema. Con frecuencia (lo que no quiere decir que siempre), cuando se emplea dicho algoritmo se debe medir con precisión la bondad de la aproximación. Por ejemplo, en un contexto de optimización con algún algoritmo heurístico, se puede decir que el resultado obtenido está dentro de un determinado porcentaje de optimalidad, o que bajo cierta hipótesis, la respuesta heurística será óptima en un determinado porcentaje de veces.

En términos generales se puede decir que una *heurística* es una apelación intuitiva a una regla interna o particular para trabajar con un aspecto del problema que se esté resolviendo. Un programa heurístico es por tanto, un grupo de heurísticas o algoritmos heurísticos. Su aplicación principal es frecuente, y a la vez ideal, en los programas de cómputo de programación lineal. Por ejemplo, algunos programas de computación utilizan la heurística en la fase I del método simplex para tratar de encontrar con rapidez el vértice inicial. También se emplea la heurísica para obtener una iniciación rápida en el algoritmo del transporte y en casi todos los algoritmos de programación entera y combinatoria del tipo lineal.

En el contexto de la programación matemática, a menudo se emplea la heurística en conjunción con estrategias de resolución de problemas más rigurosas o generales, o como caso particular de ellas. El punto importante es que un procedimiento o algoritmo heurístico recurre a la intuición, pero puede garantizar sus resultados, si los hay, solo estadísticamente o dentro de ciertos márgenes de incertidumbre. Se emplea sobre todo por su eficiencia para producir con rapidez y con la certeza de que sean buenos, si no óptimos, los resultados.

Para explicar como funciona un procedimiento *heurístico* se plantea el siguiente ejemplo.

3.2 Aplicación de un método heurístico al problema de programación de instalaciones.

Se supone un solo medio de producción, a través del cual se deben procesar numerosos trabajos. Por lo general, el medio de producción tiene que detenerse después de ejecutar un trabajo, con el objeto de preparar el siguiente. Dicho *tiempo muerto* se llama *tiempo de instalación* o *por cambio*. Su duración puede depender del siguente trabajo que se va a procesar y del que se acaba de completar. Una sucesión de trabajos semejantes se interrumpiría durante menos tiempo por cambio que una secuencia de trabajos heterogéneos. Un problema típico consistiría en *secuenciar los trabajos de tal modo que el tiempo total de instalación se minimice.*

Desde el punto de vista combinatorio esto puede ser un problema difícil. Si solo hay tres trabajos por ejecutar, por ejemplo, A, B y C, se puede empezar por cualquiera de los tres, continuar con uno de los que restan y el tercero queda determinado (es decir el trabajo que queda). Las sucesiones posibles se pueden distribuir como un árbol en el que cada arco representa una sucesión. Las seis posibilidades se observan en la figura 1. En general, con n trabajos hay $n! = n(n - 1)(n - 2)...1$ combinaciones posibles o secuencias. Con solo 10 trabajos se producen $10! = 3{,}628{,}800$ sucesiones diferentes. Se puede advertir que el número de sucesiones de trabajos posibles ($n!$) aumenta rápidamente según la magnitud de n.

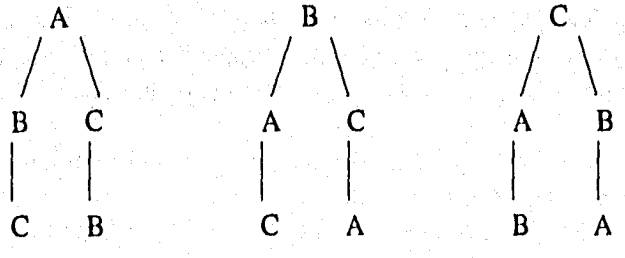

Figura 1
Arbol que muestra seis secuencias
posibles para los tres trabajos A, B y C

Una forma de resolver el problema anterior es mediante la enumeración integral. Es decir, producir cada una de las secuencias posibles de trabajos y calcular el tiempo total de instalación con ellas. Después, se escoge la sucesión asociada con el tiempo mínimo total. Aunque este algoritmo produciría un óptimo verdadero, no es práctico ni siquiera para valores modestos de n debido al gran número de secuenciasque se tendrían que enumerar.

En estos problemas, a menudo se aplican reglas heurísticas, aunque no garanticen una solución óptima, porque en general, conducen con suficiente rapidez a una solución satisfactoria.

Por ejemplo, se supone a un operador de computadora que tiene tres corridas largas por hacer en un determinado tiempo. La computadora está ociosa en la actualidad. Para cada uno de estos trabajos hay un tiempo de instalación indicado en la siguiente tabla.

Tabla 1

Tiempo de instalación en minutos

Al trab / Del trab	A	B	C
0	27	21	32
A		35	22
B	49		46
C	46	12	

Dado que sólo hay 3! = 6 secuencias posibles, se pueden enumerar todas. Los resultados aparecen en el cuadro 2, donde se observa que la sucesión óptima (tiempo mínimo de instalaciones) es 0→A→C→B.

Tabla 2

Resultados de la enumeración completa

TIEMPO DE SECUENCIA	INSTALACION	TOTAL (MIN)
0 → A → B → C	27 + 35 + 46	108
0 → A → C → B	27 + 22 + 12	61
0 → B → C → A	21 + 46 + 46	113
0 → B → A → C	21 + 49 + 22	92
0 → C → A → B	32 + 46 + 35	113
0 → C → B → A	32 + 12 + 49	93

Para ejemplificar un procedimiento heurístico para este problema se utiliza el método llamado *el mejor sucesor* o también *algoritmo glotón.* El procedimiento se ejecuta como sigue:

1. En el paso 1 (es decir, para el primer trabajo), se realiza la tarea de menor tiempo de instalación inicial.

2. En cada etapa subsecuente se elige la tarea que tenga el tiempo mínimo por cambio, basándose en el estado actual.

Con base en los datos de la Tabla 1, se observa que el trabajo con el menor tiempo de instalación inicial es B. Por lo tanto, el primer paso es 0 → B. De acuerdo con el algoritmo glotón el trabajo que seguiría sería C ya que el movimiento de B → C es menor que para B → A. En consecuencia, se tiene 0 → B → C y en seguida solo se puede concluir añadiendo A. Por lo tanto, se obtiene

heurística glotona $0 \rightarrow B \rightarrow C \rightarrow A$

tiempo total de instalación $= 21 + 46 + 46 = 113$

Como puede observarse este resultado estamuy lejos del óptimo. En efecto, en este ejemplo, el algoritmo glotón, aunque apela a la intuición, proporciona un mal resultado para el problema. Sin embargo, aplicar esta regla es muy fácil y los estudios de este tipo de problemas han demostrado que, estadisticamente, la regla no es mala para el tipo anterior de problemas de secuenciación. Por ejemplo, J.W Gavett[1] [1985] demuestra que la heurística produce con frecuencia mejores resultados de los que que se podrían obtener mediante una selección aleatoria de tareas. En el mismo artículo se demuestra que la siguiente heurística modificada da resultados aun mejores.

Tabla 3
Datos transformados

	A	B	C
0	0	9	10
A		23	0
B	22		24
C	19	0	

[1] J.W. Gavett, "Three heuristic rules for sequencing jobs to a single production facillity", *Management Science 11, 1985, Pags. B166-76*

1. Transformar los datos originales del cuadro 1 restando a todos los datos de una columna el menor de los tiempos de instalación que aparezca en esa columna. Esto produce los resultados presentados en la tabla 3.

2. Aplicar el algoritmo glotón a este conjunto trasformado de datos. Al hacerlo se obtiene

Primer etapa mejor $0 \rightarrow A$

Segunda etapa mejor $A \rightarrow C$
Tercera etapa $C \rightarrow B$

y así la heurística modificada produce la secuencia $0 \rightarrow A \rightarrow C \rightarrow B$, de la que ya se demostró que es óptima para este problema.

Aunque esta heurística modificada no siempre produce la solución óptima, es fácil de implantar y en la práctica para problemas extensos, con frecuencia produce buenos resultados.

3.3 Aplicaciones heurísticas a problemas combinatorios.

Para comprender como se procede hurísticamente en la solución de problemas combinatorios, se presenta a continuación una estrategia de este tipo para resolver el problema de *secuenciación de rutas* y el de la *determinación del tamaño óptimo de una flota de vehículos*.

3.3.1 Problema de secuenciación de rutas

El problema de secuenciación de rutas, fue introducido en la literatura por Dantzig y Ramser[2], y modelado como problemas enteros binarios (tipo cero-uno) por Balinski y Quandt[3] [1964] así como por Garvin, Crandall, John y Spellan[4] [1967]. Aquí se presenta un método de solución ideado por Clarke y Wright[5] [1963].

---

2 Dantzing G. y Ramser J.H. *The truck dispatching problem.* Management Science, Vol. 6. 1959. pp 80 - 89.

3 Balinski M.L. y Quandt R.E. *On an integer program for a delivery problem.* Operation Research, Vol. 12. 1964. pp 300 - 306.

4 Garvin W.W., Crandall H.W., John J.B. y Spellman R.A. *Applications of linear programming in the oil industry.* Management Science, Vol. 13. 1967. pp. 407 416.

5 Clarke G. y Wright J.W. *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Reseachs. Vol. 11. 1963 pp. 568 - 582.

El problema de secuenciación de rutas se refiere a un conjunto de clientes, todos con dirección y demanda de servicio de un sólo producto conocidos. A todos estos clientes se les sirve desde un sólo punto, de donde se despachan una serie de vehículos.

El problema consiste en diseñar a costo mínimo rutas de estos vehículos basados en las siguientes restricciones:

a) Se debe satisfacer la demanda de servicios por unidad de tiempo de cada cliente.

b) No se puede exceder la capacidad de carga de cada vehículo.

c) El tiempo total de servicio o bien la distancia total de recorrido, no debe exceder de una cantidad prefijada. Esto suele suceder cuando se tienen restricciones de tipo legal o sindical.

d) Existe un rango de tiempo en el cual el cliente debe ser atendido.

Varios problemas de esta naturaleza pueden resolverse utilizando los métodos anteriormente descritos. Estos pueden ser:

a) Si la flotilla de vehículos consiste en un sólo vehículo con capacidad de carga bastante grande, tal que se puedan ignorar las restricciones (b), (c) y (d) dadas en el párrafo anterior, el problema se convierte en el del *agente viajero*, que ya se indicó en el capítulo anterior.

b) Si se ignoran las restricciones (c) y (d) y se trata de encontrar el número mínimo de vehículos que cumplan con los requisitos (a) y (b), entonces se tiene un problema con la estructura *mochila*, que ta también se discutió en el capítulo anterior. A este tipo de problemas se les llama de *carga de vehículos*.

c) Dado un número fijo de vehículos, que sean compatibles con las restricciones (a), (b), (c) y (d) se requiere diseñar la secuenciación de los mismos que darán origen a rutas de costo mínimo. A este tipo de problemas se les llama de *secuenciación de rutas*.

d) Si la localización de los clientes y su demanda permanece constante en todos los períodos de tiempo futuros, se trata de encontrar el tamaño de la flota, que siendo compatible con las restricciones (a), (b), (c) y (d), tenga el mínimo costo. A este tipo de problemas se les llama de *determinación del tamaño de la flota de vehículos*.

La formulación matemática del problema es la siguiente. Sea N el número total de clientes a satisfacer, y se les numera del 1 al N. El cliente i, $i = 1,2,\ldots,N$ requiere de $q(i)$ unidades (por unidad de tiempo) del único artículo que se distribuye y la distancia entre el cliente i y cliente

j, $i \neq j$, $i, j = 1,2,...N$, es d_{ij}, con $d_{ij} = d_{ji}$, considera, por claridad de exposición, que el cliente número 1 es el punto de partida de todos los vehículos.

En el punto de partida existe un número no especificado (bastante grande) de vehículos, todos con capacidad de carga de Q unidades. Por cuestiones de tipo sindical un vehículo no puede recorrer en una ruta más de M kilómetros. Una secuenciación específica k rutas, $R_1, R_2,...,R_k$, en que cada una empieza en el punto de partida (localización del cliente 1), sirve a un numero de clientes y regresa al punto de partida. El objetivo es determinar el mínimo número de rutas que satisfagan todas las restricciones, tal que el costo total de recorrido (o kilometraje total del recorrido) sea mínimo.

Si la ruta R_i, $i = 1,2,..,N$ tiene n_i clientes que se denotan por $r_i(1), r_i(2), ..., r_i(n_i)$, entonces el recorrido total para la ruta R_i será

$$D_i = d(1, r_i(1)) + \sum_{j=2}^{n_i} d(r_i(j-1)), r_i(j) + d(r_i(n_i), 1),$$

y el recorrido total para las k rutas será de

$$\sum_{i=1}^{k} D_i$$

Las variables del problema son:

a) los clientes $r_i(j)$ en una ruta,
b) el número n_i de clientes en la ruta R_i y
c) el número k de diferentes rutas.

Se quiere minimizar el número k y el número $\sum_{i=1}^{k} D_i$, tal que se respeten las disposiciones sindicales

$$D_i \leq M, \qquad i = 1, ...,k.$$

la capacidad de carga

$$\sum_{j=1}^{n_i} q(r_i(j)) \leq Q, \; i = 1, ...,k.$$

se supone que

$$Q \geq q(i) \quad i = 1,...,k.$$

Para resolver este problema, Clarke y Wright, definen un *criterio de ahorro s(i,j)* dada por

$$s(i,j) = d(1,j) + d(1,i) - d(i,j).$$

En palabras, este criterio de ahorro mide el kilometraje ahorrado si una ruta que originalmente acaba en el punto i, ahora también incluye al punto j.

El algoritmo de Clarke y Wright[6] se desarrolla de la siguiente forma:

Paso 1. Se calcula el ahorro $s(i,j)$, para todas las parejas de clientes $i, j = 1, 2, \ldots, N$, $i \neq j$.

Paso 2. Se ordenan todos los a horros $s(i,j)$ en orden descendente de magnitud.

Paso 3. Empezando con el primero de la lista se hace lo siguiente:

3.1 Si al unirse dos clientes por un tramo de la ruta, esta resulta ser factible y respeta a todas las restricciones del problema, entonces se agrega este tramo a la solución. Si no, se rechaza. Para una ruta factible se eliminan todos los tramos no analizados que conectan con este tramo.

3.2 Se toma la siguiente pareja de la lista del paso 2, y se repite el paso 3.1. Se continúa con este paso (3.2) hasta terminar de analizar todos los elementos de la lista del paso 2.

Paso 4. Se unen todos los tramos en orden $(1,j)$, (j,k), (k,p), ..., $(m,1)$, a fin de formar todas las posibles rutas.

Este algoritmo por lo general no produce una solución óptima, pero si una buena solución factible que está en torno del óptimo. Sin embargo debido a su simplicidad y rapidez la solución factible que genera este algoritmo, se puede considerar para fines prácticos como aceptable.

3.3.2 Problema de determinación del tamaño del una flota de vehículos.

Este tipo de problemas puede tener dos presentaciones. En la primera, se supone que todos los vehículos tienen la misma capacidad. El tamaño óptimo de la flota de vehículos se calcula minimizando los costos totales (que incluyen costo fijos y variables). La capacidad de los vehículos se toma como un parámetro, al cual una vez dado un valor determinado, se obtiene un tamaño de flota y un costo óptimo asociado Después se analizan todos los costos resultantes del proceso de parametrización y se escoge el mínimo de ellos. Eso indica de inmediato el

---

6 Clarke G. y Wright J.W. *Op. Cit.* 1963 pp. 570

tamaño de flota, costo y capacidad de cada vehículo. Este resultado puede después ajustarse en función del producto que se va a distribuir (si es perecedero o no, si es líquido, gaseoso o sólido, etc.), accesibilidad de vías de comunicación, etc. En la segunda presentación de este problema, se relaja la restricción de todos los vehículos deben tener la misma capacidad de carga.

La literatura relacionada a estos problemas es bastante raquítica, habiéndose encontrado que solamente Gould y Wyatt[7] han hecho ciertos trabajos al respecto. Para formular el modelo que resuelve a este tipo de problemas se hacen las siguientes suposiciones:

a) La localización y demanda por ciclo, de cada cliente, es conocida y permanece constante, es decir, no cambia en cada ciclo.

b) Cada vehículo tiene asociado un costo fijo k por período de tiempo. El costo fijo ya incluye los costos de depreciación, salario de choferes, licencias, seguro y otros costos que no dependen del uso del vehículo. La flota consta de N vehículos propios, donde N es una variable de decisión.

c) Existe un costo variable V que es función del kilometraje recorrido. Este costo absorbe gasolina, lubricantes, mantenimiento, llantas, etc.

d) Si por algún motivo, la compañía de vehículos necesita alquilar vehículos ajenos, estos acarrean un costo fijo k' y un costo variable V', donde $k' > k$ y $V' > V$.

El objetivo del problema es determinar el tamaño de N tal que los costos totales se minimicen.

Para estructurar la función objetivo se supone que durante el período i, los vehículos propios de la compañía recorren d_i kilómetros, y que durante ese período fue necesario alquilar N_i vehículos ajenos que recorren una distancia total de d'_i. El costo total para la compañía durante el período i es

$$C_i = kN + Vd_i + K'N_i + V'd'_i$$

Si el horizonte de planeación tiene p períodos, el costo total para la compañía es

$$C = \sum_{i=1}^{p} C_i = kpN + V \sum_{i=1}^{p} d_i + K' \sum_{i=1}^{p} N_i + V' \sum_{i=1}^{p} d'_i$$

Las incógnitas en la formulación anterior son N, N_i, d_i, d'_i y se requiere encontrarles valores tales que minimicen el costo total C.

---

[7] Gould J. y Wyatt J.K. *The size and composition of a road transport fleet operational.* Research Quaterly, Vol. 20. 1969. pp. 81 - 92.

Se considera que durante el período i, se requiere un total de S_i vehículos, con los cuales se cubre un recorrido total de por lo menos D_i kilómetros, con objeto de satisfacer la demanda de los clientes durante ese período.

Como el método que se presenta no genera normalmente soluciones óptimas sino factibles en torno al óptimo, se define a continuación una cota superior e inferior de la solución óptima. Sean

s_{i1} el número mínimo de vehículos en el período i con los que se puede satisfacer la demanda de clientes durante el período i. A este número corresponde un recorrido de D_{i1} kilómetros. S_{i1} constituye una *cota inferior* de S_i. esta cota inferior depende de que la suposición $k' > k$ y $V' > V$ sea válida.

s_{i2} el número de vehículos en el período i (se calcula como resultado de un análisis de sensibilidad para métricas sobre las s_{i1})que general un recorrido total mínimo denotado por D_{i2}. S_{i2} constituye una *cota superior* de S_i. Esta cota superior es independiente de los parámetros de los costos.

Obviamente se tiene:

$$S_{i2} \geq S_i \geq S_{i1}.$$
$$D_{i2} \leq D_i \leq D_{i1}.$$

El resultado que genera el método para el período i es precisamente S_i (con su D_i correspondiente). Obviamente no son deseables valores menores a S_{i1}, puesto que ocasionan recorridos mayores a D_{i1}. Por otro lado, no se requieren recorridos totales mayores a D_{i2} porque generarían tamaños de flotas mayores a S_{i2}.

Para calcular los valores de S_{i1} se utilizan los métodos tipo *mochila* y para el cálculo de S_{i2} los métodos explicativos en la sección anterior.

Existen tres situaciones que se deben analizar:

a) $N < S_{i1}$,
b) $S_{i1} \leq N \leq S_{i2}$,
c) $S_{i2} \leq N$.

En el primer caso ($N < S_{i1}$), la flota propia de la compañía no es suficiente para abastecer a todos los clientes. ¿Entonces, cuántos vehículos deben alquilarse? La respuesta es un número que oscila entre una cota inferior de S_{i1}-N vehículos y una superior de S_{i2}-N vehículos. Paramétricamente utilizando una combinación de los problemas tipo *mochila* y el método de Clarke y Wright, se encuentra la respuesta deseada.

En el segundo caso ($S_{i1} \leq N \leq S_{i2}$), se tiene un número suficiente de vehículos propios para

satisfacer la demanda de los clientes. Sin embrago, eso no quiere decir que se tiene un costo de operación óptimo, pues se sabe que si se alquilan vehículos ajenos, los recorridos totales tienden a disminuir y por ende los costos asociados. Sin embargo, por otro lado, se incrementan los costos asociados al alquiler de vehículos. El costo en el entorno al costo óptimo deberá calcularse por un análisis paramétrico, utilizando una combinación del método *mochila* y el de Clarke-Wright, variando el valor de N en la forma

$$S_{i1},\ S_{i1} + 1,\ S_{i1} + 2,\ldots,S_{i2}.$$

En el tercer caso ($S_{i2} \leq N$) se tiene un número suficiente de vehículos propios para satisfacer la demanda de clientes, y si en efecto $k' > k$ y $V' > V$, no es necesario analizar el efecto de alquilar vehículos ajenos.

Una vez que se tenga un valor de N para el período i, se puede calcular C_i. Hecho esto para todos los p períodos se tiene un valor de C. Si este valor de C se encuentra en forma paramétrica para valores de $N = S_{i2}$ y después para valores de $N = S_{i2} - 1$, y después $N = S_{i2} - 2$, etc., hasta llegar a valores de $N = S_{i1}$, se escoge aquel valor de N que genera el mínimo costo total C. Si por algún motivo, la condición $V' > V$ y $k' > k$ no se cumple, el análisis paramétrico debe continuarse hasta el valor $N = 0$, inclusive.

3.4 Aspectos relevantes de la programación heurística.

Como en la mayoría de los modelos cuantitativos, es costumbre que los enfoques heurísticos sean implementados mediante un programa de computación. En la práctica una diferencia entre el uso de procedimientos heurísticos, en lo que se opone a modelos más formales como los que se manejan en la programación lineal o cuadrática, consiste en que el último caso ya existen programas de cómputo. No obstante en el caso heurístico a menudo la aplicación es *ad hoc*, lo que implica que deben elaborarse los programas. Una aplicación típica de la heurística es, como se ha establecido, el área de los problemas combinatorios extensos, para los cuales sería prohibitivo por lo costoso que es obtener una solución, ya sea por enumeración o aplicando un modelo matemático formal o de programación entera. En todas las aplicaciones de la heurística hay un criterio implícito del investigador en el que la *aceptabilidad*, en lugar de la *optimalidad*, es un modo adecuado de pensar. En otras palabras, se siente que las *buenas soluciones*, en oposición a *las soluciones óptimas*, pueden ser valiosas y satisfactorias. Esta filosofía encaja bien, en particular, en problemas que más bien son difíciles en su formación, tales como los problemas de alto nivel con objetivos sustitutos, o para los cuales hay numerosos criterios en conflicto de intereses y para los que, en consecuencia, no está definida con claridad una sola función objetivo.

En la práctica, el uso de la heurística está ligado, como se verá más adelante, al campo de la *inteligencia artificial* donde la computadora se programa con técnicas heurísticas.

En la implantación de modelos heurísticos, la iteración administrativa y la retroalimentación

desempeñan un papel quizá mayor que en el caso de la construcción de modelos más formales, ya que, en el caso heurístico, el unvestigador de operaciones debe evaluar no sólo el modelo, sino en forma implícita, también el algoritmo. Esto se debe a que, para el mismo modelo, heurísticas diferentes conducen a soluciones distintas.

Esta iteración estrecha entre el modelo y el que toma las decisiones se manifiesta también en la programación meta[8], cuando el que decide debe asignar prioridades a diversas metas, como en la forma de arreglo ordinal (o sea, de *prioridades absolutas*). La programación meta se basa en la intuición y en este sentido, es *heurística*, en su enfoque de problemas con objetivos múltiples. En la programación de metas con prioridades absolutas, el investigador debe considerar con cuidado la *importancia relativa* o *utilidad* de sus metas. Según el resultado del modelo, el que toma las decisiones puede querer cambiar las prioridades, o aun el número de metas, y volver a correr el modelo. En otras palabras, así como con la programación lineal, el análisis de sensibilidad viene a ser una parte importante de la implementación. Dado que la programación de metas está más o menos en la infancia, el campo se desarrolla a gran velocidad, desde el punto de vista teórico, y parece claro que esto impulsará un uso mayor de la técnica, en especial cuando el análisis de sensibilidad se considere con mayor importancia.

3.5.1 Tendencias actuales .

Como se verá en el siguiente capítulo, los problemas que caen dentro del ámbito de la *inteligencia artificial* son demasiado complejos para ser resueltos mediante técnicas directas. Mejor dicho, estos deben ser atacados a través de un método apropiado de búsqueda incluyendo todos los aspectos que están disponibles en las técnicas directas para guiar la búsqueda.

Los métodos de búsqueda heurística pueden describirse independientemente de cualquier tarea en particular o problema. Pero cuando se aplica a problemas particulares su eficacia depende ampliamente de la forma en que se explota el dominio de ciertos conocimientos y que por si mismos son incapaces de superar la explosión combinatoria para lo cual los procesos de búsqueda son demasiado vulnerables.

Por esta razón, esos métodos son frecuentemente llamados métodos débiles. Pese a sus limitaciones estas técnicas continúan proporcionando la estructura en la cual el dominio de ciertos conocimientos puede ser considerados como resultado de un *aprendizaje automático*. Algunas de las técnicas de búsqueda de este tipo que adoptan principios de inteligencia artificial son los siguientes:

---

8 La programación meta es un concepto introducido por A. Charnes y W. W. Cooper en *Management models and industrial applications of linear programming* (New York: John Wiley & Sons, Inc., 1961) y se aplica en general a problemas lineales; es una extensión de la PL que permite al investigador acercarse lo más posible a la satisfacción de metas y restricciones diversas. Permite a quien toma las decisiones, al menos en el sentido heurístico, incorporar su sistema de preferencias al trabajar con metas múltiples en conflicto. A veces se considera como un intento de poner en el contexto de la programación matemática el concepto de *satisfacción. Este término fue sido acuñado para comunicar la idea de que, a menudo, los individuos no buscan soluciones óptimas, sino más bien quieren soluciones que sean suficientemente buenas o bastante proximas.*

- Generación y prueba
- Escalamiento (Hill climbing)
- Búsqueda del Mejor-primero
- Reducción del problema
- Satisfacción de restricciones
- Análisis medios-finales.

Estas *heurísticas* son criterios, métodos o principios para decidir sobre varias alternativas de cursos de acción que prometen ser lo más efectivo en orden a alcanzar alguna meta. Estas representan la atención a dos requerimientos fundamentales: la necesidad de hacer más simple el criterio de selección y al mismo tiempo que sean capaces de discriminar correctamente entre buenas y malas elecciones.

En conclusión, el papel de las *heurísticas* es el de enfocar la atención sobre las estrategias más promisorias e intentar encontrar una solución sin explotar todas las posibles alternativas.

CAPITULO IV.

MÉTODOS DE BÚSQUEDA HEURÍSTICA PARA PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA BASADOS EN LOS PRICIPIOS DE INTELIGENCIA ARTIFICIAL

4.1 INTRODUCCIÓN

No todos los modelos matemáticos en investigación de operaciones poseen algoritmos o métodos de solución que converjan siempre al nivel óptimo. Existen dos razones de ser de esta dificultad.

En primer lugar, se puede probar que el algoritmo de solución converge al nivel óptimo, pero solo en sentido técnico. La convergencia técnica señala que hay un límite superior finito para el número de iteraciones, pero no indica cuan alto puede ser este límite. Por lo tanto, se pueden consumir horas de tiempo de la computadora sin llegar a la iteración final, lo que es peor aún que si las iteraciones se detienen en forma prematura antes de llegar al nivel óptimo, generalmente no se puede medir la calidad de la solución obtenida en relación con el nivel óptimo verdadero.

Además, como se destacó en el capítulo I, en los modelos matemáticos el número de iteraciones es función de la eficacia del algoritmo de solución y la estructura específica del modelo y quizá el usuario no pueda controlar ninguno de estos factores.

En segundo lugar, la complejidad del modelo matemático puede hacer imposible idear un algoritmo de solución. En este caso el modelo se puede mantener infactible en término de cálculos.

Las dificultades evidentes en los cálculos de los modelos matemáticos ha obligado a los analistas a buscar otros métodos de cálculo. Estos métodos también son de naturaleza iterativa, pero no garantiza la optimalidad de la solución final. En cambio simplemente buscan una buena solución al problema. Tales métodos suelen denominarse *heurísticos* porque su lógica está basada en reglas o métodos prácticos que lleven a la obtención de una buena solución. La ventaja de los métodos heurísticos es que normalmente implican un menor número de cálculos cuando se comparan con algoritmos exactos. Asimismo, debido a que están basados en reglas prácticas, normalmente son más sencillos de explicar por los usuarios que no están orientados a las matemáticas.

En investigación de operaciones los métodos heurísticos suelen emplearse para dos fines:

1.- Se pueden utilizar dentro del contexto de un algoritmo de optimización exacto, con

el fin de aumentar la velocidad del proceso para alcanzar el nivel óptimo. La necesidad de "fortalecer" el algoritmo de optimización se hace más evidente con modelos a gran escala.

2.- Se utilizan simplemente para obtener una buena solución al problema. La solución resultante no tiene la garantía de ser óptima y de hecho, su calidad en relación con el nivel óptimo real puede ser difícil de determinar.

En la actualidad los avances tecnológicos en el área de informática han propiciado la creación de nuevas propuestas para la solución de problemas de optimización que se resuelven a partir de métodos heurísticos, basados lo que se ha dado llamar Inteligencia Artificial.

Tomando como base que la inteligencia artificial se apoya en ciertos mecanismos de procesamiento que los hace apropiados para la solución de problemas que requieren de técnicas heurísticas de búsqueda, basadas en operaciones y especialmente en mecanismos de razonamiento lógicos y analógicos, ha dado lugar a que en la actualidad se propicie un enlace bilateral entre investigación de operaciones y la inteligencia artificial[1].

Dirigido a una amplia posibilidad de situaciones, los métodos heurísticos basados en la inteligencia artificial permiten la búsqueda de soluciones a problemas financieros, administrativos, de inventarios, localización de instalaciones; y en la ingeniería, en casos como: análisis de datos meteorológicos, administración de recursos acuíferos, diseño de circuitos integrados, operación y monitoreo de satélites y sistemas de mantenimiento. Ejemplos similares pueden tomarse en economía, psicología y biología.

Dentro de toda una serie de nuevas técnicas, recientemente se ha puesto especial atención en cuatro métodos de manipulación en la solución de problemas complejos de decisión: **Algoritmos genéticos, redes neuronales, recocido simulado y búsqueda tabú.** Los dos primeros; algoritmos genéticos y redes neuronales, están inspirados por los principios derivados de las ciencias biológicas; el recocido simulado se deriva de las ciencias físicas, principalmente de la termodinámica. La búsqueda tabú se basa en las técnicas de solución de problemas inteligentes.

Cada uno de estos métodos guarda un enfoque diferente para la solución de un mismo problema. Algunos presentan, en ciertos casos soluciones más confiables que los otros, sin que ello demerite los procesos empleados en cada caso.

Los algoritmos genéticos por ejemplo, se basan en la noción de propagación de nuevas soluciones derivadas de soluciones **padres**, aplicando los supuestos de descendencia genética. La mejor descendencia de las soluciones padres es retenida para una siguiente generación, procediendo, de este modo, de una forma evolutiva que fomenta la supervivencia del más apto. Como una aptitud de la cualidad (mejor descendencia) se construye un entorno cada vez mas alto, compatible con la mejor solución global registrada.

---

[1] Glober, Fred y Harvey J. Greenberg. *New approaches for heuristic search: A bilateral linkage with artificial intelligence*. En European Journal of Operational Research. No. 39, 1989. pp. 119 - 130.

Los componentes del proceso de algoritmos genéticos pueden ser descritos dentro de tres encabezados: **reproducción, cruzamiento y mutación**. La *reproducción* es una muestra aleatoria de individuos de una población que crea uno o más descendientes suyos. El *cruzamiento* se define como el resultado del cambio de un gene, el cual especifica valores que son llamados alleles. Estos son las iteraciones de un sistema sensor experto. El cambio de genes sigue el tradicional modelo de reproducción biológico. Finalmente, la *mutación* es simplemente la introducción de un elemento aleatorio, frecuentemente utilizado para enmendar el resultado de un gene cambiado cuando el resultado no es el apropiado para las restricciones ofrecidas.

El método de **redes neurales** por su parte, supone la creación de un sistema de inteligencia artificial que tenga la flexibilidad, creatividad y la habilidad de aprendizaje del sistema biológico humano. La búsqueda para imitar este proceso humano ha estado enmarcado en la historia por varios años. Durante estos períodos de investigación, muchos modelos de sistema biológico inteligente han aparecido. En cada uno de ellos se ha tratado de diseñar sistemas que funcionen del mismo modo que el cerebro humano y su sistema nervioso. El modelo biológico inteligente que preocupa este capítulo es el llamado modelo del **redes neurales.** Este modelo realiza la representación de una neurona principal y su iteración con otras neuronas dentro del cerebro humano.

Asimismo, **el recocido simulado** ha sido considerado como una nueva y poderosa metodología para la solución de problemas combinatorios, con implicaciones en el campo de la inteligencia artificial. Más ampliamente aplicado para problemas de optimización que los métodos de redes neurales, recocido simulado es no obstante compatible con estos, y ha sido propuesto como un posible proceso alterno para la solución de este tipo de problemas.

El nombre de recocido simulado se deriva del intento por desarrollar una aproximación del proceso físico de recocido, el cual consiste en la reducción de temperatura hasta un mínimo o un estado establecido, llamado equilibrio termal.

Por su parte, el método de **búsqueda tabú** puede ser considerado como una técnica basada en algunos conceptos seleccionados de inteligencia artificial, que sigue un procedimiento heurístico general para guiar la búsqueda y obtener buenas soluciones en problemas de optimización combinatoria considerados NP difíciles. Además sus reglas son lo suficientemente amplias que son utilizadas con frecuencia para dirigir la operación de otros procedimientos heurísticos.

Estas técnicas son resultado de una propuesta de integración de la inteligencia artificial con investigación de operaciones que brinda una nueva estrategia para la solución de problemas combinatorios de optimización. En la realidad se puede utilizar cualquier estrategia convencional, tal como la enumeración implícita, y subordinar el control de parámetros a un modelos de aprendizaje que ejecute y resuelva problemas de clasificación.

El objetivo de este capítulo consiste en explicar de una manera sencilla en que consisten y cómo opera cada una de estas cuatro propuestas en su aplicación a problemas de optimización combinatoria, tomando como base el análisis que han realizado los principales precursores de estas propuestas.

También se hace una breve revisión de la literatura actual que ha emergido en el desarrollo de estas técnicas, resaltando los aspectos más relevantes de cada caso.

Dentro de este capítulo, inicialmente se hace un planteamiento de lo que es la inteligencia artificial, sus principios y su trascendencia dentro de la investigación de operaciones. Posteriormente se hace un breve planteamiento de cada una de las propuestas de búsqueda heurística que se indicaron en los párrafos anteriores, indicando en que consisten, cual es su base, su aplicación en algún problema de optimización combinatoria y los resultados que ha presentado.

4.2 LA INTELIGENCIA ARTIFICIAL Y SUS PRINCIPIOS

4.2.1 Concepto de inteligencia artificial.

La Inteligencia Artificial (IA) es una tecnología que se ocupa de la comprensión de la inteligencia y del diseño de sistemas inteligentes, entendiendo por tales aquellas que presentan características asociadas al entendimiento humano[2] como el razonamiento, la comprensión del lenguaje hablado y escrito, el aprendizaje, la toma de decisiones y otras similares.

La IA es un campo científico que surge a mediados de la década de los cincuenta, pero es a finales de los 80's cuando cobra un verdadero auge. Actualmente las expectativas son muy prometedoras en cuanto a que hay un número creciente de aplicaciones prácticas en los campos científicos, administrativos y educativos.

Aunque la IA se basa en programas de ordenador, al igual que lo que se podría llamar informática tradicional o convencional, los procedimientos de resolución de problemas y las técnicas de programación de IA presentan una serie de características que los hace diferentes de los de la informática convencional (IC) por las siguientes razones: En primer lugar, mientras que en la IC los objetos que manejan las máquinas son datos, de tipo numérico o alfa-numérico, en la IA los objetos son ideas y conocimientos, que se ajustan más a una descripción de tipo simbólico. En la IC los procesos a los que se someten los datos son de tipo algorítmico, perfectamente definidos y estructurados *a priori*, con una secuencia de operaciones predefinida y que se repite en su totalidad ante las mismas condiciones de partida. En IA, por el contrario, el proceso es de tipo lógico abierto, en el que el ordenador dispone de unas reglas de inferencia y de una base de conocimientos, y en función de ambas y de la información que adquiere, o se le suministra, inicia un proceso de búsqueda empírico o *heurístico* con notables dosis de *complejidad*, *incertidumbre* y ambigüedad propias.

Estos últimos que se mencionan encuentran cabida en los problemas de decisión con que se enfrentan las personas en el mundo real y sus resultados guardan una expectativa similar. Así como en el cálculo científico se requieren respuestas exactas y se buscan las mejores soluciones posibles, en IA *los programas no siempre garantizan respuestas correctas y se aceptan normalmente respuestas satisfactorias*, tal como sucede con los problemas humanos

---

[2] La IA, como tecnología que trata de producir sistemas inteligentes, se engloba dentro de la ciencia e ingeniería informática, y los métodos que utiliza no tienen que emular necesariamente el comportamiento de la mente humana.

en los que incluso ocurre a veces que las decisiones que se toman son equivocadas.

Un aspecto importante de los programas de IA es el dominio del área del conocimiento en que se van a presentar los problemas que habrán de resolverse. El dominio de conocimientos debe estar claramente acotado y estructurado para poder ser manejado. Por ello en IA es usual tener por separado los conocimientos y el mecanismo que controla la búsqueda de soluciones (al contrario de lo que ocurre con los programas convencionales). Como consecuencia los programas de IA son normalmente *fáciles de modificar, actualizar y ampliar*, ya que se puede cambiar la estructura de una instancia cambiando la base de conocimientos y utilizar el mismo método de búsqueda. Debido a esta organización y al hecho de que el sistema informático puede inferir nuevas reglas, en el caso de IA, el determinismo algorítmico que caracteriza a la programación convencional desaparece en buena parte.

Por otra parte, en la IA el programa indica al ordenador lo que debe hacer sin especificarle como debe hacerlo, por lo que en las diferencias que se establecen entre el "cómo" y el "porqué" de llegar a ciertas conclusiones y conocimientos están las características que separan a ambas formas de programación. Por lo tanto, lo que se llama *la representación del conocimiento* es un punto clave en los problemas de IA y es una de las líneas de investigación más activas en la actualidad. Cuando se quiere resolver un problema complejo, los conocimientos que se requieren manejar son numerosos y por ello se necesitan métodos eficientes para representarlos y manejarlos.

El otro punto clave en los problemas de IA es el del proceso de búsqueda de soluciones. La resolución de un problema supone frecuentemente una serie de *elecciones* entre varias alternativas hasta llegar a la meta final. Este método se representa mediante una estructura jerárquica de *árbol* y la solución sigue un camino desde el nodo de origen hasta el nodo de destino o meta final. Para problemas de gran complejidad resulta demasiado complicado explicitar todos los posibles caminos para llegar a determinar el que conduce a la mejor resolución.

Por tanto, con la aplicación de principios de IA se trata de encontrar *métodos de búsqueda eficientes que reduzcan el abanico de posibilidades empleando reglas empíricas o razonamientos que no consideren todas las ramas del árbol sino solamente las más probables.* Como consecuencia, el camino seguido y la solución encontrada pueden no ser los óptimos pero sí razonablemente aceptables con la ventaja de *reducir enormemente el proceso de búsqueda*. De esta forma los nuevos sistemas se aproximan más a la forma de comportarse de las personas que, en su proceder diario o en sus procesos de decisión, lejos de buscar respuestas precisas y soluciones exactas manejan reglas aproximadas y hechos imprecisos para alcanzar conclusiones útiles, y aceptables.

4.2.2 Aspectos fundamentales

En investigación de operaciones la propuesta basada en los principios de la Inteligencia Artificial se centra principalmente en la resolución de un cierto tipo de problemas. Existen dos procedimientos para la resolución de problemas: los algorítmicos y los heurísticos.

Los algorítmicos son procedimientos claramente definidos que *garantizan la solución de un problema*. Sin embargo, como ya se ha visto, existen problemas que debido a su excesiva complejidad dan lugar a algoritmos impracticables, en los cuales el ordenador más rápido tardaría años o incluso siglos en resolverlos.

La heurística es un conjunto de estrategias que permiten encontrar la solución de un problema. La aplicación de esta técnica está sujeta a un ordenador que controla la búsqueda de valores que resuelven de manera satisfactoria el problema, haciendo que éstos *razonen* de manera muy similar a la del ser humano.

La principal diferencia entre la algorítmica y la heurística es que mientras la primera garantiza la solución del problema, la segunda es un conjunto de estrategias que permiten encontrar la solución del problema, pero no la garantizan.

Así, la IC se refiere a los métodos que resuelven los problemas mediante el uso de algoritmos, y la ÍA es la rama de la informática que trata de resolver los problemas mediante el uso de procedimientos heurísticos.

En el campo de la investigación de operaciones, la solución a un problema de optimización, utilizando los principios de la IA, implica la creación de un sistema de procedimientos heurísticos basados en una serie de opciones de *razonamiento*. Un sistema así constituido deberá tener una representación del conocimiento en la que se definan dos estructuras de datos; una que describe el proceso de razonamiento, y otra que describe el problema. Así, al procesar los datos de la segunda estructura también deberán procesarse los de la primera, ya que esta describirá los distintos caminos para conseguir la solución del problema.

Otro módulo integrante del sistema será el procedimiento de *inferencia*, el cual permite efectuar deducciones lógicas al procesar la base del conocimiento (o estructuras de datos). Para que la base del conocimiento sea consistente e independiente del sistema de inferencia se precisa de un intérprete o conjunto de procesos del sistema, de modo que al procesar la base del conocimiento, ésta pueda ser transformada de manera consistente con su significado, para que sea posible efectuar, sobre la base del conocimiento, las inferencias y razonamientos necesarios.

Una gran ventaja de los sistemas de tratamiento de la información es que, cuando hay nueva información, ésta se añade a la base de conocimientos y todo sigue funcionando; en cambio, con los modelos algorítmicos había que diseñar todo el algoritmo. Así pues, este nuevo sistema es más apto para añadir información deducida, principalmente del aprendizaje, a través de un proceso de análisis de la información disponible.

A continuación se hace un análisis descriptivo de cuatro métodos basados en los principios de la inteligencia artificial que en los últimos años han venido cobrando auge dentro de la literatura de investigación de operaciones, sobre todo en lo que se refiere a problemas combinatorios. Dichos métodos son: Algoritmo genético, búsqueda tabú, redes neuronales y recocido simulado.

4.3 REDES NEURONALES

Considerando la complejidad computacional que representa resolver ciertos problemas de optimización que se presentan en la actualidad, muchos autores en el área de investigación de operaciones han puesto un especial interés en los métodos basados en lo que se ha dado a llamar, dentro del área de la inteligencia artificial, modelos de redes neuronales.

Este interés radica principalmente en el hecho de que los modelos de redes neuronales se caracterizan por el ahorro en los procesos computacionales que supone seguir una trayectoria lógica de nodos que mejoran con cada iteración una función de energía.

Para la investigación de operaciones esta metodología explota la notación de programación heurística y las reglas base en la toma de decisiones. Pero ciertamente no todos los problemas pueden resolverse a través de esta estructura. Un sistema experto depende de la existencia de reglas de decisión fundamentales y también de la habilidad de quien toma la decisión de cómo deben articularse dichas reglas. Las redes neuronales emulan aquellos procesos de decisión para los cuales las personas no tienen una certeza total y, cómo los humanos, toman decisiones con base a un aprendizaje previo.

4.3.1 Definición de redes neuronales.

La creación de un sistema de procesamiento por computadora que tenga la flexibilidad, creatividad y la habilidad para imitar el funcionamiento de comprensión y razonamiento del cerebro humano y su sistema nervioso[1], representa uno de los aspectos centrales de los investigadores en el campo de la inteligencia artificial (AI).

El modelo simple de una neurona consiste de dendritas, el cuerpo de la neurona, el axón y la synapses. Las dendritas llevan la señal de entrada al cuerpo de la neurona. Las señales de la dendrita puede tener voltajes positivos y negativos. El voltaje positivo contribuye a excitar el cuerpo de la neurona, el voltaje negativo contribuirá a inhibirlo.

El cuerpo de la neurona es el que realiza el procesamiento de las señales llevadas por las dendritas. El modelo del cuerpo de la neurona que se utiliza en desarrollo de los métodos basados en redes neuronales considera las sumas de las señales llevadas por sus dendritas y luego determina el rumbo que seguirán las señales que emita. Si la suma excede un valor de "umbral" el cuerpo de la neurona encenderá, generando su propia señal y enviándola a otras neuronas dentro del sistema. Si la suma no excede el valor del "umbral", el cuerpo de la neurona no

---

[1] El sistema nervioso humano consiste en redes de neuronas altamente interconectadas que actúan con base a señales emitidas unas a otras de diferente intensidad. Cada una de las neuronas desarrolla un cálculo de la intensidad de las señales recibidas y con base en el resultado determina la señal que enviará a otras neuronas. Los resultados del cálculo son transmitidas de una neurona a otra a lo largo de una ruta neural. Una simple neurona puede enviar señales a otras 10,000 en la forma de voltajes. Esas señales pueden inhibir o excitar a otras neuronas. Por ejemplo, la neurona analizada por McCulloch-Pitts acepta señales de entrada inhibitorias como valores negativos y señales excitatorias como valores positivos. Después se suman los valores positivos y negativos y si la suma excede un valor de "umbral" la neurona puede entonces enviar la señal a otras neuronas a las cuales este conectada. El proceso de exceder un umbral y enviar una señal es llamado "encendido".

encenderá.

La axón es el ducto de salida de una neurona, la cual se encarga de llevar el voltaje encendido desde el cuerpo de la neurona a varias conexiones con otras neuronas. Estas conexiones son las synapses, y puede haber miles de synapses conectadas a una sola axon. Las synapses forman una conexión entre al axon de una neurona con la dendrita de otra. La intensidad de la conexión del synapse determina el valor de la señal alcanzado por la dendrita. El valor de la intensidad de la synapse es llamada el "factor de peso".

Los factores de peso actúan para multiplicar la señal de la axon. El producto es pasado a la dendrita conectada, proporcionando la señal de entrada a la neurona conectada. El factor de peso de la synapses multiplica la señal del axon pasando el producto a la dendrita.

4.3.2 El modelo de redes neuronales.

Aunque hay muchas analogías entre los conceptos básicos de la neurofisiología y los modelos de redes neuronales no significa necesariamente que éstos sean verdaderos modelos del cerebro.

Físicamente las redes neuronales artificiales se parecen a su patrón biológico por su composición en nodos y conexiones entre ellos. Esta arquitectura posibilita la satisfacción de una *restricción local* dentro de la red neuronal. La restricción local, que se deriva de los requerimientos biológicos, implica que las neuronas sólo puedan utilizar información suministrada por ellas localmente. La información local es aquella que se transmite vía conexiones. Por lo tanto si no existe conexión entre dos neuronas, entonces la actividad de una neurona no puede ser afectada (directamente) por cualquiera de las otras.

La información local que es suministrada por una neurona es esencialmente un *peso* resultante de las señales enviadas por todas las neuronas a las que está conectada.

Por otra parte, los elementos individuales de cálculo que forman la mayoría de los modelos de sistemas neuronales artificiales no suelen denominarse neuronas artificiales; lo más frecuentes es darles el nombre de nodos, unidades o elementos de procesamiento (UP's). Todos estos términos se emplean de manera indistinta.

Otro detalle que debe tomarse en cuenta es que, no siempre es correcto pensar que los elementos de procesamiento poseen una relación biunívoca con neuronas biológicas reales. A veces es mejor tomar los elementos de procesamiento individuales, como representantes de la actividad colectiva de un grupo de neuronas.

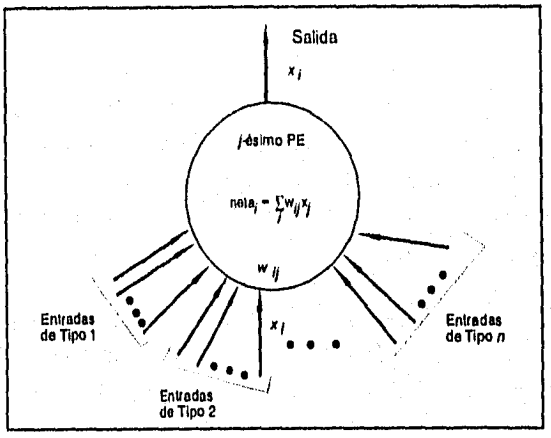

Figura 1

Esta estructura representa una única unidad de procesamiento (UP) de una red. Las conexiones de entrada se representan en forma de flechas procedemes de otras unidades de procesamiemo. Cada conexión de entrada tiene asociada una cantidad w_{ij} que se denomina peso. Hay un único valor de salida, que se puede aplicar a otras unidades.

La figura 1 muestra un modelo general de unidad de procesamiento UP. Cada UP está numerada, siendo la i-ésima la que aparece en la figura. Al igual que una neurona verdadera, la UP tiene muchas entradas pero tiene una sola salida, que se puede aplicar a muchos otras UP's de la red. La entrada que recibe la i-ésima UP procedente de la j-ésima UP se indica en la forma x_j. Cada conexión con la i-ésima UP tiene asociada una magnitud llamada *peso* o *intensidad de conexión*. El peso de la conexión procedente del j-ésimo nodo y que llega al i-ésimo nodo, se denota mediante w_{ij}. Todas las cantidades tienen sus análogos en el modelo de una neurona estándar: la salida de la UP se corresponde con la frecuencia de disparo de la neurona, y los pesos corresponden a la intensidad de las conexiones sinápticas entre neuronas. En los modelos de redes neuronales estas cantidades se representan mediante números reales.

Por otra parte, las entradas que llegan a una UP están desglosadas en varios tipos. Este desglose refleja el hecho consistente en que cada conexión de entrada puede tener uno de entre varios efectos. Una conexión de entrada puede ser excitatoria o inhibitoria, por ejemplo. En los modelos las conexiones excitatorias tienen pesos positivos y las conexiones inhibitorias tienen pesos negativos. Las conexiones excitatorias e inhibitorias suelen considerarse conjuntamente, y son las formas más comunes de entrada de las UP.

Cada UP determina un valor de entrada neto basándose en todas las conexiones de entrada. En ausencia de conexiones especiales, lo típico es calcular el valor de entrada neto sumando los valores de entrada, ponderados (multiplicados) mediante sus pesos correspondientes. Esto es, la entrada neta de la i-esima unidad se puede escribir en la forma

$$\text{neta}_i = \sum_j x_j w_{ij} \qquad (1)$$

en donde el índice j recorre todas las conexiones que posea la UP. La exitación y la inhibición se tienen en cuenta automáticamente mediante el signo de sus pesos. Dado que es frecuente que haya un número de interconexiones muy elevado en las redes, la velocidad con la que se puede llevar a cabo este cálculo suele ser determinante para el rendimiento de la simulación de cualquier red dada.

Una vez que la entrada neta ha sido calculada, se transforma en el *valor de activación*, o *activación* simplemente, para esa UP. Se puede escribir ese valor de activación en la forma

$$a_i(t) = F_i(a_i(t - 1), \text{neta}_i(t)) \tag{2}$$

para denotar que la activación es una función explícita de la entrada neta. Se observa que la activación actual puede depender del valor anterior de la activación, $a(t - 1)$[2].

En la mayoría de los casos, la activación y la entrada neta son idénticas, y los términos suelen emplearse de manera intercambiable. En algunas ocasiones, la activación y la entrada neta no son iguales, y es preciso prestar atención a la diferencia. En general sin embargo, siempre se puede utilizar la activación para denotar la entrada neta y viceversa.

Una vez que se ha calculado la activación de la UP, se puede determinar el valor de salida aplicando la *función de salida*:

$$x_i = f_i(a_i) \tag{3}$$

Dado que normalmente $a_i = \text{neta}_i$, esta función suele escribirse en la forma

$$x_i = f_i(neta_i) \tag{4}$$

Una de las razones por las cuales se estudia cuidadosamente el tema de la activación frente a la entrada neta es que el término *función de activación* se utiliza en algunas ocasiones para aludir a la función f_i, que se transforma en el valor de entrada neta, neta_i, en el valor de salida del nodo, x_i.

4.3.3 Redes neuronales en problemas de optimización.

En lo que se refiere a la solución de problemas de optimización basados en los modelos de redes neuronales, los investigadores han puesto un especial interés en los llamados *conjuntos adaptativos* para la construcción de técnicas de búsqueda de buenos resultados a problemas complejos de tipo combinatorio.

---

[2] Se considera generalmente el tiempo como una magnitud medida en pasos discretos. La notación $t - 1$ denota un paso temporal antes del instante t.

Los conjuntos adaptativos constan de dos capas de elementos de procesos que están completamente interconectadas entre ellas. Las unidades pueden no tener conexiones de retroalimentación consigo mismas. El caso general se ilustra en la siguiente figura.

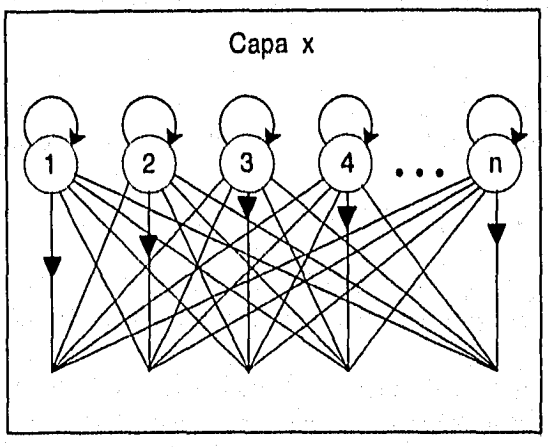

Figura 2

Cuando se tienen conjuntos como los anteriores, la información empieza a propagarse a través del sistema vía la presentación de un patrón de entrada que llega a ser la actividad de una primer capa de nodos. De ahí en adelante la activación de los nodos viaja a través conexiones directas con pesos w_{ij}, donde los subíndices indican que el peso está asociado con la conexión entre el nodo i y el nodo j, así, el nodo j recibe la sumatoria de los pesos de las actividades de todos sus nodos adyacentes vía

$$\text{neta}_j = \sum_{i}^{N} w_{ij} v_i \tag{5}$$

donde N es el número de nodos en la red, $w_{ij} = 0$ si este no tiene conexión directa del nodo i al nodo j y v_i es el estado (de inhibición o excitación) del nodo i. La función de transferencia da por tanto la nueva actividad del nodo j, v_j. El proceso continua hasta la salida final de la red que surge con los estados de aquellos nodos no alimentados en dirección lejana. Los conjuntos adaptativos tienen la propiedad de que sus pesos se adaptan en respuesta a una función de las entradas proporcionadas externamente y en la salida del sistema, generalmente los pesos son ajustados en una representación de entrada vía

$$w(t + 1) = w(t) + a\delta(t) \tag{6}$$

donde a es una tasa fraccional de aprendizaje, y $\delta(t)$ es un tiempo de error computado en el tiempo t. Sin embargo estos no aprenden adaptativamente de la repetición de los ejemplos y por tanto los pesos no cambian. Mejor dicho, los pesos son derivados inicialmente de una llamada *función de energía* (E), la cual representa el problema que debe ser resuelto y codificar los datos

del problema. Mientras los pesos permanecen fijos, el estado de los nodos cambia de acuerdo al impacto de la información local, hasta que un estado firme es alcanzado. Puesto que la función de energía suministra pesos, esta guía a los nodos de la red hacia el equilibrio. Esta función de energía, como la descrita por Hopfield[3], debe ser una función para la cual los valores inferiores representan buenas soluciones para un problema particular. La función de energía de Hopfield es

$$E = -0.5 \left[\sum_i \sum_j w_{ij} V_i V_j + \sum_i T_i V_i \right] \qquad (7)$$

donde V_i es el estado del nodo i, w_{ij} es el peso den la conexión entre i y j, y Ti es la tendencia asociada con el nodo i. Normalmente, si un nodo recibe una entrada como en (5), la cual es mayor que su tendencia, su estado es de excitación o 1; si la entrada es menor que la tendencia, su estado es se inhibición o Cero.

4.3.4 Problemas de optimización combinatoria y la red de Hopfield.

La demostración de que una red neuronal puede representar y resolver un problema difícil de optimización combinatoria ha generado mucho interés. Para ejemplificar el funcionamiento de una red basada en el modelo de Hopfield a esta clase de problemas, se hace a continuación un resumen de la propuesta de este autor para resolver el problema del agente viajero[4].

Como se estableció, de investigaciones anteriores, en estos problemas la mejor solución suele definirse mediante un criterio específico. Por ejemplo, podría haber un cierto costo asociado a cada posible solución, y la mejor solución es la que minimice el costo, satisfaciendo todos los requisitos necesarios para ser una solución aceptable. De hecho, en muchos casos los problemas de optimización se describen en términos de una función de costo.

Uno de estos problemas es el del agente viajero (TSP). En su forma más sencilla, un viajero tiene que hacer una ruta que pasa por cierto número de ciudades, visitando cada una de ellas una sola vez, minimizando la distancia total recorrida. El problema es hallar la secuencia correcta en que hay que visitar las ciudades. Las limitaciones son que hay que visitar todas las ciudades, cada una, sólo una vez y que el viajero vuelve al punto inicial al final del viaje. L función de costo que hay que minimizar es la distancia total recorrida durante el viaje.

Se observa que la distancia mínima no es una *ligadura* en este problema. Las ligaduras *deben* ser cumplidas, mientras que la distancia mínima es tan solo el objetivo perseguido. se puede

---

[3] La descripción de la memoria de Hopfield y de cómo se estructura la función de energía, se explica a detalle en: Freeman J.A. y Skapura. *Redes neuronales: Algoritmos, aplicaciones y técnicas de programación*. Addison-Wesley, U.S.A. 1991.

[4] Hopfield J.J. y Tank D.W. *Neural computation of decisions in optimization problems*. En Biology Cybernet, No. 56. Jul. 1985. pp. 141-152.

incluir la distancia mínima como ligadura si se desea distinguir entre dos tipos de ligaduras: fuertes y débiles. Las ligaduras débiles, tales como la distancia mínima en el TSP, son condiciones que se desean pero quizá no se alcancen. Las ligaduras fuertes son condiciones que se deben satisfacer, o bien la solución obtenida no será válida.

El TSP es intensivo en términos de cálculo si se debe analizar una búsqueda exhaustiva que compare todas las rutas posibles con objeto de encontrar la mejor. Para una ruta de *n* ciudades, hay *n!* rutas posibles. Debido a las degeneraciones, el número de soluciones "distintas" es menor que *n!*. El término *distintas*, en este caso, se refiere a viajes cuyas distancias totales sean distintas, para un viaje dado, no importa cual de las *n* ciudades sea el punto inicial, en términos de la distancia total recorrida. Esta degeneración reduce el número de rutas distintas en un factor *n*. De manera similar, no importa en cuál de las dos direcciones se desplace el viajero. Este hecho reduce aún más el número de rutas en un factor 2. De esta forma, para una ruta de *n* ciudades, hay que considerar *n!/2n* recorridos distintos.[5]

La cantidad de tiempo de cálculo que requiere un computador digital para resolver este problema crece exponencialmente con el número de ciudades. El problema pertenece a la clase de problemas que se conocen con el nombre de problemas NP-completos. Como consecuencia de la carga de cálculo, en los problemas de optimización suele suceder que una solución *buena* que se encuentre rápidamente sea preferible a la *mejor* solución, cuando esta última se encuentra demasiado tarde para ser útil.

Los modelos de redes neuronales qua han aparecido para resolver este problema se basan en la memoria de Hopfield[6]. La característica de interés es la rápida minimización de una función de energía, E. Aunque se garantiza que la red converja a un mínimo de la función de energía, no se garantiza que vaya a converger al mínimo *más bajo* de energía: la solución será, con toda probabilidad, una buena solución, pero no necesariamente la mejor. Dado que las Unidades de Procesamiento (UP) funcionan en paralelo (en circuito real), el tiempo de cálculo se ve minimizado. De hecho, añadir una ciudad más no tendría efectos significativos sobre el tiempo requerido para determinar una solución.

Para utilizar la memoria de Hopfield para esta aplicación se debe hallar una forma de hacer corresponder el problema con la arquitectura de la red.

---

[5] Para una ruta de cinco ciudades, habría 120/10 = 12 recorridos distintos; por supuesto no merece la pena resolver por computadora este problema. Sin embargo, una ruta de 10 ciudades tiene 3,628,800/20 = 189,440 rutas distintas; una ruta de 30 ciudades tiene más de 40 X 10^{30} posibilidades. La adición de una sola ciudad a una ruta da lugar a un incremento del número de recorridos distintos dado el factor

$$\frac{(n+1)!/2(n+1)}{n!/2n}$$

por tanto una ruta de 31 ciudades requiere examinar 21 veces más recorridos que una ruta de 30 ciudades.

[6] Burke I. Laura y James P. I. *Neural Networks and operation research an overview.* En Computer Operations Research. Vol. 19, No. 3/4. 1992. pp. 179 - 189.

Lo primero que hay que desarrollar es una representación de las soluciones del problema que encaje con una arquitectura que posea una sola lista de UP. Esta se desarrolla reservando un conjunto de n UP's (para un recorrido de n ciudades) con objeto de representar las n posiciones posibles de una ciudad dada dentro de la secuencia de recorrido. Para un recorrido de cinco ciudades esta información podría ser aportada por cinco UP. Por ejemplo, los cinco elementos 0 0 0 1 0 indicarían que la ciudad en cuestión era la cuarta en ser visitada durante el recorrido. Habría cinco grupos, cada uno de los cuales tendría información acerca de la posición de una ciudad. En la siguiente relación se ilustra esta representación.

1 2 3 4 5	
0 1 0 0 0	**A**
1 0 0 0 0	**B**
0 0 0 1 0	**C**
0 0 0 0 1	**D**
0 0 1 0 0	**E**

en lo que sigue se empleará este formato matricial, por lo que las salidas se indicarán con v_{Xi}, en donde el subíndice **X** se refiere a la ciudad, y el subíndice i se refiere a su posición dentro del recorrido. Para tener en cuenta la ligadura consistente en que el recorrido debe empezar y acabar en la misma ciudad, es preciso definir $v_{Xn+1} = v_{X1}$ y $v_{X0} = v_{Xn}$. La necesidad de estas definiciones de hará evidente en breve.

Para definir la matriz de pesos de conexión, se empieza por establecer la función de energía. Una vez definida, se puede deducir una matriz de pesos de conexión adecuada.

Es preciso construir una función de energía que satisfaga los criterios siguientes:

1. Los mínimos de energía deben favorecer aquellos estados que incluyan a cada ciudad una sola vez en el recorrido.

2. Los mínimos de energía deben favorecer aquellos estados que contengan una sola vez cada una de las posiciones del recorrido. Por ejemplo, dos ciudades diferentes no pueden encontrarse en siguiente posición de recorrido.

3. Los mínimos de energía deben favorecer aquellos estados que incluyan a las n ciudades.

4. Los mínimos de energía deben favorecer aquellos estados que tengan las distancias totales más cortas.

La función de energía que satisface estas condiciones no es fácil de construir. La ecuación de energía propuesta es la siguiente.

$$E = \frac{A}{2}\sum_{X=1}^{n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} v_{X_i}v_{X_j} + \frac{B}{2}\sum_{i=1}^{n}\sum_{X=1}^{n}\sum_{\substack{Y=1\\ Y\neq X}}^{n} v_{X_i}v_{Y_i}$$

$$+ \frac{C}{2}\left(\sum_{X=1}^{n}\sum_{i=1}^{n} v_{X_i} - n\right)^2 + \frac{D}{2}\sum_{X=1}^{n}\sum_{\substack{Y=1\\ Y\neq X}}^{n}\sum_{i=1}^{n} d_{XY}v_{X_i}(v_{Y,i+1} + v_{Y,i-1}) \qquad (8)$$

En el último término de dicha ecuación, si $i = 1$, entonces vuelve aparecer la magnitud v_{y0}. Si $i = n$, entonces aparece la magnitud $v_{Y,n+1}$. Estos resultados determinan la necesidad de las definiciones $v_{Xn+1} = v_{X1}$ y $v_{X0} = v_{Xn}$, que se habían mencionado anteriormente.

En esta ecuación se observa en primer lugar que d_{XY} representa la distancia entre las ciudades X e Y, y que $d_{XY} = d_{YX}$. Además, si los parámetros A, B, C y D son positivos, entonces E es no negativa[7].

Tomando el primer término de la ecuación (8) y considerando la ruta de cinco ciudades propuesta con anterioridad. El producto de términos, $v_{Xi}v_{Xi}$, se refiere a una única ciudad, X. Las dos sumas interiores dan lugar a una suma de productos $v_{Xi}v_{Xi}$, para todas las combinaciones de i y de j, con tal de que i diferente j Una vez que la red se ha estabilizado en una solución, y en el límite de ganancia en el cual $v \in \{0,1\}$ todos estos términos serán cero si y sólo si existe un único $v_{Xi} = 1$ y todos los demás $v_{Xi} = 0$. Esto es, la contribución del primer término de la ecuación (1) será cero si y sólo si aparece una sola ciudad en cada fila de la matriz de UP. Esta situación corresponde a la ligadura consistente en que cada ciudad debe aparecer una sola vez en el recorrido. Cualquier otra situación da lugar a un valor positivo de este término en la ecuación de energía.

Empleando un análisis similar, se puede mostrar que el segundo término de la ecuación (8) será cero si y sólo si cada una de las columnas de la matriz de UP contiene un único valor igual a 1. Esta condición corresponde a la segunda ligadura del problema, consistente en que cada posición del recorrido tenga asociada a ella una única ciudad.

En el tercer término, $\sum_X \sum_i v_{X_i}$ es una suma sencilla de los n^2 valores de las salidas. Sólo debería haber n de estos términos que tengan un valor de 1; todos los demás deberían ser cero.

---

[7] Los parámetros A, B, C y D de la ecuación (1) no tienen nada que ver con las ciudades marcadas como A, B, C, y D.

Entonces el tercer término será cero, puesto que si $\sum_X \sum_i v_{X_i} = n$. Si hay más o menos de n términos iguales a 1, entonces la suma será mayor o menor que n, y las contribuciones del tercer término de la ecuación (8) serán mayores de cero.

El último término de la ecuación (8) calcula un valor proporcional a la distancia recorrida durante el viaje. Por tanto, un recorrido de distancia mínima da lugar a una contribución mínima de este término a la función de energía.

El resultado de este análisis es que la función de energía dada en la ecuación (8) será minimizada sólo cuando una solución satisfaga las cuatro ligaduras (tres de ellas fuertes y una débil) enumeradas anteriormente. Ahora se desea construir una matriz de pesos que corresponda a esta función de energía para que la red calcule correctamente las soluciones.

Para ello, la matriz de pesos de conexión se define solamente en términos de inhibiciones entre UP. En lugar de un índice doble, aquí se adopta un sistema de cuatro índices que corresponde al sistema de dos índices de los vectores de salida. Los elementos de la matriz de conexiones son n^2 por n^2 magnitudes, $T_{Xi Yj}$, en donde X e Y se refiere a las ciudades, e i, j se refieren a las posiciones del recorrido.

El primer término de la función de energía es cero si y sólo si nada más que un elemento de cada fila de la matriz de unidades de salida es 1. Esta situación se ve favorecida si, cuando está activada una unidad de una fila (esto es, cuando tiene una salida grande respecto a las demás), esa unidad inhibe a las otras unidades de su fila. Esta situación es, en esencia, una competencia del tipo *"el que gana se lo lleva todo"*. Considérese la magnitud

$$-A\delta_{XY}(1 - \delta_{ij})$$

en donde $\delta_{uv} = 1$ si $u = v$ y $\delta_{uv} = 0$ si u diferente de v. La primera Delta es cero salvo en la primera fila en la cual X = Y. La magnitud que está entre paréntesis es 1 a no ser que $i = j$. Este factor asegura que inhiba a todas las demás unidades de la fila pero que no se inhiba a sí misma. Por lo tanto, si esta magnitud representase un peso de conexión entre la unidades, todas las unidades de una cierta fila tendrían una conexión inhibitoria de intensidad -A con todas las demás unidades de la misma fila. A medida que la red evolucionase hacia una solución, si una unidad de una fila empezara a mostrar un valor de salida mayor que los demás, tendería a inhibir a las demás unidades de la misma fila. Esta situación se denomina ***inhibición lateral.***

El segundo término de la función de energía es cero si y sólo si una sola unidad de cada columna de la matriz de unidades de salida es 1. La magnitud

$$-B\delta_{ij}(1 - \delta_{XY})$$

da lugar a una inhibición lateral entre todas las columnas. La primera delta asegura que esta

inhibición quede confinada a cada columna en la que sea $i = j$. La segunda *delta* asegura que cada unidad no se inhiba a sí misma. La contribución del tercer término de la ecuación de energía no es, posiblemente, tan directamente comprensible como eran las dos primeras. Dado que implica una suma de todas las salidas tiene un carácter *global*, a diferencia de los dos primeros términos, que se limitaban a filas y columnas. Por lo tanto, se incluye una inhibición global, -C, tal que todas las unidades de la red son inhibidas por esta cantidad constante.

Por último, hay que recordar que el último término de la función de energía contiene información acerca de la distancia recorrida en el viaje. El deseo de minimizar este término se puede traducir a conexiones entre unidades que inhiban la selección de ciudades adyacentes en proporción a la distancia existente entre esas dos ciudades. Considérese el término

$$-Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1})$$

Para una columna dada, j (esto es, para una posición dada del recorrido), los dos términos Delta aseguran que se hagan conexiones inhibitorias sólo con las unidades de columnas adyacentes. Las unidades de columnas adyacentes representan ciudades que podrían venir o bien antes o bien después de las ciudades de la columna j. El factor $-Dd_{XY}$ asegura que las unidades que representen a ciudades más distantes recibirán la mayor señal inhibitoria.

Ahora se puede definir toda la matriz de conexión sumando todas las contribuciones de los cuatro párrafos anteriores:

$$T_{Xi,Yj} = -A\delta_{XY}(1 - \delta_{ij}) - B\delta_{ij}(1 - \delta_{XY}) \quad (9)$$
$$-Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1})$$

Las conexiones inhibitorias entre unidades se ilustran gráficamente en la siguiente figura.

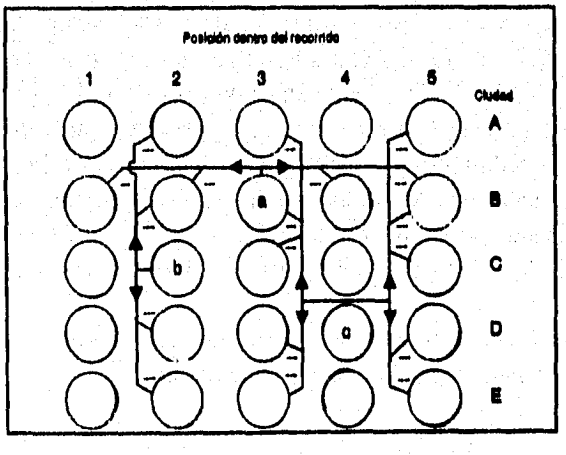

Figura 3

Este esquema ilustra la trama de conexiones inhibitorias entre UP's para el problema del TSP: la unidad *a* muestra la inhibición dentro de una sola fila, la inhibición *b* muestra la inhibición dentro de una sola columna y la unidad *c* muestra la inhibición de unidades en columnas adyacentes. No se muestra la inhibición

global.

Para hallar una solución del TSP, es preciso volver a las ecuaciones que describen la evolución temporal de la red. Para ello se utiliza la siguiente ecuación:

$$C\frac{du_i}{dt} = \sum_{j=1}^{N} T_{ij}V_j - \frac{u_i}{R_i} + I_i \tag{10}$$

Aquí se utiliza N como límite de la suma. Dado que todos los términos de T_{ij} contienen constantes arbitrarias, y que I_i se puede ajustar para que tenga cualquier valor deseado, se puede dividir por C esta ecuación y escribir

$$\frac{du_i}{dt} = \sum_{j=1}^{N} T_{ij}V_j - \frac{u_i}{\tau_i} + I_i \tag{11}$$

donde $\tau = RC$ es la constante de tiempo del sistema, y se supone que $R_i = R$ para todos los i.

La simulación digital de este sistema requiere integrar numéricamente el grupo de ecuaciones anterior. Para un valor de Δ_t suficientemente pequeño, se puede escribir

$$\frac{\Delta u_i}{\Delta t} = \sum_{j=1}^{N} T_{ij}V_j - \frac{u_i}{\tau_i} + I_i \tag{12}$$

$$\Delta u_i \left(= \sum_{j=1}^{N} T_{ij}V_j - \frac{u_i}{\tau_i} + I_i \right) \Delta_t$$

Entonces se pueden actualizar iterativamente los valores de u_i según la expresión

$$u_i(t+) = u_i(t) + \Delta u_i \tag{13}$$

en donde Δu_i está dado por la ecuación (12). Los valores finales de la salida se calculan entonces utilizando la función de salida

$$v_i = g_i(u_i)$$

Se observa que en estas ecuaciones se vuelve a emplear la notación de los subíndices utilizada en el tratamiento del sistema general: v_i en lugar de v_{xj}.

En la notación de doble subíndice se tiene

$$u_X(t + 1) = u_X(t) + \Delta u_X \tag{14}$$

Si se sustituye $T_{Xi,Yj}$ de la ecuación (9) en la ecuación (10), y se definen como entradas externas en la forma $I_{Xi} = Cn'$, en donde n' es una constante, y C es igual a la de C de la ecuación (9), el resultado sería:

$$\Delta u_{Xi} = \left[\frac{u_{Xi}}{\tau} - A\sum_{\substack{j=1 \\ j\neq i}}^{n} v_{Xj} - B\sum_{\substack{Y=1 \\ Y\neq X}}^{n} v_{Yi} - C\left(\sum_{X=1}^{n}\sum_{j=1}^{n} v_{Xj} - n\right) - D\sum_{\substack{X=1 \\ Y\neq X}}^{n} d_{XY}(V_{Y,i+1} + V_{Y,i-1}) \right] \Delta_t$$

Para completar la solución del TSP, es preciso seleccionar valores adecuados para las constantes, junto con los valores iniciales de los uXi. Hopfiel proporciona los parámetros adecuados para un problema de 10 ciudades A = B = 500, C = 200, D = 500, τ = 1, λ = 50 y n' = 15. Puede observarse que no es necesario hacer $n' = n'$. Dado que n'no entra en las ecuaciones a través de las entradas externa Tt = Cn', se puede utilizar como un parámetro ajustable más. Estos parámetros deben seleccionarse empíricamente, y los de una ruta de 10 ciudades pueden no ser válidos para rutas de otros tamaños.

Cabe sentir la tentación de hacer que todos los valores iniciales de los u_{Xi} sean iguales a una constante $u_{(X)}$, de tal forma que, en $t = 0$ sea

$$\sum_X \sum_i v_{Xi} = 10$$

porque esto es lo que se espera que valga esta suma concreta cuando la red se haya estabilizado en una solución. Al asignar valores iniciales de esta manera, sin embargo, se produce el efecto consistente en colocar el sistema en un punto de equilibrio inestable, en una situación parecida a la de la pelota que estuviese exactamente en la sima de la colina. Si no se le da al menos un empujón, la pelota quedará allí para siempre. Sin embargo, si se le llega a dar un empujón, la pelota rodaría colina abajo. Podemos darle a nuestro sistema TSP, el empujón añadiendo un término de ruido aleatoria los valores $u_{(X)}$ de tal manera que sea $u_{Xi} = u_{(X)} + \delta u_{Xi}$, donde δu_{Xi} es el término de ruido aleatorio, que puede ser diferente para cada unidad.

En la analogía de la pelota y la colina, la dirección del empujón determina la dirección en que la pelota cae rodando por la colina. De forma similar, distintas selecciones de ruido aleatorio para los valores iniciales de u_{Xi} pueden dar lugar a distintos estados estables finales. Volviendo a la descripción de los problemas de optimización, donde se dice que una buena solución en este momento puede ser mejor que la mejor solución en un instante posterior. La solución de Hopfield para el TSP puede no encontrar siempre la mejor solución (la que tenga la menor distancia posible), pero la experimentación ha mostrado que la red suele estabilizarse generalmente en rutas que se encuentran en la distancia mínima o cerca de ella. La figura 4

muestra una representación gráfica de la forma en que se iría progresando hacia una solución.

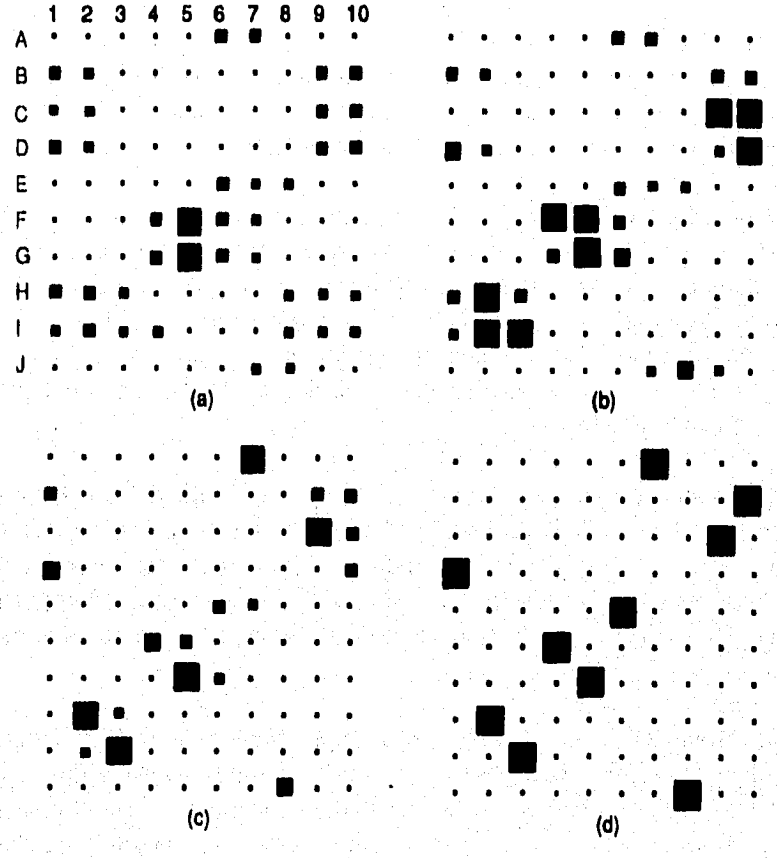

Figura 4

Esta secuencia de diagramas ilustra la convergencia de la red de Hopfield para una ruta de TSP con 10 ciudades. Los valores de las salidas Vx_i están representados como cuadrados en todas las posiciones de la matriz de unidades de salida. El tamaño del cuadrado es proporcional a la magnitud del valor de salida (*a*, *b*, *c*). En los pasos intermedios, el sistema todavía no de ha estabilizado en una ruta válida. La magnitud de los valores de salida para estos pasos intermedios se puede tomar como la estimación actual de la confianza en que una cierta ciudad venga a terminar en una posición dada dentro de la ruta. *d*) La red se ha estabilizado en una ruta válida DIHFGEAJCB.

4.3.5 Resultados presentados.

Algunos estudios comparativos con otros heurísticos han demostrado que los algoritmos basados en redes neuronales no son muy eficientes para resolver el problema del agente viajero (TSP). De hecho, el modelo original de Hopfield y Tank requiere de 10^6 neuronas interconectadas a través de 10^{12} lineas para un problema de 1000 ciudades.

Aplicando dicho modelo para el TSP, Freemman y Skapura[8] [1991] reportan que para 10 ciudades la red computa muy buenas soluciones en un período de tiempo corto, a través de un proceso de prueba y error. Sin embargo, conforme se incrementa el número de ciudades que deben recorrerse se presenta una notable descalificación de los resultados obtenidos; tan solo para el caso de 30 ciudades los resultados son mediocres y en muchos casos insatisfactorios.

Otros autores han tratado de hacer replicas de experimentos reportados, solo para obtener resultados significativamente diferentes. Una razón para esto es que no siempre usan con exactitud la misma regla de iteración para resolver un sistema de ecuaciones o hacen especificaciones diferentes a las condiciones de terminación para el mismo algoritmo.

Por ejemplo Wilson y Pawley[9] [1989] reportan su incapacidad para replicar los resultados de Hopfield y Tank en dos aspectos: Primero, la tasa de éxito del 15% en el número de pruebas que producen soluciones factibles es considerablemente menor que el reportado por Hopfield y Tank; y Segundo, la calidad de las soluciones encontradas por la red no es mejor que la selección aleatoria de recorridos. Además muestran que el modelo es impracticable para problemas muy largos a través del análisis de la relación entre el tamaño del problema y los parámetros de la función de energía.

Yue y Fu[10] [1990] por su parte argumentan que la red de Hopfield para resolver el TSP es sobrecargada en términos de memoria asociativa ocupada. La capacidad de información de la red para memoria asociativa es O(n/log n) donde *n* es el número de neuronas. El TSP tiene (N-1)!/2 soluciones factibles almacenadas en una red con solo N^2 neuronas, razón por la cual la red es sobrecargada. Esto sugiere que la red sea incapaz de explorar en su totalidad el espacio factible de soluciones.

En otro estudio, Kamgart-Parsi[11] [1990] asegura que cuando la red de Hopfield y Tank encontró una solución factible, su calidad fue generalmente muy buena. Ellos reportan que de 15 soluciones factibles en 400 ensayos 4 de ellas eran de las mejores encontradas para el TSP. Ellos también encontraron que la tasa de éxito para encontrar soluciones factibles se reduce notablemente conforme el número de ciudades se incrementa, lo que sugiere que la red de Hopfiel y Tank no es muy confiable para resolver este problema . Además, cuando ellos utilizan esta resolver el problema de agrupar N secuencias en K corridas, reportan una alta tasa de éxito y una buena perspectiva conforme el número de secuencias se incrementa.

---

8 James A. Freemman y David M. Skapura. *Neural networks algorithms. Aplications and programming techniques.* Wesley Publishing Co. Massachusetts, E.U.A. 1991.

9 Wilson G.V. y Pawley G.S. *On the stability of the TSP algorithm of Hopfield and Tank.* Biology Cybernet. No. 58. 1989. pp 63-70.

10 Yue T.W. and Fu L.C.*Ineffectiveness in solving combinatorial optimization problems using a Hopfield network: a new perspective from aliasing effect.* Biology Cybernet. No. 60. 1990. pp 115-123.

11 Kagmar-Parsi B. *On problem solving whit Hopfield neural networks.* Biology Cybernet. No.62. 1990. pp. 415-423.

En últimas fechas han aparecido propuestas que de acuerdo con los reportes presentados aportan buenas soluciones al TSP. La característica de estos es que utilizan el modelo de redes neuronales en combinación con técnicas de mejoramiento iterativo, como el recocido simulado y los algoritmos genéticos, que evita la convergencia en óptimos locales.

4.3.6 Aplicaciones en problemas de optimización.

En la actualidad han aparecido importantes propuestas en torno a la viabilidad de aplicar algoritmos basados en técnicas de redes neuronales a problemas de programación. Dentro de éstas se destacan dos propuestas básicas: la primera y la más popular consiste en formular una serie de tareas de optimización combinatoria en términos de una función de costos o de energia en la que se incluyen variables binarias, como es el caso presentado por Hopfield y Tank[12] [1985] o varibles continuas como el ejemplo del algoritmo de redes elásticas de Durbin y Willshaw[13] [1987]. En estos casos, los modelos de redes neuronales se desarrollan o interpretan como mecanismos de minimización, en los que se utiliza una red que resuelve el problema expresado como una función matemática que debe ser minimizada.

La otra propuesta básica es diseñar redes neuronales basadas en la competencia de pesos, en donde se permite a las neuronas competir para ser activadas bajo ciertas condiciones. Esta técnica suele presentar mayores dificultades en su aplicación en problemas de programación, en razón del número de neuronas que se requiere en su ejecución.

Los resultados teóricos sobre la capacidad de las redes neuronales para resolver problemas difíciles son presentados por Chee-Kit-Looi[14] [1994]. Este autor computa una red de Hopfield durante un período de tiempo largo hasta que converge. El resultado que se presenta por la aplicación de este proceso es que no puede hacerse algo mejor que sólo conseguir un mínimo local.

Por lo tanto, los métodos basados en redes neuronales no garantizan soluciones globalmente óptimas. Estos sin embargo, son convenientes para problemas que no son de gran tamaño, en donde el cálculo rápido de buenas soluciones puede ser más benéfico que un lento cómputo de soluciones óptimas.

Los problemas a los que se puede aplicar la técnica de redes neuronales para ser resueltos son, por sus características, los siguientes[15]:

---

[12] Hopfield y Tank. *Op cit. 1985.*

[13] Durbin R. and Willshaw. *An analogue approach to TSP using an elastic net method.* Nature, No 326. April, 1987. pp.389-392.

[14] Chee-Kit Looi. *Neural network methods in combinatorial optimization.* En Computers Ops. Res. Vol. 19, No.3/4, 1992. pp. 191 - 208.

[15] Burke I. Laora y James P. I. *Neural Networks and operation research an overview.* En Computer Operations Research. Vol. 19, No. 3/4. 1992. pp 179 - 189.

Problema del agente viajero.
Problema múltiple del agente viajero.
Problema de selección gráfica o corte mínimo.
Problema de ramificación.
Problema de número de particiones.
Problema de asignación cuadrática.
Problema de la mochila con múltiples restricciones cero-uno.
Problema de minimización de distancia Euclidiana.
Problema de asignación de recursos.

Para algúnos de estos problemas existen técnicas que presentan los resultados óptimos, pero a través de un largo proceso iterativo de búsqueda. Hay otros para los cuales aún no se han construido heurísticos eficientes y en los que se pueden obtener mejores soluciones utilizando técnicas de redes neuronales.

Con las avances actuales en los sistemas de información, las aplicaciones en que los modelos de redes neuronales han demostrado ser muy útiles, es en problemas de decisión relacionados con el reconocimiento de patrones[16]. Las redes que se manejan para resolver estos problemas poseen propiedades deseables que parecen ideales para ciertos problemas de asignación de recursos.

El algoritmo más comunmente usado con redes neuronales para enfrentar estos problemas de decisión es el algoritmo de retropropagación. Este consiste en un conjunto neuronal adaptativo que está configurado como un sistema de retroalimentación integrado por una capa de nodos de entrada, una capa de nodos de salida y algún número de nodos ocultos que representan funciones con característica no lineal. Durante el entrenamiento, los ejemplos de los patrones y su respuesta correcta asociada están representados por la red, la cual trata de minimizar, sobre un conjunto de entrenamiento entero, el error entre la capa de nodos de actividades de salida y la respuesta proporcionada.

Las aplicaciones de este tipo son apropiadas a problemas de determinación de carteras de inversión, determinación de riesgo asociado a crédito, clasificación de riesgo, así como a problemas de asignación de trabajos.

---

16 Bart, Dean S. y Mani, Genesh. *Using neural nets to manage investments*. En AI Expert. Vol. 9, No. 2, Feb. 1994. pp. 16 - 22.

4.4 ALGORITMO GENÉTICO.

Dentro de las áreas de la investigación de operaciones existe en la actualidad un interés bastante amplio por los algoritmos genéticos (AG), sobre todo entre la comunidad que se basa en la inteligencia artificial (IA). Este interés se debe principalmente a la insuficiencia, en algunos casos, de la programación lineal y los sistemas tradicionales basados en reglas heurísticas para resolver sistemas o modelos complejos. El concepto de algoritmo genético (AG) se refiere a la utilización de una estrategia de búsqueda heurística, basada en la genética hereditaria, que ofrece "buenas soluciones" a problemas de optimización.

Los investigadores del tema definen el algoritmo genético como *una técnica aleatoria de búsqueda que imita la evolución natural*. Esta se inicia seleccionando una población de soluciones generadas aleatoriamente para el problema que se trate. Las soluciones se mueven de una generación de soluciones a otra por crianza de nuevas soluciones usando solo evaluación objetiva y el llamado operador genético. En este sentido, dicho proceso puede ser clasificado como una técnica de iteraciones de asignación y mejoramiento. En general, el algoritmo genético trabaja con un código de las variables en lugar de las variables mismas. Típicamente una solución esta representada por una cadena de *bits*, también llamados *cromosomas*. Cada posición de un bit es llamado un *gene*, y los valores de cada uno que de los genes puede tomar se llaman *alleles*.

El algoritmo genético básico tiene tres operadores principales.

- El primer operador es *la reproducción*, donde los cromosomas (o soluciones) son copiados para la siguiente generación, con igual probabilidad, con base en la cualidad de el valor de su función objetivo.

- El segundo operador es el *cruzamiento*, donde son seleccionadas aleatoriamente pares de cromosomas que son unidos, creando nuevas parejas.

- El tercer operador, *mutación* es la alteración aleatoria del allele de un gene. Estos juegan un papel secundario en los algoritmos genéticos, puesto que en la práctica este es ejecutado con una probabilidad muy pequeña (en el orden de 1/1000). La mutación diversifica el espacio de búsqueda y protege de pérdida de material genético que pueda ser causado por la reproducción y el cruzamiento.

Para imitar el proceso de selección natural en la solución de un problema generalmente se siguen los siguientes pasos:

Paso 1. Se crea una población inicial de un cromosoma o solución.

Paso 2. Se realizar la mutación de uno o más genes, de uno o más de los cromosomas actuales, produciendo una nueva generación de cada cromosoma mutado

Paso 3. Cruzamiento de uno o más pares de cromosomas.

Paso 4. Se añade a los cromosomas mutados y a los padres de estos a la población actual.

Paso 5. Se Crea una nueva generación manteniendo lo mejor de los cromosomas de la población actual, junto con otros cromosomas seleccionados al azar de la población actual. Para asegurar la mejoría se debe de influir en la selección aleatoria de acuerdo con la adaptación evaluada.

Al aplicar un algoritmo genético en un problema de optimización se deben de tomar en cuenta los siguientes aspectos.

- ¿Cuantos cromosomas deben estar en la población? Si el número es muy bajo, pronto todos los cromosomas tendrán rasgos idénticos y el cruzamientos no hará nada; si el número es muy alto, el tiempo de cálculo será innecesariamente excesivo.

- ¿Cual es la rapidez de las mutaciones? Si esta es muy baja aparecerán los nuevos rasgos muy lentamente de la población; si es muy alta cada generación estará desligada de la anterior.

- ¿Puede, cualquier cromosoma, aparecer más de una vez en una población.

4.4.1 Algoritmo genético de optimización para el problema de asignación de trabajos.

Para ejemplificar esta técnica se toma como base el procedimiento que Atiel Ben Hadj-Alouane y Katta G. Murty[1], realizan para solucionar un problema de asignación de tareas en la industria automotriz utilizando un algoritmo genético.

El problema de asignación de tareas que se presenta se basa en los sistemas de computación distribuidos, donde un número de tareas (archivos, programas, bases de datos, etc.) deben ser asignadas a una serie de procesadores (computadoras, discos, consolas, etc.) asegurándose que todas éstas sean ejecutadas dentro de un cierto período de tiempo. El objetivo consiste en minimizar el costo de los procesadores y del procesamiento de los datos en un ancho de banda de comunicación.

En el ensamblado de un automóvil, por ejemplo, algunas labores tales como la integración de chasis, el monitoreo de la suspensión, la inyección de combustible, etc. son realizados por microcomputadoras conectadas con líneas de comunicación de alta y baja velocidad. El costo del sistema es la suma de los costos de los procesadores (o microcomputadoras), y los costos de los enlaces de las bases de datos que proporcionan comunicación entre los procesos en un ancho de

---

[1] Atiel Ben Hadj-Alouane y Katta G. Murty. *A hibrid Genetic Optimization algorithm for task allocation*. Department of industrial and operations engineering. University of Michigan. Noviembre de 1992.

banda.

Cada tarea tiene que ver con el procesamiento de una base de datos que proviene de los sensores, impulsos, procesadores de señales, filtros digitales, etc. y tiene una producción requerida en KOP (mil operaciones por segundo). Varios tipos de procesadores están disponibles. Por cada uno, se estará dando su costo y su capacidad de producción en términos de los KOP's que puedan realizar. Las tareas son inter-dependientes; para completar una tarea se puede necesitar la base de datos de algún otra tarea. Ahora, si dos tareas son asignadas a diferentes procesadores estas necesitan una capacidad de conexión (bits/segundo). Las tareas ejecutadas en el mismo procesador no tendrían comunicación como el anterior. Se especifica también que cualquier tarea puede ser procesada por cualquier procesador.

Para este modelo se define lo siguiente:

m = número de tareas que deben ser ejecutadas.

n = número máximo de procesadores que pueden ser necesarios.

a_t = requerimientos de producción (en KOP) para las tareas t, $t = 1,\ldots, m$.

s_p, b_p = costo (en dinero) y capacidad en KOP's del procesador p, $p = 1,\ldots, n$.

$c_{tt'}$ = costo (de instalación) del ancho de banda entre un par de tareas t, t' si estas son asignadas a diferentes procesadores (este término de costos incluye la más alta y la más baja conexión de comunicaciones).

Para modelar este problema, se definen las siguientes variables de decisión 0 - 1 para $t = 1,\ldots,m$ y $p = 1,\ldots,n$.

$$x_{tp} = \begin{cases} 1, \textit{si la tarea } t \textit{ es asignada al procesador } p \\ 0, \textit{en otro caso} \end{cases}$$

$$y_p = \begin{cases} 1, \textit{si el procesador } p \textit{ es usado} \\ 0, \textit{en otro caso} \end{cases}$$

En los términos de estas variables de decisión, el modelo de costo mínimo diseñado para la arquitectura de una microcomputadora es el siguiente programa entero cuadrático 0 - 1

$$\min f(x, y) = \sum_{t=1}^{m-1} \sum_{t'=t+1}^{m} c_{tt'} \left(1 - \sum_{p=1}^{n} x_{tp} x_{t'p} \right) + \sum_{p=1}^{n} s_p y_p$$

sujeto a

$$\sum_{t=1}^{m} a_t x_{tp} \leq b_p y_p, \quad para\ p = 1,...n \tag{1}$$

(T)

$$\sum_{t=1}^{m} x_{tp} \leq m y_{tp}, \quad para\ p = 1,...n \tag{2}$$

$$\sum_{p=1}^{n} x_{tp} = 1, \quad para\ p = 1,...m \tag{3}$$

$$x_{tp}, y_p \in (0, 1), \quad para\ toda\ t, p$$

La primera restricción (1) garantiza que el total de requerimientos en KOP's de todas las tareas asignadas al procesador p no excedan en su capacidad de KOP's a b_p. La restricción (2) asegura que el procesador es considerado si este es asignado al menos a una tarea. La restricción (3) garantiza que cada tarea es asignada a exactamente un procesador.

Se puede observar que la función objetivo es cuadrática y no convexa. Este problema puede ser transformado en un programa lineal entero 0 - 1 a expensas de incrementar sustancialmente el número de variables 0 - 1 dentro del modelo. Cuando el número de procesadores es sólo de dos, puede ser transformado en un problema de corte de costo mínimo y resolverse eficientemente a través de técnicas de flujo de redes. Sin embargo, para seis o más procesadores el problema se considera NP-difícil. La particularidad de este problema es que introduce costos fijos para los procesadores.

La mayoría de los algoritmos de asignación de tareas en la literatura, están basados en la técnica de ramificación y acotamiento, en heurísticos basados en una asignación inicial constructiva, o en iteraciones de asignación y mejoramiento.

En investigación de operaciones un algoritmo genético permite desarrollar el programa entero de selección múltiple (MCIP). Para ello es necesario que algunas restricciones sean relajadas a través de una función de penalización no lineal, para la cual el problema correspondiente tiene dualidad fuerte y débil, permitiendo por tanto, que el algoritmo genético sea usado para resolver el problema dual. Con base en ello se puede hacer una extensión de esta aproximación para encontrar buenas soluciones para el problema de asignación de tareas (T).

Ahora bien, aplicando relajación no lineal a la restricción (1) con una función de penalización y calculando valores de y (restricción (2)) directamente de x. El término de penalización

agregado a la función objetivo es la suma de los pesos cuadrados de la restricción de violación (1), dada como

$$p_\lambda(x, y) = \sum_{p=1}^{n} \lambda_p \left[\min(0, b_p y_p - \sum_{t=1}^{m} a_t x_{tp}) \right]^2 ,$$

donde $\lambda = (\lambda_p) \in \mathbb{R}^n$, es ≥ 0. Entonces, el problema relajado es el siguiente programa no lineal 0 - 1.

$$\min f(x, y) + p_\lambda(x, y)$$

sujeto a

$$\sum_{t=1}^{m} x_{tp} \leq m y_p, \quad para\ p = 1,...n$$

(PT_λ)

$$\sum_{p=1}^{n} x_{tp} = 1, \quad para\ t = 1,...m$$

$$x_{tp}, y_p \in (0, 1), \quad para\ toda\ t, p$$

Para el caso de programas enteros de selección múltiple, se muestra aquí que existen multiplicadores, λ, tales que el problema relajado resuelve el problema original con exactitud.

Estos resultados solo dependen de que sea finito el conjunto de soluciones factibles de (PT). Dada la estructura de selección múltiple de (PT), esta condición es satisfecha

4.4.1.1 El algoritmo Genético para (PT_λ) λ fijo.

El uso de un código directo en el cual una solución está representada por una cadena de longitud igual al número de tareas. El *allele* es un gene dado para una tarea dada y toma el valor del índice del procesador al cual está asignado, puesto que cada gene de la cadena puede tomar cualquier entero en $\{1,..., n\}$. Se observa que las variables y_p, $p = 1,...,n$, no están incluidas en la representación. La razón es que estas pueden ser fácilmente determinadas dentro del proceso de asignación de tareas. Puesto que y_p representa que el procesador p es usado o no, éste es igual a 1 si al menos un gene tiene un valor igual a p, de otra forma este tiene un valor de cero. Por lo tanto, las variables y_p's están implícitamente incluidas en la representación.

Dada una generación actual, la siguiente generación es creada como sigue:

1 Se copian las N_c soluciones iniciales. Las soluciones de la actual generación son sorteadas en orden creciente al valor de la función objetivo, entonces las soluciones N_c son copiadas en la siguiente generación. N_c es usualmente fijado en un 10% del tamaño de la población. Esta aproximación, llamada reproducción elitista, reemplaza la tradicional reproducción probabilística. La ventaja de usar reproducción elitista es que la mejor solución es monotamente mejorada de una generación a la siguiente.

2 Apareamiento de parejas aleatorias: primero se selecciona aleatoriamente dos cadenas (padres) del total de la generación actual. Cada padre es seleccionado dando a cada individuo dentro de la generación total igual probabilidad. Después se crean dos descendencias como sigue: para cada gene se hace un sorteo aleatorio y se utiliza el resultado para determinar si se intercambian o no los alleles de los padres. Una vez que los descendientes son creados estos son evaluados, y sólo una con mejor valor objetivo es incluida en la nueva generación.

3 Crear la mutación generando aleatoriamente un pequeño número de nuevas soluciones (Nm = 1 % del tamaño de lo población) e incluirlas en la nueva generación. Una solución aleatoria consiste en una cadena de m números aleatorios, digamos $r_1\ r_2\ \ldots r_m$, donde r_i está uniformemente distribuido en el intervalo entero $[1,n]$ para cada $i = 1,\ldots m$. Esta operación es claramente diferente de la mutación gene por gene puesto que implica traer nuevos miembros a la población.

El siguiente pseudocódigo muestra como ejecutar Ng iteraciones del algoritmo genético descrito anteriormente. Se observa que la mejor solución factible encontrada es actualizada a cada nueva generación.

ALGORITMO A_1.

Inicialización. Escoger el tamaño de la población, N_g (experimentos realizados por estos autores muestran que N$[m, 3m]$ trabajos están bien en general). Colocar primero el mejor valor objetivo desde el inicio a infinito. Generar y evaluar aleatoriamente las N soluciones como se describió anteriormente.

Sea G_1 el conjunto de esas soluciones.

Sea $k = 1$.

Paso principal

Mientras ($k <$ Ng).

INICIAR

Colocar $G_{k+1} = 0$

1 Sortear las soluciones en G_k en orden creciente del valor objetivo.

Si la solución de hasta arriba en G_k es factible, actualizar la mejor solución factible hasta aquí.

2 Mientras que (| G_{k+1} |) < (N_g - N_m)

INICIAR (cruzamiento)

Seleccionar aleatoriamente dos soluciones, P_1 y P_2 de G_k.

Aparear P_1 y P_2 para producir la descendencia O_1 y O_2.

Evaluar O_1 y O_2, incluidos en G_{k+1} únicamente con el valor objetivo mas bajo.

FIN (cruzamiento).

3 Generar aleatoriamente N_m soluciones e incluirlas en G_{k+1}.

Colocar $k = k + 1$.

4.4.1.2 Un algoritmo genético para *(T)*.

La idea básica es correr el algoritmo **A1** con un parámetro de penalización oscilatorio λ. Es decir, se fija λ en un cierto valor (empíricamente determinado) y se hace la corrida del algoritmo A1 mientras se sigue con atención el valor de la penalización de la solución de hasta arriba en cualquier generación. Periodicamente para toda N_f, se revisa la posible alteración del valor de λ. Hay tres posibles soluciones.

1 Si la solución de hasta arriba tiene una penalización de cero parta todas las N_f generaciones pasadas, el valor de λ se incrementa dando a la solución una alta penalización por violación a la restricción de capacidad, por tanto, es empujado hasta abajo de la población.

2 Si la solución de hasta arriba tiene una penalización de cero para todas las N_f generaciones pasadas, el valor de λ es reducido hasta dejar la solución más infactible para competir por la posición de hasta arriba; en este caso, la propuesta de estar intercambiando λ es para diversificar la búsqueda.

3 Si el valor de penalización de la solución de hasta arriba ha oscilado entre cero y no cero al menos una vez en la pasada generación; en este caso, se continua el algoritmo con el mismo valor de λ.

Se observa que en las primeras dos situaciones, todas las soluciones de la actual generación son reevaluadas con el nuevo λ antes de continuar el algoritmo.

Se puede decir como un antecedente que el valor inicial de λ que se utiliza, está basado en una aproximación de λ_0 (referido en la ecuación (4)). Esta utiliza los valores de la solución óptima del programa lineal de relajación del problema como una aproximación de la solución óptima entera. Sin embargo (T) es un problema cuadrático no convexo para el cual al igual que en la versión continua es difícil de resolver. Por lo tanto, se utiliza una aproximación diferente de λ_0. Esta aproximación utiliza una solución heurística, en la cual sólo el costo del procesador es minimizado, puesto que éste es usualmente más grande que el costo del proceso de intercomunicación (por un factor de 100). Las tareas (tomadas en cualquier orden) son asignadas sucesivamente para el procesador menos caro sin exceder su capacidad.

Sea (x_a, y_a) la solución que representa la asignación descrita arriba. Entonces la siguiente aproximación es utilizada. $\lambda^{(1)} = \max \{\lambda_0; \in\}$ $_n$, donde $\in > 0$ y arbitrariamente pequeño, y

$$\lambda_0 = 1/\tilde{N}_1 \sum_{i \in G_1, \alpha(x_i, y_i) > 0} \left(\frac{f(x_a, y_a) - f(x_i, y_i)}{\alpha(x_i, y_i)} \right)$$

donde G_1 es el conjunto de soluciones de la generación inicial, $\tilde{N}_1$ es el número de soluciones en G_1 y con penalización diferente de cero $\alpha(x_i, y_i) \neq 0$).

Si corre lo suficientemente rápido, el procedimiento descrito arriba encontrará una solución óptima con probabilidad uno. Sin embargo este es normalmente detenido heurísticamente, colocando un límite al número de generaciones creadas. Un pseudocódigo que describe este procedimiento heurístico se presenta a continuación

ALGORITMO A_2

INICIALIZACIÓN. Escoger dos escalares $\beta_1 > \beta_2 > 1$. Los valores usados aquí son $\beta_1 = 8$ y $\beta_2 = 5.6$.

Sea $\lambda^1 = \max (\lambda_0, \epsilon)$ $_n$ como se describió arriba.

Escoger una frecuencia, N_f, para alterar λ (se utiliza $N_f =$ $_m/2$).

Escoger un valor para N_{max}, el máximo número de la generación será creado (en este caso se utiliza $N_{max} = 5000$).

Generar aleatoriamente una población de soluciones (como la descrita en el paso de inicialización del algoritmo A_1 y evaluar el valor objetivo.

Colocar $k = 1$.

PASO PRINCIPAL.

Mientras ($k < N_{max}$).

INICIAR

Crear una nueva generación como en el paso principal del algoritmo A_1.

Si la última generación consecutiva N_f tiene una penalización diferente de cero en la solución de hasta arriba, entonces $\lambda^{(k+1)} = \beta_1\lambda^{(k)}$, y se revalúa la generación actual con $\lambda^{(k+1)}$.

Si la última generación consecutiva N_f tiene una penalización de cero en la solución de hasta arriba, entonces $\lambda^{(k+1)} = \lambda^{(k)}/\beta_2$, y se revalúa la generación actual con $\lambda^{(k+1)}$.
De otra manera $\lambda^{(k+1)} = \lambda^{(k)}$.
$k = k + 1$.

FIN.

4.4.2 Resultados computacionales

Los experimentos realizados por Hadj-Alouane, en los que se aplica un algoritmo genético al problema de asignación de tareas como el que se plantea en este apartado, han tenido una importante influencia en otros investigadores, ya que sus resultados son presentados como los mejores para dicho caso en comparación a los presentados por Rao y Stone[2] [1979], Sarje y Sagar[3] [1991] y Koehler[4] [1992].

Hadj-Alouane realiza secuencias eficientes para problemas en los que se asignan 15 tareas a 5 procesadores, 20 a 6 y 40 a 12, utilizando tamaños de población de 100 y 120. El tiempo de procesamiento reportado es bastante aceptable, ya que le toma 46.45 segundos encontrar una solución óptima para el caso de 20 a 6 (tareas a procesadores) con una población de 120 valores y un procesamiento de 25 generaciones. Para el caso de 40 tareas a 12 procesadores, población de 125 y descendencia de 30 generaciones utiliza 394.84 segundos. En este caso, se observa que el tiempo de procesamiento no aumenta considerablemente conforme se incrementa el número de variables en el modelo. Esto obedece a que el operador del programa establece el tamaño de

---

2 Rao G.S. and Stone H. S. *Assignment of task in a distributed processor system whit limited memory*. IEEE Transactions on Computers. April 1979. pp 291-299.

3 Sarje A.K. y Sagar G. *Heuristic model for task allocation in distributed computer systems*. IEE Proceedings. Part E, computers and digital techniques, Vol. 138. Sep. 1991. pp 313-318.

4 Koeheler G. *On optimal allocation in a distributed processing environment*. Management Science, No. 28, Agosto, 1992. pp.839-853.

la población y el número de generaciones con los que se buscará el resultado del problema.

4.4.3 Principales resultados presentados.

Dentro de las aplicaciones actuales, se destaca la presentada por Pirkul y Polland[5] [1994] para el problema de partición gráfica uniforme. Este problema consiste en la partición de los nodos de una gráfica en dos conjuntos de igual tamaño para minimizar la suma de los costos representada por los arcos. Dichos autores elaboran un algoritmo de mejoramiento de soluciones, basado en la evolución genética de cruzamiento entre dos conjuntos de nodos. Los resultados presentados, en cuanto a valores y tiempo de procesamiento por computador, para casos en que se utilizan de 10 a 80 nodos muestran una notable mejoría en comparación con los resultados de un algoritmo basado en la técnica de recocido simulado. En comparación con los mejores resultados presentados para este problema, se observaron resultados similares en los casos mayores de 50 nodos.

Otro importante resultado es el que presentan G. Conway y A. Venkataramanam[6] [1994] a un problema de distribución de instalaciones dinámico. Estos autores aplican un algoritmo genético para minimizar una función de costo de disposición de un nuevo arreglo y el costo de flujo de material entre departamentos durante un horizonte de planeación. Los resultados que presentan se caracterizan tanto por su cercanía al mejor óptimo conocido (0.31% por abajo), como por la notable reducción en el tiempo de procesamiento de información con respecto al algoritmo que presenta el mejor óptimo. Esto último es en razón de que el modelo basado en algoritmo genético utiliza una población de 400 resultados y procrea 100 generaciones, lo que significa la creación de 40,000 miembros que deben evaluarse, para un caso en que la solución óptima global se encuentra de entre 4×10^{22} posibles soluciones.

Una de las ventajas de los algoritmos genéticos y quizá la principal, es su capacidad para incluir múltiples restricciones, además de funciones objetivo no lineales y no convexas.

Goldberg[7] [1989] señala que los algoritmos genéticos ofrecen resultados aceptables a problemas de optimización en los que la meta es ofrecer una secuencia que presente un resultado, de entre una gran cantidad de posibles resultados, que satisfaga un determinado número de restricciones. La calidad de los resultados varía de acuerdo con las características del problema en cuanto estructura, por lo que existen casos para los cuales su aplicación no aporta importantes avances.

---

5 Pirkul, Hasan y Rolland, Erik. *New heuristic solution procedures for the uniform graph partitioning problem: Extensions and evaluation.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 895 - 907.

6 Conway, Daniel G. y Venkataramanam, M. A. *Genetic search and the dynamic facility layout problem.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 955 - 960.

7 Goldberg D. E. *Genetic Algorithms in search, optimization, and machine Learning.* Addison-Wesley Publishing Co., Inc. 1989.

Conway y A. Venkataramanam[8] [1994], señalan que la convergencia al óptimo a través de este método se obtiene con base a la realización de múltiples experimentos con poblaciones diferentes, y que el aumento en el tamaño de la población de soluciones, a partir de la cual parte la descendencia genética, no es un factor de mejoramiento de la solución.

Otro aspecto que favorece el uso del algoritmos genéticos es su velocidad de procesamiento, lo que se traduce en tiempos cortos para la obtención de buenos resultados y por tanto la posibilidad de experimentar con múltiples poblaciones.

Sin embargo, la realización de múltiples corridas para poder presentar una solución que pueda considerarse como satisfactoria representa una de sus principales desventajas. En la mayoría de los casos reportados se hacen arriba de 50 experimentos para poder encontrar un óptimo considerado como bueno, además de que cada experimento es independiente de otros experimentos, por lo que resultados de cada uno pueden ser peores o mejores que los otros.

Finalmente puede decirse, que los algoritmos genéticos difieren de las técnicas de optimización tradicionales en muchos aspectos. Estos trabajan con una codificación de las variables (típicamente como cadenas) más que con las variables en sí mismas, y utilizan las reglas de transición probabilística para moverse de una población de soluciones a otras más que de una solución sencilla a otra. La más interesante e importante característica de los AG es que ellos utilizan tan solo evaluaciones de la función objetivo. Esto es, los AG no utilizan ninguna información sobre diferenciabilidad, convexidad o alguna otra característica auxiliar. Esto hace que los AG sean fáciles de utilizar y de implantar en gran variedad de problemas de optimización.

---

8 Conway, Daniel G. y Venkataramanam, M. A. *Op. cit* 1994.

4.5 RECOCIDO SIMULADO

En algunos textos sobre métodos heurísticos basados en inteligencia artificial se considera la técnica de recocido simulado como una variación del *hill climbing*, solo que en sentido descendente y con movimientos de mayores magnitudes que buscan disminuir las posibilidades de caer en puntos mínimos locales.

En optimización combinatoria, el concepto de recocido simulado (*annealing*) está basado en una fuerte analogía entre el proceso físico de recocido de sólidos y el conflicto de resolver problemas de optimización combinatoria de gran tamaño, en los cuales el número de iteraciones de un estado dado a otro es muy largo (tal como el número de permutaciones que puede hacerse en el problema del agente viajero). Para tales problemas, puede no tener sentido intentar todos los movimientos posibles, y en lugar de ello, puede usarse algún criterio que involucre el número de movimientos que han sido probados desde cada mejoramiento encontrado.

En efecto, el método de recocido simulado como un proceso computacional, se basa en el proceso físico de *recocido*, en el cual algunas sustancias físicas, tales como los metales, son derretidos subiendo la temperatura a niveles elevados y luego enfriados gradualmente hasta que algún estado sólido es alcanzado. El objetivo de este proceso es producir en una función objetivo un estado mínimo de energía final.

En investigación de operaciones el algoritmo de recocido simulado (RS) es una técnica aplicable a problemas de optimización combinatoria cuya eficacia depende de la habilidad que se tenga para identificar y definir cada uno de los factores que deben ser involucrados. Al respecto, se puede pensar que la técnica del RS tiene una analogía con respecto a la programación dinámica que se usa para resolver muchos problemas de optimización pero su aplicabilidad depende de la habilidad que se tenga para definir las etapas del problema, las ecuaciones recursivas, las variables de estado y las variables de decisión para un problema específico.

4.5.1 El proceso de recocido simulado.

El Recocido o templado denota un proceso de calentamiento de un sólido a una temperatura en la que la estructura de sus moléculas se recristalizan para producir nuevas estructuras. La temperatura de recocido o de recristalización, depende del tipo de material, del grado de maleabilidad 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 cristalización. Si se comienza con un valor máximo de la 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 alcanza 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.

Para caracterizar el equilibrio térmico se utiliza la distribución de Boltzmann. De acuerdo a esta distribución, la probabilidad que el sólido esté en un estado i con energía E_i a la temperatura T, viene dada por

$$P_T\{X = i\} = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{k_B T}\right) \tag{1}$$

donde X es una variable aleatoria que denota el estado actual del solido. $Z(T)$ es una constante de normalización llamada la *función partición*, que esta definida como

$$Z(T) = \sum_j \exp\left(\frac{-E_i}{k_B T}\right) \tag{2}$$

donde la sumatoria se extiende sobre todos los posibles estados y K_B es una constante física conocida como la constante de Boltzmann. El factor $\exp\left(\frac{-E_i}{k_B T}\right)$ se conoce como el factor de Boltzmann.

Se observa que (1) 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. Además, 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.

Bajo este esquema, el proceso de recocido físico consta de tres factores que deben determinarse. El primero es el valor inicial al que es elevada la temperatura. El segundo es el criterio que se utilizará para decidir cuando la temperatura del sistema debe ser reducido. El tercero es la cantidad a la cual la temperatura será reducida en cada tiempo.

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*.

4.5.2 El algoritmo de recocido simulado.

El proceso de recocido simulado puede usarse para describir un proceso de generación de

sucesiones de soluciones de un problema de optimización combinatoria en donde se vaya obteniendo, conforme el proceso avanza, mejores soluciones al mismo. Para este propósito, la analogía entre el sistema físico y un problema de optimización combinatoria guarda las siguientes equivalencias:

- Las soluciones de un problema de optimización combinatoria son equivalentes a los estados de un sistema físico.
- El costo de una solución es equivalente a la energía de un estado.
- Se introduce un parámetro de *control* que es equivalente al de la temperatura.

En la técnica de recocido simulado, al igual que en los algoritmos de búsqueda local, se supone la existencia de una estructura de vecindades y un mecanismo de generación en donde *(S, f) denota una instancia de un problema de optimización combinatoria y además se denota i y j como dos soluciones con costo f(i) y f(j), respectivamente. Entonces el criterio de aceptación determina si j se acepta de i a partir de aplicar la siguiente probabilidad de aceptación:*

$$P_c\{aceptar\ j\} = \begin{cases} 1 & si\ \ f(j) \le f(i) \\ \exp\frac{f(i)-f(j)}{c} & si\ \ f(j) > f(i) \end{cases} \qquad (3)$$

donde $c \in R^+$ denota el parámetro de control.

Por otra parte, *una transición es una acción combinada que transforma la solución actual en una subsecuente. Esta acción consiste de los siguientes dos pasos: (i) aplicación del mecanismo de generación, (ii) aplicación del criterio de aceptación.*

Sea c_k el valor del parámetro de control y L_k el número de transiciones generadas en la k-ésima iteración del algoritmo.

Entonces el algoritmo de recocido simulado puede describirse en Pseudo-código como se muestra a continuación.

Comienza
  INICIALIZA ($i_{inicial}, c_0, L_0$)
  $k := 0$
  $i := i_{inicial}$
  Repite
    para $l := 1$ **a** L_k **haz**
    Comienza
      GENERA (j de S_i)
      si $f(j) \leq f(i)$ **entonces** $i := j$
      sino

      si $\exp\left(\frac{f(i)-f(j)}{c_k}\right) >$ número aleatorio en [0,1]

      entonces $i := j$
    finpara
    $k := k + 1$
    CALCULA-LONGITUD (L_k)
    CALCULA-CONTROL (c_k)
  hasta criterio de paro
fin

Figura 1. Algoritmo de Recocido Simulado en pseudo-código

El algoritmo de la Figura 1, planteado por Gutierrez Andrade M. A., comienza llamando un procedimiento de inicialización donde se definen la solución inicial, la temperatura inicial y el número inicial de generaciones necesarias para alcanzar equilibrio térmico para la temperatura inicial. La parte medular del algoritmo consta de dos ciclos. El externo **Repite...hasta** y el interno **para...finpara**. El ciclo interno mantiene fijo el parámetro de control hasta que se generan L_k soluciones y se acepta o se rechaza la solución generada conforme a los criterios de aceptación considerados en (3). El ciclo externo disminuye el valor de la temperatura mediante el procedimiento **CALCULA-CONTROL** y calcula el número de soluciones a generar para alcanzar equilibrio térmico mediante el procedimiento **CALCULA-LONGITUD**. Este ciclo finaliza cuando la condición de paro se cumple.

Un rasgo característico del algoritmo de recocido simulado es que, además de aceptar mejoramientos en el costo, también acepta soluciones más malas en costo. Inicialmente, para valores grandes de c, puede aceptar grandes soluciones deterioradas; cuando c decrece, únicamente serán aceptadas pequeñas desviaciones y finalmente, cuando el valor de c se aproxima a cero, no se aceptarán desviaciones. Este hecho significa que el algoritmo de recocido

simulado, en contraste con el algoritmo de búsqueda local, puede escapar de mínimos locales además de exhibir los rasgos favorables de los algoritmos de búsqueda local; es decir, simplicidad y aplicabilidad general.

Por otra parte, la probabilidad de aceptar desviaciones esta implementada al comparar el valor de $\exp((f(i) - f(j))/c)$ con un número aleatorio generado de una distribución uniforme en el intervalo [0,1]. Además, debe ser obvio que la velocidad de convergencia del algoritmo está determinada al escoger los parámetros L_k y c_k, k = 0, 1, ... Si los valores c_k decrecen rápidamente o los valores de L_k no son grandes, se tendrá una convergencia más rápida que cuando los valores de c_k decrecen lentamente o los valores de L_k son grandes.

Ahora bien, se puede extender la analogía entre física estadística y optimización con recocido simulado a través del siguiente camino descrito por Gutierrez Andrade.

Dada una instancia (S,f) de un problema de optimización combinatoria y una estructura de vecindades adecuada, entonces, después de un número suficientemente grande de transiciones en un valor fijo de c, aplicando la probabilidad de aceptación de (3), el algoritmo de recocido simulado debe encontrar una solución $i \in S$ con una probabilidad igual a

$$P_c\{X = i\} \equiv q_i(c) = \frac{1}{N_0(c)} \exp\left(-\frac{f(i)}{c}\right) , \qquad (4)$$

donde X es una variable aleatoria que denota la actual solución obtenida por el algoritmo de recocido simulado y

$$N_0(c) = \sum_{j \in S} \exp\left(-\frac{f(i)}{c}\right) , \qquad (5)$$

denota una constante de normalización.

La distribución de probabilidad (4) se llama la *distribución estacionaria o de equilibrio* y es el equivalente a la distribución de Boltzmann en (1). La constante de normalización $N_0(c)$ es el equivalente de la función partición de (2).

Sea A y $A' \subset A$ dos conjuntos. Entonces la función característica $\chi_{(A')} : A \rightarrow \{0,1\}$ *del conjunto* A' *está definida como* $\chi_{(A')}(a) = 1$ si $a \in A'$ y $\chi_{(A')}(a) = 0$ en otro caso.

Corolario 1 Dada una instancia (S,f) de un problema de optimización combinatoria y una estructura de vecindades adecuada. Además, sea la distribución estacionaria dada por (4), entonces

$$\lim_{c \to 0} q_i(c) \equiv q_i^* = \frac{1}{|S_{opt}|} \chi_{(S_{opt})}(i) , \qquad (6)$$

donde S_{opt} denota el conjunto de soluciones globalmente óptimas.

Prueba. Usando el hecho de que para toda $a \leq 0$, $\lim_{c\downarrow 0} e_c = 1$ si $a = 0$ y 0 en otro caso, obtenemos

$$\lim_{c\downarrow 0} q_i(c) = \lim_{c\downarrow 0} \frac{\exp\left(-\frac{f(i)}{c}\right)}{\sum_{j\in S}\exp\left(-\frac{f(j)}{c}\right)}$$

$$= \lim_{c\downarrow 0} \frac{\exp\left(\frac{f_{opt}-f(i)}{c}\right)}{\sum_{j\in S}\exp\left(\frac{f_{opt}-f(j)}{c}\right)}$$

$$= \lim_{c\downarrow 0} \frac{1}{\sum_{j\in S}\exp\left(\frac{f_{opt}-f(j)}{c}\right)} \chi(S_{opt})(i)$$

$$+ \lim_{c\downarrow 0} \frac{\exp\left(\frac{f_{opt}-f(i)}{c}\right)}{\sum_{j\in S}\exp\left(\frac{f_{opt}-f(j)}{c}\right)} \chi(S - S_{opt})(i)$$

$$= \frac{1}{|S_{opt}|}\chi_{(S_{opt})}(i) + 0,$$

que completa la prueba.

El resultado de este corolario es muy importante puesto que garantiza *convergencia asintótica* del algoritmo de recocido simulado al conjunto de soluciones *globalmente óptimas* bajo la condición que la distribución estacionaria de (4) se obtiene en cada valor de c.

Usando la distribución estacionaria de (4), un conjunto de cantidades útiles pueden definirse para problemas de optimización en un camino similar al de sistemas físicos con muchas partículas o conjuntos estadísticos. Aquí se define lo siguiente:

- El costo esperado $E_c(f)$ en equilibrio esta definido como

$$E_c(f) = \langle f \rangle_c = \sum_{i\in S} f(i) P_c\{X = i\} = \sum_{i\in S} f(i) q_i(c) \qquad (7)$$

- El costo cuadrático esperado $E_c(f^2)$ en equilibrio esta definido como

$$E_c(f^2) \equiv \langle f^2 \rangle_c = \sum_{i \in S} f^2(i) P_c\{X = i\} = \sum_{i \in S} f^2(i) q_i(c) \tag{8}$$

- La varianza $Var_c(f)$ del costo de equilibrio esta definida como

$$Var_c(f) \equiv \sigma_c^2 = \sum_{i \in S} (f(i) - E_c(f))^2 \{X = i\} = \sum_{i \in S} (f(i) - \langle f \rangle_c)^2(i) q_i(c) = \langle f^2 \rangle_c - \langle f \rangle_c^2 \tag{9}$$

Las notaciones $\langle f \rangle_c$, $\langle f^2 \rangle_c$ y σ_c^2 se introducen como notaciones cortas.

- La entropía en equilibrio está definida como

$$S_c \equiv - \sum_{i \in S} q_i(c) \ln q_i(c) \tag{10}$$

La entropía es una medida natural de la cantidad de desorden o información en un "sistema".

Corolario 2 Sea la distribución estacionaria dada por (4), entonces las siguientes relaciones se cumplen:

$$\frac{\partial}{\partial_c} \langle f \rangle_c = \frac{\sigma_c^2}{c^2} \tag{12}$$

y

$$\frac{\partial}{\partial_c} S_c = \frac{\sigma_c^2}{c^3} \tag{13}$$

Prueba. Las relaciones pueden verificarse directamente usando las definiciones de (7)-(10) y sustituyendo la expresión para la distribución estacionaria dada por (4).

Corolario 3 *Sea la distribución estacionaria dada por* (4). Entonces se tiene:

$$\lim_{c \to \infty} \langle f \rangle_\infty \equiv \langle f \rangle_\infty = \frac{1}{|S|} \sum_{i \in S} f(i), \tag{13}$$

$$\lim_{c \downarrow 0} \langle f \rangle_0 = f_{opt}, \tag{14}$$

$$\lim_{c \to \infty} \sigma_c^2 \equiv \sigma_\infty^2 = \frac{1}{|S|} \sum_{i \in S} (f(i) - \langle f \rangle_\infty)^2, \tag{15}$$

$$\lim_{c \downarrow \infty} \sigma_c^2 = 0, \tag{16}$$

$$\lim_{c \to \infty} S_c = S_\infty = \ln |S|, \tag{17}$$

$$\lim_{c \downarrow 0} S_c = S_0 = \ln |S_{opt}|, \tag{19}$$

Prueba. Las relaciones pueden fácilmente verificarse usando las definiciones de costo esperado (7), varianza (9) y entropía (10) sustituyéndolos en la distribución estacionaria de (4) y aplicando argumentos similares a los de la prueba del Corolario 1.

La entropía puede interpretarse como una medida natural del orden del sistema físico; valores altos de la entropía corresponde a caos; valores bajos de la entropía a orden. Una definición similar de la entropía como la dada en (10) se conoce en teoría de la información, que pueden interpretarse como la cantidad de información contenida en un sistema. En el caso de recocido simulado y de optimización en general, la entropía puede interpretarse como una medida del grado de optimalidad.

Usando (11) y (12) se sigue que durante la ejecución del algoritmo de recocido simulado el costo esperado y la entropía decrecen monótonamente, con tal que se alcance el equilibrio para cada valor del parámetro de control, a sus valores finales f_{opt} y $\ln |S_{opt}|$, respectivamente.

La dependencia de la distribución estacionaria en (4) del parámetro de control c es el propósito del siguiente corolario.

Corolario 4 Sea (S,f) denote una instancia de un problema de optimización combinatoria con $S_{opt} \neq S$ y sea $q_i(c)$ denote la distribución estacionaria asociada con el algoritmo de recocido simulado y dada por (5). Entonces se tiene

(i) $\forall i \in S_{opt}$:

$$\frac{\partial}{\partial c} q_i(c) < 0, \tag{19}$$

(ii) $\forall i \notin S_{opt}, f(i) \geq \langle f \rangle \infty$:

$$\frac{\partial}{\partial c} q_i(c) > 0, \tag{20}$$

(iii) $\forall i \notin S_{opt}, f(i) < \langle f \rangle \infty, \exists \hat{c}_i > 0$:

$$\frac{\partial}{\partial c} q_i(c) > 0 \; si \; c < c_t$$
$$= 0 \; si \; c = c_t \qquad (21)$$
$$< 0 \; si \; c > c_t$$

Prueba. De (5) se deriva la siguiente expresión:

$$\frac{\partial}{\partial c} N_0(c) = \sum_{j \in S} \frac{f(i)}{c^2} \exp\left(\frac{-f(j)}{c}\right)$$

Por lo tanto se obtiene:

$$\frac{\partial}{\partial c} q_i(c) = \frac{\partial}{\partial c} \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0(c)}$$

$$= \left\{ \frac{f(i)}{c^2} \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0(c)} - \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0^2(c)} \frac{\partial}{\partial_i} N_0(c) \right\} \qquad (22)$$

$$= \frac{q_i(c)}{c^2} f(i) - \frac{q_i(c)}{c^2} \frac{\sum_{j \in S} f(i) \exp\left(\frac{-f(i)}{c}\right)}{N_0(c)}$$

$$= \frac{q_i(c)}{c^2} \left(f(i) - \langle f \rangle_c \right)$$

Así, el signo de $\frac{\partial}{\partial c} q_i(c)$ está determinado por el signo de $f(i) - \langle f \rangle_c$ puesto que $\frac{q_i(c)}{c^2} > 0$, para toda $i \in S$ y $c > 0$.

De (11), (13) y (14) se tiene que $\langle f \rangle_c$ se incrementa monótonamente de f_{opt} a $\langle f \rangle_\infty$ cuando c se incrementa, con tal que $S_{opt} \neq S$. De lo anterior, la prueba del teorema se sigue directamente.

Si $i \in S_{opt}$ y $S_{opt} \neq S$, entonces $f(i) < \langle f \rangle_c$. Así $\frac{\partial}{\partial c} q_i(c) < 0$, que completa la prueba de la parte (i).

Si $i \notin S_{opt}$, entonces el signo de $\frac{\partial}{\partial c} q_i(c)$ depende del valor de $\langle f \rangle_c$. Por lo tanto, si se usa (22),

se tiene que $\forall i \in S - S_{opt}$: $\frac{\partial}{\partial c} q_i(c) > 0$ si $f(i) \geq \langle f \rangle \infty$, mientras que $\forall i \in S - S_{opt}$, donde $f(i) < \langle f \rangle \infty$, existe $c_i > 0$ en donde $f(i) - \langle f \rangle$ cambia de signo. Consecuentemente se cumple que

$$\begin{aligned} \frac{\partial}{\partial c} q_i(c) &> 0 \; si \; c < c_i \\ &= 0 \; si \; c = c_i \\ &< 0 \; si \; c > c_i \end{aligned}$$

Esto completa la prueba de las partes (ii) y (iii).

Del corolario 4 se sigue que la probabilidad de encontrar una solución óptima se incrementa monótonamente cuando c decrece. Además, para cada solución que no sea óptima, existe un valor positivo c_i del parámetro de control, tal que para $c < c_i$ la probabilidad de encontrar esta solución decrece monótonamente cuando c decrece.

Experimentos que se han hecho con recocido simulado en una variedad de problemas sugieren que la mejor forma para seleccionar un plan de recocido es a través de la variedad de intentos y observar el efecto tanto en la calidad de la solución que es encontrada como la tasa a la cual el proceso converge.

4.5.3 Aplicación de Recocido Simulado en Problemas Combinatorios.

Para analizar las bondades técnicas del método de recocido simulado, Gutiérrez Andrade[1] [1991] realizó una investigación comparativa, en la que confronta el resultado obtenido a través de RS con los mejores resultados presentados hasta el momento dentro de la literatura para algunos de los problemas combinatorios más representativos. Uno de estos es el que se refiere al problema de localización de plantas, en donde se establece los lineamientos a seguir para implantarlo en computadora.

Para ello este autor compara un método exacto, un heurístico y un método de búsqueda local. El método exacto y el heurístico seleccionados se reportan en la literatura como los mas eficientes.

El análisis comparativo de los cuatro algoritmos se hizo en cuanto a los valores que se obtienen de la función objetivo. Para cada algoritmo se reporta la experiencia computacional.

---

[1] Gutiérrez Andrade, Miguel Angel. *La técnica de recocido simulado y sus aplicaciones*. Tesis doctoral en investigación de operaciones. México, UNAM. División de Estudios de Posgrado de la Facultad de Ingeniería. 1991.

Descripción:

El problema de localización de plantas con base en el cual se hizo la comparación presenta la siguiente estructura: hay n sitios en una región que requiere un producto. la demanda para el producto en el sitio i es d_i unidades para $i = 1,\ldots,n$. La demanda tiene que satisfacerse manufacturando el producto que satisfaga la demanda donde m se especifica. El costo por construir una planta en el sitio i es f_i pesos. Si una planta se construye en el sitio i, k_i, unidades es la capacidad de producción.

Si existe una ruta de transporte entre el sitio i, al j, k_{ij} es la capacidad sobre el horizonte de planeación y c_{ij} es el costo de transporte de una unidad entre el sitio i al j. El problema es determinar un subconjunto óptimo de sitios de localización de las plantas y un plan de transporte que minimice el costo total de construcción de plantas y el transporte de productos. El problema de determinar un subconjunto de sitios para localizar plantas y los costos fijos de transporte entre los sitios es un problema de optimización combinatoria. Una vez que se conoce una solución de este problema se dice que se ha decidido donde colocar las plantas.

Gutierrez Andrade en su investigación formula este problema como un problema entero usando variables de decisión con las siguientes interpretaciones:

y_i = 1 si la planta se localiza en el sitio i.
y_i = 0 en otro caso.
y_{ij} = si existe una ruta de transporte del sitio i al j.
y_{ij} = 0 en otro caso.
x_{ij} = cantidad (en unidades) del producto transportado del sitio i al j.

Entonces el problema de localización de plantas puede formularse en términos matemáticos como:

$$\min \sum_i \sum_j c_{ij}x_{ij} + \sum_i f_i y_i + \sum_i \sum_j f_{ij} y_{ij}$$

sujeto a

$$\sum_j x_{ij} - k_i y_i \leq 0 \quad \forall i$$

$$x_{ij} - k_{ij} y_{ij} \leq 0 \quad \forall i,j$$

$$\sum_i x_{ij} \geq d_i \quad \forall j$$

$$\sum_i y_i \leq m$$

$$x_{ij} \geq 0 \ \forall i,j; \quad y_i y_{ij} = 0 \text{ ó } 1 \ \forall i,j$$

El problema de localización de plantas también está clasificado dentro de la familia NP-completa y por lo tanto los algoritmos conocidos para resolver el problema de manera exacta, son de enumeración implícita o de ramificación y acotamiento.

El algoritmo de recocido simulado puede usarse para atacar este problema y obtener soluciones que no necesariamente son óptimas pero se puede mostrar un comportamiento muy favorable del método con respecto a instancias en que se conoce la solución.

En este caso el algoritmo de recocido simulado es aplicado de la siguiente manera:

- El espacio de soluciones S, para el problema combinatorio, esta representado por todos los vectores binarios $(y_1, y_2, \ldots, y_n)$ tal que $y_i = 0$ o 1 y $\Sigma_{i=1} y_i \leq m$. Donde $y_i = 0$ si no se coloca planta en el sitio i y $y_i = 1$ si se coloca planta en el sitio i.

- La función a minimizar, se escoge como:

$$f = \sum_i \sum_j c_{ij} x_{ij} + \sum_i f_i y_i$$

  donde x_{ij} es la cantidad (en unidades) del producto transportado del sitio i al sitio j, f es el costo de la localización más transporte.

- Las nuevas soluciones se ganarán a partir de dos mecanismos: Uno que selecciona aleatoriamente un sitio i, si existe planta en el sitio i se quita y si no existe, entonces se coloca una planta. El otro mecanismo genera aleatoriamente un sitio i, si existe una planta en i se cambia a otro sitio j donde no exista, tomando en forma aleatoria y si no existe planta en el sitio i, se busca aleatoriamente un sitio j donde exista una planta y se intercambia.

  La selección de uno u otro mecanismo se hace de manera aleatoria. Con probabilidad p se hace intercambio de plantas y con probabilidad 1 - p se abre una planta o se cierra. El valor de p puede variarse dependiendo del parámetro de control.

- La actualización de la función de costos entre la solución actual y la nueva solución puede hacerse fácilmente en la parte de costos fijos únicamente al quitar, agregar o intercambiar los costos de plantas que se den de alta y de baja. La parte del costo de transporte debe calcularse para cada nueva configuración usando la solución actual y modificando los suministros variados por medio de penalizaciones.

4.5.4 Resultado computacional.

La prueba del algoritmo se llevó a cabo generando de manera aleatoria instancias del problema de localización de plantas. Para valores preestablecidos de m número de plantas y n número de consumidores, se generaron 100 instancias aleatorias del problema. Para cada planta se tomó como costo fijo un número aleatorio con distribución uniforme entre (100-500); para cada consumidor se tomó como costo de transporte un número aleatorio con distribución uniforme entre (40-200) y finalmente para la demanda de consumidores, se tomó un número aleatorio con distribución uniforme entre (10-100).

4.5.4.1 Planteamiento del algoritmo de recocido simulado para el problema de localización de plantas.

Propósito: Obtener los sitios donde colocar las plantas y la asignación de plantas a consumidores.

Descripción:

Paso 1: Tome c un valor grande apropiado.

Paso 2: Asignar valores 0 o 1 a y_i, con $i \in \{1,2,\ldots,n\}$ de manera arbitraria.
Obtener el valor correspondiente de la función objetivo f.

Paso 3: Asignar el valor verdadero a la variable lógica CHANGE.

PASO 4: Si la variable CHANGE tiene valor falso, entonces parar.

Paso 5: Hacer CHANGE igual a falso.

Paso 6: Repetir los pasos 1 a 11 r veces.

1.- Seleccionar aleatoriamente un número $u \in \{1,\ldots,n\}$.
2.- Generar un número aleatorio x, uniformemente distribuido en (0,1).
3.- Si $x \geq p$ se abre (o cierra) la planta u, en caso contrario intercámbiarla por una cerrada (abierta).
4.- Evaluar el valor Δf.
5.- Si $\Delta f < 0$ entonces ir a 9.
6.- Sea $P(\Delta f) = \exp(-\Delta f/c)$.
7.- Generar un número aleatorio x, uniformemente distribuido en (0,1).
8.- Si $x \geq P(\Delta f)$ entonces ir a 11.
9.- Aceptar el cambio hecho y reactualice el valor de f.
10.- Si $\Delta f \neq = 0$ entonces CHANGE ← verdadero.
11.- Fin de paso 6.

Paso 7 Reducir c e incrementar p y r.

Paso 8 Ir al paso 4.

4.5.4.2. Análisis comparativo de métodos.

Se generaron para cada valor fijo de m y n 100 instancias en forma aleatoria del problema de

localización de plantas. Para cada una de estas instancias generadas, se corrieron los cuatro algoritmos y se calculó el tiempo promedio de solución.

El tiempo de solución del algoritmo de recocido simulado es muy variable, ya que depende en gran medida del valor inicial del parámetro de control c, del número r de veces que se ejecuta el paso 6 (del algoritmo) y de la forma en que se reduzca el parámetro c y se incrementen los parámetros p y r. En su análisis Gutiérrez Andrade, toma un valor inicial de c de tal manera que la tasa de aceptación de las soluciones propuestas fuera mayor o igual a 0.95 y los decrementos fueran de 0.1 (paso 7 del algoritmo). El número r inicial que se tomó en cada caso fue $r = n$ y los incrementos fueron de 0.05 (paso 7 del algoritmo).

Para el cálculo la eficiencia de cada uno de los algoritmos con respecto al valor óptimo de la función objetivo se tomó como referencia la ecuación (1).

Los cálculos que aparecen en la siguiente tabla son con base en los promedios obtenidos en las 100 instancias resultas para cada valor del número m de plantas y del número n de consumidores.

En dicha tabla, puede observarse que las eficiencias del algoritmo de recocido simulado son mejores que las de los otros dos algoritmos heurísticos y que el comportamiento de esta eficiencia se mantiene conforme crecen los valores del número de plantas y del número de consumidores.

Las pruebas computacionales con el problema de localización de plantas revelan un excelente comportamiento del algoritmo. Las soluciones subóptimas difieren en 1.2% de la solución óptima. Además si se ejecuta el procedimiento varias veces con diferentes valores iniciales, se obtiene en todos los casos la solución óptima

m	n	ALGORITMO (eficiencia)			
		EXACTO	HEURISTICO	REC. SIM.	BUS. LOC.
5	5	1.00	0.9504	0.9899	0.9051
	10	1.00	0.9490	0.9966	0.8965
	15	1.00	0.9377	0.9989	0.8679
	20	1.00	0.9371	0.9977	0.8587
10	10	1.00	0.8440	0.9962	0.9058
	15	1.00	0.8563	0.9929	0.8526
	20	1.00	0.8920	0.9957	0.8671
	25	1.00	0.9140	0.9912	0.8623
15	15	1.00	0.8033	0.9924	0.8973
	25	1.00	0.8655	0.9985	0.8631
	30	1.00	0.8690	0.9963	0.8651
	35	1.00	0.8662	0.9796	0.8249
20	20	1.00	0.7971	0.9919	0.8450
	25	1.00	0.8042	0.9847	0.8357
	30	1.00	0.7982	0.9863	0.8126
	35	1.00	0.8243	0.9895	0.7991
25	25	1.00	0.7405	0.9911	0.8552
	30	1.00	0.7959	0.9884	0.8696
	35	1.00	0.8079	0.9943	0.8229
	40	1.00	0.7659	0.9886	0.8027
30	30	1.00	0.7586	0.9891	0.8222
	40	1.00	0.7786	0.9920	0.8149
	50	1.00	0.8056	0.9881	0.7604
	60	1.00	0.8048	0.9866	0.7669
35	35	1.00	0.7753	0.9904	0.8049
	40	1.00	0.7516	0.9906	0.7982
	50	1.00	0.7755	0.9898	0.7890
	60	1.00	0.7755	0.9923	0.7883
40	40	1.00	0.7234	0.9857	0.7814
	45	1.00	0.7945	0.9943	0.7886
	50	1.00	0.7676	0.9917	0.7699
	60	1.00	0.7578	0.9862	0.7621

Comparación de los algoritmos que resuelven el problema de localización de plantas con respecto a su eficiencia.

4.5.4.3. Principales resultados presentados.

Los reportes sobre problemas de optimización combinatoria resueltos con métodos basados en la técnica de recocido simulado no se encuentran con frecuencia en la literatura de investigación de operaciones. Como referencia básica puede consultarse a E. Aarts y J. Koist[2] [1989], quienes describen en detalle el proceso heurístico de recocido y algunas aplicaciones basadas en el mecanismo de Boltzmann.

También puede consultarse a S. Kirkpatrick y C. Gelatt[3] [1993], para una mejor referencia sobre aplicaciones a problemas de optimización tales como el del agente viajero, secuenciación de vehículos y localización de plantas.

En la mayor parte de las referencias que existen sobre las aplicaciones del recocido simulado se le presenta en combinación con otros métodos heurísticos, formando estrategias híbridas para atacar problemas combinatorios que caen en el grupo de los llamados NP-completos.

Hassan[4] [1993] elabora una propuesta que incorpora el proceso de recocido simulado a la técnica de búsqueda tabú para resolver un problema de asignación de rutas de vehículos, encontrando resultados que compiten satisfactoriamente con los mejores resultados publicados, tanto en optimalidad como en tiempo de cómputo.

Otra propuesta es la de Kenneth[5] [1994], que utiliza el proceso de aleatoriedad que emplea el recocido simulado, para establecer el criterio de selección de padres bajo el cual habrán de surgir las nuevas descendencias mejoradas en los algoritmos genéticos.

Por su parte P.H. Winston[6] [1992] explica como puede utilizarse un proceso de recocido simulado en un procedimiento de retropropagación que determine cambios en el peso de las redes neuronales en diversos problemas de optimización.

---

2 Aarts E. y Koist J. *Simulated annealing and Boltzmann's machines*. Wiley, New York ,1989)

3 Kirkpatrick S. y Gelatt C. *Optimization by simulated annealing*. Science, No. 220, 1993. pp. 671 - 680

4 Hassan Osman, Ibrahim. *Memostrategy simulated annealing and tabu search algorithmsfor the vehicle routing problem*. En Annals of operations research, No. 41, 1993. pp. 421 - 451.

5 Kenneth Price U. *Genetic annealing: Functioning and use of the hybrid random-search technique*. En Dr. Dobb's Journal, Octubre 1994.

6 Winston P.H. *Inteligencia artificial*. Addison-Wesley Iberoamericana. U.S.A. 1992.

3.6 BÚSQUEDA TABÚ.

La búsqueda tabú es un procedimiento heurístico de *alto nivel*, desarrollado por Fred Glover, que se ha estado utilizando con gran éxito para resolver problemas de optimización cuya característica principal es la de *escapar* de la optimalidad local.

El principal problema que presentan algunos algoritmos heurísticos es su inhabilidad para enfrentarse con puntos de optimalidad local, es decir, son incapaces de continuar la búsqueda hacia el óptimo global después de que un óptimo local se ha alcanzado.

Al respecto, el método de búsqueda tabú puede ser considerado como una técnica basada en conceptos seleccionados de inteligencia artificial, que sigue un procedimiento heurístico general para guiar la búsqueda y obtener buenas soluciones en problemas de optimización combinatoria considerados NP difíciles. Además sus reglas son lo suficientemente amplias que son utilizadas con frecuencia para dirigir la operación de otros procedimientos heurísticos.

La Búsqueda Tabú puede esbozarse en términos generales, como sigue:

Se desea moverse paso a paso desde una solución factible inicial de un problema de optimización combinatoria hacia una solución que proporcione un valor mínimo de la función objetivo C, para esto se puede representar a cada solución por medio de un punto s (en algún espacio) y se define una vecindad N(s) de cada punto s.

El paso básico del procedimiento consiste en empezar desde un punto factible s y generar un conjunto de soluciones en N(s); entonces se escoge al mejor vecino generado s' y se posiciona en este nuevo punto ya sea que C(s^*) tenga o no mejor valor que C(s).

Hasta este punto se está acercando a las técnicas de mejoramiento local a excepción del hecho de que se puede mover a una solución peor s^* desde s.

La característica importante de la BT es precisamente la construcción de una *lista tabú T de movimientos*: aquellos movimientos que no son permitidos (movimientos tabú) en la presente iteración. La razón de esta lista es la de excluir los movimientos que pueden hacer regresar a algún punto de la iteración anterior. Ahora bien, un movimiento permanece como tabú solo durante un cierto número de iteraciones, de forma que se tiene que T es una lista cíclica donde para cada movimiento $s' \rightarrow s^*$ el movimiento opuesto $s^* \rightarrow s$ se adiciona al final de T donde el movimiento más viejo en T se elimina.

Las condiciones tabú tienen la meta de prevenir ciclos e inducir la exploración de nuevas regiones. La necesidad del significado de eliminar ciclos se debe a que, al moverse desde un óptimo local, una elección irrestricta del movimiento permite igualmente volver al mismo óptimo local.

Sin embargo, la eliminación de ciclos no es la última meta en el proceso de búsqueda. En algunos casos una buena trayectoria de búsqueda resultará al volver a visitar una solución encontrada antes. El objetivo de manera amplia es el de continuar estimulando el descubrimiento de nuevas soluciones de alta calidad.

Ahora bien, las restricciones tabú no son inviolables bajo toda circunstancia. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra encontrada, su clasificación tabú puede eliminarse. La condición que permite dicha eliminación se llama *criterio de aspiración.*

Es así como las restricciones tabú y el criterio de aspiración de la BT, juegan un papel dual en la restricción y guía del proceso de búsqueda. Las restricciones tabú, permiten que un movimiento se observe como admisible si se satisface.

La búsqueda tabú en una forma simple descubre dos de sus elementos claves: La de restringir la búsqueda mediante la clasificación de ciertos movimientos como prohibidos (es decir tabú) y de liberar la búsqueda mediante una función de memoria de término corto que proporciona una "estrategia de olvido".

Dentro del proceso de búsqueda se hace énfasis a tres aspectos:

(1) El uso de T proporciona la *búsqueda restringida* de elementos de la aproximación y por lo tanto las soluciones generadas dependen críticamente de la composición de T y de la manera como se actualiza.

(2) El método no hace referencia a la condición de optimalidad local, excepto implícitamente cuando un óptimo local mejora sobre la mejor solución encontrada previamente.

(3) Un *mejor* movimiento (mas que movimiento de mejora) se elige en cada paso.

Para problemas grandes, donde las vecindades pueden tener muchos elementos o para problemas donde esos elementos son muy costosos de examinar, es importante aislar a un subconjunto de la vecindad, y examinar este conjunto en vez de la vecindad completa. Esto puede realizarse a pasos permitiendo al subconjunto de candidatos expandirse si los niveles de aspiración no se encuentran.

4.6.1 Características del algoritmo de búsqueda tabú

El método de BT para problemas combinatorios es aplicable en la fase de mejoramiento del proceso en orden a continuar la búsqueda cuando un óptimo local se encuentra.

Dada una solución inicial el método realiza movimientos hasta que un tiempo de computadora (*tiempo_limite)* específico transcurre. El movimiento que se realiza en cierta iteración se

encuentra mediante revisar el valor del movimiento de todos los movimientos *candidatos* para solución actual.

Por ejemplo, dada una función objetivo, *Minimizar* F(x): $x \in X \subset R_n$, el valor del movimiento es la diferencia entre el valor de la función objetivo después del movimiento, F(x*), y el valor de la función objetivo antes del movimiento, F(x), es decir:

$$valor\_movimiento = F(x^*)\text{-}F(x).$$

Los valores del movimiento por lo general proporcionan una base fundamental para evaluar la calidad de un movimiento, aunque otros criterios también son importantes.

Dado que se está minimizando, el mejor movimiento candidato es aquel que posee el menor valor algebraico. De manera más precisa, el mejor movimiento se selecciona del conjunto de *movimientos admisibles*. Un movimiento es admisible si es no tabú o sí su estatus tabú puede eliminarse por medio del criterio de aspiración. El mejor movimiento entonces se realiza y la estructura de los datos tabú se actualiza.

En la solución de F(x), la preocupación principal es el de crear un estatus tabú que prevenga que algún movimiento se invierta bajo la jurisdicción de la memoria de término corto, la cual se escoge para que F(x) tenga un número específico de movimientos futuros.

La memoria de término corto de la BT constituye una forma de exploración agresiva que busca realizar el mejor movimiento posible (Esquema 1), sujeto a requerir elecciones posibles para satisfacer ciertas restricciones (esquema 2). Esas restricciones están diseñadas para prevenir el regresarse o la repetición de cierto número de veces de ciertos movimientos mediante la ejecución de atributos seleccionados de esos movimientos prohibidos (tabú).

El esquema 1 presenta la forma de elección del mejor movimiento admisible, esto es, dado un movimiento que es candidato, se elige el mejor de todos ellos considerando que si es tabú debe de satisfacer el criterio de *aspiración*.

Sea x^* el movimiento que proporciona el mejor valor de la función objetivo hasta el momento, entonces

Para (todos los movimientos candidatos) {

Si (estatus del movimiento diferente de tabú ó F(x) + *valor_movimiento* < $F(x^*)$)

{

Si (*valor_movimiento* < *mejor_valor_movimiento*) < {
mejor_valor_movimiento ← *valor_movimiento*;

mejor_movimiento ← *movimiento actual*;
}
}
}
Ejecute *mejor_moviniento*

Esquema 1. Selección del mejor candidato admisible.

Por otra parte, en el esquema 2, se continúa con el proceso de búsqueda hasta que un criterio de paro se satisface, éste por lo general para la técnica de búsqueda tabú consiste de un número predeterminado de iteraciones.

Generar solución inicial;

Haga {
Crear lista de candidatos { Ejecute *mejor_movimiento* };
Actualizar condiciones de admisibilidad;
}
Mientras (Criterio de paro = Falso);

Terminar globalmente o transferir.

Esquema 2. Componente de memoria de corto término de la búsqueda tabú.

Otra forma de establecer un movimiento de intercambio es a través de la introducción información adicional, mediante no sólo hacer referencia de los elementos intercambiados, sino también de las posiciones ocupadas por esos elementos en el momento de su cambio. Las primeras restricciones irán de menor a mayor en cuanto a que son restrictivas, pero esto no se puede afirmar de manera uniforme. Ahora bien no existe una forma que se pueda decir que es la mejor, esto sólo se puede realizar mediante pruebas. En ocasiones es importante asegurar que las condiciones puedan manejar los procesos de solución de manera eficaz desde la vecindad actual.

La meta principal de las restricciones tabú es el permitir al método ir a puntos más allá de la optimalidad local mientras se permita la realización de movimientos de alta calidad en cada paso al mismo tiempo de que exista una negociación balanceada con respecto al esfuerzo computacional al examinar muestras muy grandes, por lo que en ocasiones es deseable incrementar el porcentaje de movimientos posibles para que reciban una clasificación tabú. Esto se puede lograr ya sea mediante el argumento en la pertenencia tabú o mediante el cambio en la restricción tabú.

Además, se requiere de una estructura de datos para guardar el seguimiento de los movimientos que son clasificados como tabú y para liberar aquellos movimientos de su condición tabú cuando su pertenencia a la memoria de término corto expire. El acompañamiento de la memoria basada en la pertenencia junto con la memoria basada en la frecuencia adicionan un componente que típicamente opera sobre un sentido. El efecto de tal memoria se puede estimular por medio de que la BT mantenga una historia selectiva H de los estados encontrados durante la búsqueda, y reemplazando la vecindad actual $N(s)$ por una vecindad modificada que depende de éste proceso histórico $N(H,s)$.

El último elemento en el procedimiento básico es el *criterio del nivel de aspiración*, cuyo propósito es el de permitir que se seleccionen "buenos" movimientos tabú si el nivel de aspiración se alcanza. El uso apropiado de tal criterio puede ser muy importante para posibilitar que un método de BT alcance sus mejores niveles de realización. Este criterio de aspiración (que puede ser estándar) es el que permite que el estatus tabú se elimine si una mejor solución que la alcanzada hasta el momento se puede obtener, por ejemplo a un movimiento tabú se le permite ejecutarse si:

$$F(x) + valor\_movimiento < F(x').$$

Ahora bien, otros criterios de aspiración pueden también proporcionar efectividad para mejorar la búsqueda.

Una base para uno de esos criterios proviene de introducir el concepto de *influencia*, que mide el grado de cambio inducido en la estructura de solución o de factibilidad. La influencia por lo general se asocia con la idea de distancia del movimiento, por ejemplo, donde un movimiento de gran distancia se concibe como de mayor influencia.

El método se inicia con una solución heurística factible inicial, la cual se guarda como la mejor encontrada.

Un paso crítico, el cual envuelve la orientación agresiva de la memoria de término corto, es la elección del mejor candidato admisible. La función *mejor_movimiento* es la que identifica a un movimiento para el cual el valor del movimiento es el más pequeño. El dominio de la función es el conjunto de todos los movimientos admisibles. El *mejor_movimiento* no tiene que ser necesariamente uno que mejore. Primero cada uno de los movimientos de la lista de candidatos se evalúa en turno.

Conforme la búsqueda progresa, la forma de evaluación empleada por la búsqueda tabú llega a ser más adaptativa, incorporando referencias concernientes para la *intensificación* y la *diversificación* regional de búsqueda. Cabe aclarar que en las estrategias basadas en consideraciones de término corto la clasificación tabú sirve para identificar elementos de la vecindad del movimiento actual, mientras que en las estrategias de término intermedio y largo pueden no tener soluciones en esta vecindad, sino que por lo general consisten de seleccionar soluciones *élites* (óptimos locales de alta calidad) encontrados en varios puntos en el proceso de

solución. Dichas soluciones élites se identifican como elementos de un conglomerado regional en las estrategias de intensificación de término intermedio, y como elementos de diferentes conglomerados en las estrategias de diversificación a de término largo.

La esencia del método depende de cómo el registro de la historia *H* se define y se utiliza, y cómo los candidatos y la función de evaluación se determinan.

Revisar el estatus tabú es el primer paso en la escena de la admisibilidad. Si el movimiento no es tabú, es inmediatamente aceptado como admisible; de otra forma, el criterio de aspiración da una oportunidad para eliminar el estatus tabú, proporcionando al movimiento una segunda oportunidad para clasificarse como admisible.

En algunos casos, si las restricciones tabú y el criterio de aspiración son suficientemente limitados, ninguno de los movimientos posibles, serán clasificados como admisible. Un movimiento "menos admisible" se salva para manipular tal posibilidad y se elige si no emergen alternativas admisibles.

La longitud de la lista tabú es un parámetro, si es demasiado pequeño el ciclado ocurrirá, pero si es demasiado grande, restringirá bastante la búsqueda para poder saltar "valles profundos" (por ejemplo el mejor mínimo local) del espacio de valores de la función objetivo. Una fase importante de la BT es la habilidad de localizar un rango robusto de longitudes de la lista tabú mediante pruebas empíricas preliminares para identificar, para una clase de problemas, los tipos de atributos y de restricciones tabú que se realizan de manera más efectiva. Acerca de esto, existe el uso de listas tabú múltiples, cada una para un tipo particular de atributo.

Otros factores que dan vitalidad a la estrategia de búsqueda del método de BT y que merecen ser resaltados son la *estrategia de oscilación*, la *memoria de término intermedio y largo*, el proceso de *intensificación y diversificación* y el *criterio de aspiración*.

4.6.1.1 Estrategia de oscilación

La estrategia de oscilación opera mediante el moverse hasta pegarle a una frontera representada por la factibilidad o un estado de construcción que normalmente puede representarse por un punto donde el método puede parar. En vez de parar, la definición de vecindad se extiende o el criterio de evaluación para seleccionar movimientos se modifica, para permitir que se pueda cruzar esa frontera. La aproximación entonces procede para una profundidad específica más allá de la frontera y entonces se regresa. En este punto la frontera de nuevo se aproxima y se cruza, esta vez desde la dirección opuesta, procediendo a un nuevo punto en turno. El proceso de aproximar y cruzar repetidamente la frontera desde diferentes direcciones crea una forma de oscilación que es la que le da el nombre de estrategia.

El control sobre esta oscilación se establece mediante el generar evaluaciones y reglas de movimientos modificadas, dependiendo de la región en la cual se está actualmente navegando

y de la dirección de la búsqueda.[1]

Ahora bien, para incorporar la estrategia de oscilación, no necesariamente se tiene que definir en términos de factibilidad, sino que puede definirse donde la búsqueda parece gravitar. La oscilación consiste en forzar la búsqueda a movimientos fuera de esta región, ofreciendo de esta manera una forma efectiva para eliminar entrampamientos suboptimales en las búsquedas estándares.

4.6.1.2 Memoria de término intermedio y largo.

El método de BT empieza con una solución factible inicial y en el proceso de ejecución, el procedimiento actualiza a los arreglos y elementos de la función memoria. Entonces el proceso se repite hasta que el criterio de terminación se encuentra.

En el método de BT, el "mejor" movimiento que se realiza en cada iteración se especifica como el movimiento admisible con el mejor valor objetivo. Ahora bien, esta estrategia no garantiza que el movimiento seleccionado permita la búsqueda en la dirección de la solución óptima, por lo que, se requiere de técnicas que permitan integrar las estrategias de intensificación y diversificación de manera efectiva, basándose sobre las funciones de memoria de término intermedio y largo de la BT. En otras palabras, es de importancia vital el "mirar" la *dependencia regional* de buenos criterios de decisión, no sólo en términos de movimientos de mejoramiento y no mejoramiento.

La memoria de término intermedio opera para registrar y comparar características de las mejores soluciones generadas durante un período particular de búsqueda. Las características que son comunes o que competen a mayoría de esas soluciones se toman como atributo regional. El método entonces procura nuevas soluciones que tengan esas características regionales.

La función de memoria de término largo, emplea principios que son inversos a los de la función de término intermedio. *La memoria de término largo* se utiliza para producir un punto inicial de búsqueda en una nueva región, mediante el penalizar las características que la memoria de término intermedio encuentra que prevalecen en la región actual de búsqueda.

4.6.1.3 Intensificación y diversificación.

La fase de intensificación proporciona una forma simple para enfocar la búsqueda alrededor de la mejor solución (o conjunto de soluciones élites) hasta el momento.

---

[1] Un ejemplo simple de esta aproximación ocurre para el problema de la mochila multidimencional donde los valores de las variables cero-uno se cambian de 0 a 1 hasta alcanzar la frontera de factibilidad. El método entonces continua dentro de la región infactible utilizando el mismo tipo de cambios, pero con un evaluador modificado. Después de un número seleccionado de pasos, la dirección se invierte mediante el cambiar las variables de 0 a 1. El criterio de evaluación se maneja para mejorar y varía de acuerdo a cuando el movimiento es de más a menos infactible y se acompañan mediante restricciones asociadas sobre cambios admisibles para los valores de las variables.

Ahora bien, la diversificación se restringe para operarse sólo en ocasiones particulares. En este caso, se seleccionan las ocasiones donde ningún movimiento de mejora admisible existe. Por lo general, se utiliza la información de la frecuencia para penalizar a los movimientos que no mejoran la búsqueda mediante el asignar una penalización grande a intercambio de pares con mayores contadores de frecuencia.

Una implantación sencilla de tal técnica se puede realizar de la siguiente manera: Primero, se cuenta el número de veces que cada movimiento, digamos m, se ha realizado en orden a calcular su frecuencia $f(m)$. Entonces una penalización $p(m)$ se asocia a cada movimiento de la siguiente manera:

$$p(m) = \begin{cases} 0, & \textit{si } m \textit{ alcanza un criterio de aspiración} \\ wf(m), & \textit{de otra forma} \end{cases}$$

donde w es una constante. Entonces el valor del movimiento puede ser:

$$F(x + m) - f(x) + p(m).$$

El peso w depende del problema, del tipo de movimiento y de la vecindad. Glover y Laguna[2] indican que en muchas aplicaciones se ha observado que este peso es aproximadamente proporcional a la raíz cuadrada del tamaño de la vecindad multiplicada por la desviación estándar del valor de (sin penalización) de cada movimiento ejecutado durante la búsqueda.

Como se puede observar, la función de penalización depende directamente del criterio de aspiración.

4.6.1.4 Criterios de aspiración.

Los criterios de aspiración se introducen en la búsqueda tabú para determinar cuando las restricciones tabú pueden sobrellevarse para remover una clasificación tabú que de otra manera se aplicaría a un movimiento. El uso apropiado de tales criterios puede ser muy importante para que un método de BT alcance sus mejores niveles de realización.

En las primeras aplicaciones de la BT se aplicó tan solo un tipo sencillo de criterio de aspiración, consistente de remover una clasificación tabú a un movimiento si éste permitía una solución mejor que la mejor encontrada hasta el momento.

De acuerdo con Glover y Laguna, las aspiraciones son de dos clases: *aspiraciones de movimiento*

---

[2] Laguna M. y Glober F. *Integrating target analysis and tabu search for improved scheduling systems*. En Expert systems with aplication, vol. 6, 1993. pp. 287 - 297.

y *aspiraciones de atributo*. Una aspiración de movimiento, cuando se satisface revoca la clasificación tabú del movimiento. Una aspiración de atributo, cuando se satisface revoca el estatus tabú del atributo. En éste último caso el movimiento puede o no cambiar su clasificación tabú, dependiendo de sí la restricción tabú puede activarse por más de un atributo. Así entonces se pueden tener los siguientes criterios de aspiración:

Aspiración por default: Si todos los movimientos posibles son clasificados como tabú, entonces el movimiento "menos tabú" se selecciona.

Aspiración por objetivo: Una aspiración de movimiento se satisface, permitiendo que un movimiento x sea un candidato para seleccionarse si $F(x)$ < mejor costo.

Aspiración por dirección de búsqueda: Un atributo de aspiración para e se satisface si la dirección de e proporciona un mejoramiento y el actual movimiento es un movimiento de mejora.

4.6.2 Aplicación del método de búsqueda tabú al problema de asignación cuadrático (QAP).

En su forma más simple, el problema de asignación cuadrático (QAP, por sus siglas en inglés) se describe como sigue. Encontrar la asignación óptima de n plantas a n ubicaciones para minimizar el producto acumulado de flujo entre cualesquiera dos plantas por las distancias entre cualesquiera dos ubicaciones. De manera más precisa, si f_{ip} es el flujo entre las plantas i y p, y d_{jq} es la distancia entre las ubicaciones j y q, el problema puede escribirse como:

$$\text{Minimizar} \quad \sum_i \sum_p \sum_j \sum_q f_{ip} d_{jq} x_{ij} x_{pq},$$

sujeto a:

$$\sum_j x_{ij} = 1 \quad \forall_i, \quad \sum_i x_{ij} = 1 \quad \forall_j, \quad x_{ij} = 0,1.$$

La variable x_{ij} es diferente de cero si el objetivo i es asignado a la ubicación j y es cero de otra forma. La función a optimizarse es cuadrática y no convexa, por ejemplo, existe cierto número de óptimos locales, además el conjunto factible es un conjunto de permutaciones.

Una forma sencilla de interpretar el problema planteado es suponer que existen n sitios disponibles y se van a construir n edificios en estos lugares, entonces si f_{ip} es el número de gentes por unidad de tiempo que viajarán entre los edificios i y p, y d_{jq} es la distancia entre los sitios j y q, lo que se desea es encontrar la asignación de los sitios de construcción de manera que la distancia total recorrida se minimice.

Un aspecto interesante de este problema es el hecho de que cuando una de las matrices de la función objetivo se restringe a ser una matriz de permutaciones cíclica, el problema de asignación cuadrático se reduce al problema del agente viajero.

Una formulación equivalente del problema es

$$\text{Minimizar } F(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)},$$

donde π es una permutación del conjunto $N=\{1,2,\ldots,n\}$. Equivalentemente, se desea encontrar una permutación π del conjunto N que minimice el valor de $F(\bullet)$.

4.6.2.1 Elementos de la BT para el QAP.

Las siguientes definiciones describen los elementos de la búsqueda tabú para el QAP. Un movimiento es una transición de una permutación a otra. Un atributo de un movimiento es este caso es simplemente un par no ordenado (i,j) de plantas que cambian sus ubicaciones. El valor de un movimiento es la diferencia entre los valores de la función objetivo después y antes del movimiento. Si este valor es negativo, el movimiento proporciona una mejora. La función *mejor_movimiento* es una que identifica al par (i_{mejor}, j_{mejor}) para el cual el valor de movimiento es el más pequeño. El dominio de la función *mejor_movimiento* es el conjunto de movimientos admisibles. El mejor movimiento no tiene que ser necesariamente uno que mejore.

Una lista tabú de pares (i,j) de *longitud_tabú* se construye y actualiza. Si un par (i,j) pertenece a la lista tabú para una iteración dada no se permite el intercambio de las plantas i y j en esa iteración, a menos que satisfaga el criterio de aspiración que en este caso se cumple si el intercambio de las plantas i y j proporciona un valor de la función objetivo estrictamente mejor que cualquiera de los obtenidos hasta el momento.

Estrategia básica.

La estrategia básica que se utilizará está dividida en cuatro fases: fase de inicialización, fase de mejoramiento, fase de intensificación y fase de diversificación.

Cada iteración de la fase de mejoramiento evalúa todos los posibles movimientos y determina el mejor movimiento valuado admisible. Si ninguna solución mejora después de cierto número prefijado de iteraciones, por ejemplo, $k1xn$, donde $k1$ es número natural, la fase de intensificación inicia.

La fase de intensificación inicia, con una lista tabú vacía, desde la mejor solución encontrada

en la región actual. Si después de $k2$ x n iteraciones con $k2$ número natural no se ha encontrado una mejor solución, la fase de diversificación inicia. El algoritmo termina después de que la búsqueda se ha diversificado a un número dado $k3$ de regiones.

Este proceso se encuentra expresado en pseudocódigo en el algoritmo 1.

La **Fase 1** da la iniciación del algoritmo al encontrar una solución inicial.

En la **Fase 2** se comienza a generar nuevas soluciones a partir de la actual vía a la estructura de vecindades. La Fase 2*a* genera todas las soluciones vecinas a la actual y selecciona a la mejor. En la Fase 2*b* se toma como solución actual a la mejor solución encontrada y se actualizan la matriz tabú y la matriz de frecuencias.

La Fase 2*c* pregunta si ha habido mejoramiento en el valor objetivo de la solución actual en menos de $k1$ x n iteraciones. Si la respuesta es afirmativa se va nuevamente al paso (a) de la Fase 2, en caso contrario se va a la Fase 3.

El objetivo de la Fase 2 del algoritmo 1 es encontrar una solución de calidad a partir de una solución dada. Esta fase termina identificando la mejor solución encontrada en una región de factibilidad en la que se encuentra una solución inicial dada.

ALGORITMO.

Fase 1 *Inicialización:* Generar solución inicial e ir a la Fase 2.

Fase 2 *Mejoramiento:*

a) Evaluar todos los movimientos, determinar y realizar el mejor.
b) Actualizar.
c) Si no hay mejoramiento en $k1$ x n iteraciones ir a la Fase 3.
   De otra forma ir a la Fase 2.

Fase 3 *Intensificación*:

a) Ir a la mejor solución encontrada e inicializar la lista tabú.
b) Evaluar todos los movimientos, determinar y realizar el mejor.
   i) Actualice.
   ii) Si no hay mejoramiento en $k2$ x n iteraciones:
      iii) Si la Fase 3 ha encontrado una mejor solución, ir a Fase 3a.
      De otra forma:
      iv) Si $k3$ reinicios se han realizado, parar.
      De otra forma ir a Fase 4.

De otra forma ir a Fase 3b.

Fase 4 *Diversificación:*

a) Determinar las restricciones tabú de la memoria de término largo.
b) Evaluar todos los movimientos, determinar y realizar el mejor.
i) Actualizar.
Si no hay mejoramiento en $k3$ iteraciones:
ii) Inicializar con la mejor solución.
iii) Eliminar las restricciones tabú adicionales debidas a la memoria de largo término, actualizar el número de reinicios e ir a la Fase 2.
De otra forma vaya a Fase 4b.

La **fase 3** inicia con la mejor solución encontrada hasta el momento, si es que no ha habido mejoramiento en las últimas $k1$ x n iteraciones en el proceso de mejoramiento y se inicializa la tabla tabú. Se procede de manera análoga a la fase de mejoramiento durante $k2$ x n iteraciones, con la diferencia de que, si se encuentra una mejor solución se regresa a la Fase 3a siempre y cuando no se haya alcanzado un número de reinicios, en cuyo caso el algoritmo para. En el caso de que no se encuentren mejores soluciones durante $k2$ x n y no se alcance el número de reinicios, se pasa a la Fase 4.

La **Fase 4** primero determina las restricciones tabú de la memoria de término largo. Determinadas las restricciones de la fase 4a se encuentra una nueva solución y se genera su estructura de vecindades. La Fase 4b evalúa a todas las soluciones vecinas, seleccionando a la mejor y actualizando a la lista tabú y a la matriz de frecuencias. Si no hay mejoramiento en $k3$ iteraciones entonces se inicializa con la mejor solución encontrada, se eliminan las restricciones tabú adicionales, se actualiza el número de reinicios y se pasa a la Fase 2, en caso contrario se continúa con la Fase 4b.

Para la diversificación se utiliza la frecuencia basada en la memoria de término largo. Esta memoria de término largo puede representarse por medio de una matriz LT de nxn donde cada elemento LT_{ij} almacena el número de veces en que cada entrada de una permutación toma el valor de 1 en todas las soluciones examinadas por la búsqueda, de esta manera, si un cierto porcentaje de las entradas con mayor frecuencia, se guardan como tabú para un número suficiente de iteraciones, la búsqueda se forzará para la explotación de nuevas regiones. Alternativamente, una nueva solución inicial puede construirse basada en una matriz de término largo.

Ahora bien, la diversificación se restringe para operarse sólo en ocasiones particulares. En este caso se seleccionan las ocasiones donde ningún movimiento de mejora admisible existe. Por lo general, se utiliza la información de la frecuencia para penalizar a los movimientos que no mejoran la búsqueda mediante el asignar una penalización grande a intercambio de pares con mayores contadores de frecuencia.

Una implantación sencilla se tal técnica de puede realizar como se indica a continuación: Primero, se cuenta el número de veces que cada movimiento m se ha realizado en orden a calcular su frecuencia $f(m)$. Entonces una penalización $p(m)$ se asocia a cada movimiento de la siguiente manera.

$$p(m) = \begin{cases} 0, & \textit{si } m \textit{ alcanza un criterio de aspiración} \\ wf(m), & \textit{de otra forma} \end{cases}$$

donde w es una constante. Entonces el valor del movimiento puede ser:

$$F(x + m) - F(x) + p(m).$$

El peso w depende del problema, del tipo de movimiento y de la vecindad como ya se mencionó antes.

Como se puede observar, la función de penalización depende directamente del criterio de aspiración, por lo que a continuación se proponen algunos de éstos.

Una vez que se han identificado los movimientos factibles y se ha calculado su correspondiente valor de la función objetivo, el estatus tabú de los candidatos factibles se prueba. Supóngase que en una iteración dada (*iter*) el valor de $\pi(s)$, sonde $s = s_i$, cambia de i a i', entonces a la planta $\pi(i)$ le está prohibido ocupar la posición i durante un número específico de iteraciones. La matriz *tiempo_tabú* se utiliza para forzar esta restricción tabú. El elemento $(i, \pi(i))$ de la matriz *tiempo_tabú* contiene el número de iteración en la cual se le permite de nuevo a la planta $\pi(i)$ ser asignada a la posición i. Si el valor de la función objetivo de la solución anterior al movimiento es mejor que el valor de la función objetivo después de éste, entonces el tiempo tabú se calcula como:

$$\textit{tiempo\_tabú}(i, \pi(i)) = 0.25\ n^2,$$

de otra forma

$$\textit{tiempo\_tabú}(i, \pi(i)) = 0.75\ n^2.$$

Este es un criterio de aspiración implícito, donde a los elementos que contribuyen a soluciones con "buenos" valores de la función objetivo se les permite regresar más rápidamente para la prueba de soluciones actuales. El máximo tiempo que un movimiento puede clasificarse como tabú es el equivalente al 75% del tamaño de la estructura de *tiempo_tabú*. Esto significa que una iteración dada un mínimo del 25% de los movimientos posibles no se clasifican como tabú.

Ahora bien un criterio de nivel de aspiración de tipo explícito es el siguiente: el estatus tabú de un movimiento se elimina si el movimiento permite la búsqueda a una nueva solución *mejor de*

la región. La solución mejor de la región es aquella con el mejor valor de la función objetivo durante digamos las n^2 últimas iteraciones, se consideró el valor de n^2 puesto que no es ni muy pequeño como n ni demasiado grande. Se observa que este nivel de aspiración es más flexible que el utilizado típicamente en las implantaciones de BT, donde la mejor solución sobre todas sirve para propósitos similares.

Otro criterio explícito es una aspiración por *dirección del movimiento*. Para implantar este criterio es necesario construir una matriz *dirección_tabú*. Si el movimiento $(i, \pi(i))$ llega a ser tabú durante una fase de mejoramiento, entonces el elemento $(i, \pi(i))$ de la matriz de *dirección_tabú* contiene el número de la iteración de cuando la fase de mejoramiento comenzó, de otra forma al elemento se le da el valor de cero. El movimiento candidato factible $(i, \pi(i))$ es elegible para seleccionarse en la iteración *iter* si:

F(movimiento posterior) $\neq$ F(movimiento anterior) y ocurre al menos una de las siguientes:

$$\begin{cases} - \; tiempo\_tabú(i,\pi(i)) < iter. \\ - \; F(movimiento\ posterior) \mid F(mejor\ de\ la\ región) \\ - \; dirección\_tabú(i,\pi(i)) \neq 0 \; y \; dirección\_tabú(i,\pi(i)) = fase\_actual. \end{cases}$$

Donde F(•) es el valor de la función objetivo y *fase_actual* es cero si la búsqueda se encuentra en una fase de no mejoramiento o el número de la iteración donde la actual fase de mejoramiento comenzó. El mejor movimiento de cualquier iteración es el movimiento elegible con mejor valor de movimiento.

4.6.2.2 Manejo dinámico de la lista tabú.

El *manejo de la lista tabú* significa la actualización de ésta, por ejemplo, la decisión de cuántos y cuáles movimientos se pondrán como tabú para una iteración de la búsqueda. La mayoría de las implantaciones de la búsqueda tabú utilizan un manejo de la lista tabú de tipo estático. De manera más precisa, los movimientos permanecen como tabú durante un número de iteraciones fijo, por lo que la eficiencia del algoritmo depende de la elección de la duración del estatus tabú, equivalentemente, de la longitud de la lista tabú.

Este manejo de la lista tabú de tipo dinámico permite el examen más detallado de la región factible, por lo que es posible romper ciclos y diversificar la búsqueda.

Ahora bien también se puede manejar una longitud de LT de manera dinámica, donde la longitud de la lista tabú varia de manera dinámica a través de diferentes configuraciones pasando de una a otra si ningún mejoramiento se encuentra en un número dado de iteraciones. Este conjunto de configuraciones permite un examen más detallado de la región factible mediante el incremento

y decremento del número de movimientos tabú actuales, vía las configuraciones. Además se propone un componente aleatorio que actúa cada vez que la lista tabú cae en la configuración 1 como un elemento de seguridad adicional contra el ciclado.

4.6.3 Resultados computacionales.

Los siguientes resultados son presentados por De Los Cobos[3] [1994], de la aplicación de varias versiones de BT para el problema de asignación cuadrático (QAP), sobre el conjunto estándar de problemas de tamaño n = 5, 6, 7, 8, 12, 15, 20 y 30, los cuales se utilizan actualmente de manera amplia para probar la eficiencia de los algoritmos. Los procesos usados para la experimentación fueron implementados en lenguaje C en una PC con procesador 486DLC a 40 Mhz.

Las versiones que se utilizaron de la búsqueda tabú tienen las siguientes características:

BT1 : Estructura básica de BT, por ejemplo, se consideró la longitud de la lista tabú como un parámetro constante, el criterio de nivel de aspiración fue el estándar de la literatura, es decir, el nivel de aspiración se satisface si el movimiento a realizarse mejora el valor de la función objetivo con respecto a todos los movimientos realizados.

BT2 : Se consideró una estructura básica de la BT y se agregó el criterio de nivel de aspiración proporcionado por la matriz *tiempo_tabú* dado en la sección anterior.

BT3 : En el marco de la BT se consideró el criterio de nivel de aspiración estándar, así como, un manejo de la lista tabú para programación en paralelo.

En las tablas 2 y 3 se presentan entre paréntesis los mejores valores encontrados en la literatura, así como, el número de iteraciones en las que alcanzó, antes de llegar a un número máximo de iteraciones impuesto, el cual dependía del tamaño del problema y se proporciona en la Tabla 1. Los valores con asterisco son los valores óptimo.

Tabla 1

n	5	6	7	8	12	15	20	30
iter	15	15	25	25	700	2500	7500	12000

Los resultados presentados fueron los siguientes:

---

3 De los Cobos, Sergio. *La técnica de búsqueda tabú y sus aplicaciones. Tesis doctoral en investigación de operaciones*. México, UNAM. División de Estudios de Posgrado de la Facultad de Ingeniería. 1994.

Tabla 2

Valores de la función objetivo y número de iteraciones.

n	Sol Inic.	Valor Inic.	Long Tabú	BT1		BT2		BT3		Valor Mejor
				Valor	Iter	Valor	Iter.	Valor	Iter	
5	1	66	5	(58)	1	50	11	50	14	50*
	2	58	5	(58)	0	50	11	50	11	50*
	3	72	5	50	1	50	1	50	1	50*
	4	68	5	(52)	6	50	12	50	4	50*
	5	92	5	(52)	5	50	12	50	12	50*
6	1	86	4	86	0	86	0	86	0	86*
	2	110	4	86	2	86	2	86	2	86*
	3	108	4	86	6	92	1	92	6	86*
	4	116	4	86	1	86	1	86	1	86*
	5	116	4	86	8	86	11	86	8	86*
7	1	174	5	148	23	148	21	148	23	148*
	2	240	5	148	10	148	10	148	10	148*
	3	216	5	148	4	148	4	148	4	148*
	4	228	5	148	18	148	18	148	18	148*
	5	202	5	148	19	148	19	148	19	148*
8	1	214	6	214	10	214	16	214	10	214*
	2	322	6	214	5	214	5	214	5	214*
	3	201	6	214	7	214	15	214	7	214*
	4	320	6	214	5	214	5	214	5	214*
	5	288	6	214	4	214	4	214	4	214*
12	1	784	9	(586)	57	578	683	578	368	578*
	2	784	9	578	177	(586)	110	578	18	578*
	3	874	9	578	12	578	16	578	11	578*
	4	850	9	(586)	23	578	84	578	53	578*
	5	746	9	578	5	578	5	578	4	578*

Tabla 2 (Continuación)
Valores de la función objetivo y número de iteraciones.

n	Pto. Inic	Valor Inic.	Long. Tabú	BT1		BT2		BT3		Valor Mejor
				Valor	Iter.	Valor	Iter.	Valor	Iter	
15	1	1438	10	1150	1498	1150	46	1150	201	1150*
	2	1612	10	1150	1468	1150	375	1150	231	1150*
	3	1596	10	1150	2368	(1152)	53	(1152)	51	1150*
	4	1610	10	1150	1424	(1152)	181	(1152)	305	1150*
	5	1626	10	(1152)	31	(1152)	32	(1152)	32	1150*
20	1	3444	12	2580	7435	2570	539	2570	1014	2570
	2	3302	12	2570	5626	2570	954	2570	1841	2570
	3	3540	12	2570	1869	2570	704	2570	2197	2570
	4	3436	12	2570	918	2570	898	2570	887	2570
	5	3322	12	2570	1191	2570	613	2570	2408	2570
30	1	8060	18	6124	11048	6124	3628	(6150)	171	6124
	2	7758	18	6124	7293	(6128)	6223	6124	10692	6124
	3	8172	18	(6128)	972	6124	9842	(6148)	7740	6124
	4	7638	18	(6128)	2713	6124	3995	(6128)	1039	6124
	5	8224	18	6124	5748	6124	3839	(6148)	7881	6124

La tabla 3 proporciona los promedios de los valores y el número de iteraciones de las corridas para cada uno de los diferentes tamaños de los problemas, la tabla 4 proporciona las eficiencias de los promedios de los valores encontrados con respecto de los mejores valores reportados en la literatura, finalmente, la tabla 6 indica en su lado izquierdo el número promedio de iteraciones para los casos en que se encontraron los valores óptimos o los mejores reportados, y en la parte derecha de la tabla están los cocientes del número de iteraciones de cada uno de los algoritmos, indicando la proporción de iteraciones de cada uno de éstos.

Tabla 3
Promedios de las corridas

n	BT1		BT2		BT3		Valor
	Valor	Iter.	Valor	Iter.	Valor	Iter.	Mejor
5	54	2.6	50	9.4	50	8.4	50*
6	86	3.4	87.2	3	86	3.4	86*
7	148	14.8	148	14.4	148	14.8	148*
8	214	6.2	214	9	214	6.2	214*
12	581.2	94.8	579.6	179.6	578	90.8	578*
15	1150.4	1297.6	1151.2	137.4	1151.2	163.8	1150*
20	2570	3407.8	2570	743.4	2570	1669.4	2570
30	6125.6	5555	6124.8	5505.4	6141.2	5506.6	6124

La eficiencia se calculó tomando como base el mejor valor reportado z_{mej}, y se usó la ecuación:

$$eficiencia = 1 - \frac{z - z_{mej}}{z_{mej}}$$

Tabla 4

Tabla de eficiencias sobre valores promedios			
5	0.920	1	1
6	1	0.986	1
7	1	1	1
8	1	1	1
12	0.994	0.997	1
15	0.999	0.998	0.998
20	1	1	1
30	0.999	1	0.997

En esta última tabla no se puede observar que alguno de los algoritmos sea consistentemente más eficiente. En la tabla 3.5, se presenta el promedio de iteraciones para cuando se alcanzaron los mejores valores reportados, así cómo, los porcentajes de su número de iteraciones.

Tabla 5
Promedio de iteraciones

	Prom. de Iteraciones			Porcentajes		
n	**BT1**	**BT2**	**BT3**	**BT1**	**BT2**	**BT3**
5	-	9.4	8.4	-	1.119	1
6	3.4	3	3.4	1.133	1	1.113
7	14.8	14.4	14.8	1.027	1	1.027
8	6.2	9	6.2	1	1.451	1
12	131.33	197	90.8	1.446	2.169	1
15	1613.75	210.5	215.5	7.666	1	1.023
20	3407.8	743.4	1669.4	4.584	1	2.245
30	8030	5326	10692	1.507	1	2.007

Se puede observar que el método BT2 proporciona en la mayoría de los casos el menor número de iteraciones para alcanzar el mejor valor, siendo en algunos casos una relación de más de 7 a 1 en el número de iteraciones.

4.6.4 Aplicaciones de Búsqueda Tabú en problemas de optimización.

La técnica de búsqueda tabú como base para la construcción de algoritmos de búsqueda de soluciones a problemas combinatorios aparece en últimas fechas con mucha frecuencia en la literatura de investigación de operaciones. Los resultados de aplicaciones realizadas por Glover[4] [1993] al problema de ubicación de plantas presentado como modelo se asignación cuadrática, motivaron en otros investigadores a aplicar la técnica de BT en otros problemas de programación entera y mixta.

Una aplicación interesante al problema de calendarización de la producción es la presentada por Barnes y Laguna[5] [1993], quienes aplican un algoritmo basado en BT que resuelve el problema de asignación de tiempo mínimo en la secuencia de realización de *i* trabajos en *j* maquinas, con un determinado tiempo de ejecución de cada trabajo en cada maquina y que incluye una variable de penalización por no cumplimiento en un determinado tiempo. El modelo presenta soluciones satisfactorias a casos de 20 trabajos en 5 maquinas en un tiempo de 7 segundos y de 30 trabajos en 10 maquinas en 21 segundos, utilizando un programa construido en lenguaje "C" en una maquina con procesador 386.

---

4 Glober, Fred. *A user's guide to tabu search.* En Annals of Operations Research. Vol. 41. 1993. pp 3 - 28.

5 , Barnes, J. Wesley y Laguna, Manuel. *A tabu search experience in production scheduling.* En Annals of Operations Research. Vol. 41. 1993. pp 141 - 156.

Para el problema del agente viajero Knox[6] [1994] aplica un algoritmo de BT que encuentra buenas soluciones a diversos casos, en un tiempo que se indica en la siguiente relación:

No. de ciudades	Tiempo de ejecución
25	9.5 Segundos
30	33.0 "
42	183.0 "
50	621.1 "
57	1,849.9 "
75	8,486.2 "

Por otra parte, es notable observar en la actualidad aplicaciones a de la técnica de búsqueda tabú en problemas combinatorios de la vida real. Por ejemplo, Semet y Tallard[7] [1994] resuelven un problema de asignación de rutas a 21 camiones y 7 trailers que deben abastecer 145 almacenes, con la posibilidad de que los trailers provean a los camiones en determinadas rutas, permitiendo que un número determinado de camiones pueda atender más de una ruta en cierto tiempo.

Otro caso interesante es el de Laguna y Kelly[8] [1993], quienes utilizan un modelo entero mixto para un problema de planeación de la producción en una planta, con una secuencia que tiene permutaciones de tiempo.

Por su parte Icmeli y Erenguc[9] [1994] presentan resultados para un problema de proyección de inventarios de recursos restringidos con flujo de efectivo descontado, con base en un programa que aplica BT, al cual le lleva 6 segundos encontrar una solución para una relación de 26 recursos, 17.4 segundos para 35 recursos y 46 segundos para 51 recursos.

La técnica de búsqueda tabú, por ser un método iterativo que explora, evalúa y selecciona un conjunto de soluciones de un problema, requiere de cierta capacidad de memoria expandible que le permita el almacenamiento de las listas tabú de movimientos que genera y que son elementos centrales del proceso, además de mantener el registro de un elevado número de iteraciones.

---

6 Knox, John. *Tabu search performance on the symmetric traveling salesman problem.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 867 - 876.

7 Semet, Frédéric y Taillard, Eric. *Solving real-life vehicle routing problems efficiently using tabu search.* En Annals of Operations Research. Vol. 41, 1993. pp 469 - 488.

8 Laguna, Manuel y Jaimes P., Kelly. *Master production scheduling in a single facility with sequence-dependent changeover times.* Mimeo. Septiembre, 1993.

9 Icmeli, Oya y Erenguc, Selcuk. *A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 840 - 846.

Una de las ventajas de la BT es que a pesar del elevado número de iteraciones que deben realizarse conforme aumenta el número de variables, el tiempo de procesamiento que utiliza para obtener una buena solución óptima es muy corto.

Al igual que cualquier otro método heurístico, la BT genera soluciones que dependerán de la naturaleza del problema que se deba resolver, su planteamiento, así como la estructura del programa que realice el proceso iterativo de búsqueda y la capacidad del computador que se use.

La popularidad que en últimas fechas ha cobrado esta técnica hace pensar en el desarrollo de nuevas propuestas, sobre todo encaminadas a reducir el número de iteraciones que debe procesarse, así como el espacio de memoria que utiliza.

En términos generales, la perspectiva de la BT es hacia su implementación en la mayoría de los problemas de programación lineal entera y combinatoria, a fin de investigar su factibilidad como un método eficiente para la mayoría de los casos.

CONCLUSIONES.

Los problemas de optimización combinatoria son en la actualidad un importante campo de investigación por sus características y sobre todo porque tienen una abundante aplicación en la vida real, sobre todo en los casos de administración eficiente del uso de recursos escasos para el incremento de la productividad y de los beneficios de reducir costos .

Algunos de los problemas de tipo combinatorio más representativos son:

El problema tipo de la mochila: que se usa para resolver problemas de inversiones, problemas de confiabilidad de redes, de cargo fijo, de selección de proyectos, corte de inventarios, control de presupuestos.

El problema de asignación de tareas: que se utiliza para optimizar los recursos en los procesos productivos

Problemas de cobertura de conjuntos: en los que se encuentran el problema de ubicación de instalaciones y el problema de flujo de redes con cargo fijo.

El Problema del vendedor viajero: que consiste en encontrar un recorrido que utilice la menor distancia de entre un múltiples alternativas

Los problemas con funciones no lineales separables: en los que la función objetivo y las restricciones son separables.

Problemas con restricciones disyuntivas: que ocurre cuando se tienen condiciones en que se debe elegir una de entre dos restricciones.

Problemas de asignación cuadrática: en los que se ubica a los problemas de localización.

Una característica de los problemas de tipo combinatorio radica en que éstos, en la mayoría de los casos no guardan dificultad alguna en su planteamiento. Sin embargo, el conflicto surge cuando el número de variables es muy grande, y la combinación de éstas en la búsqueda de los valores óptimos crece en muchas de las veces de manera exponencial.

En la actualidad, el objetivo hacia el cual se dirigen las investigaciones en el campo de la programación combinatoria, consiste en encontrar algoritmos que resuelvan este tipo de problemas de manera óptima y eficiente. Optima en cuanto a los resultados que presenten y eficiente en cuanto al tiempo de cómputo de su procesamiento que utilicen.

En contraste con la programación lineal, la mayor parte de los problemas de programación combinatoria pueden formularse de varias maneras. De hecho en su formulación radica la posibilidad de encontrar algoritmos que sean mejores en algunos casos.

Una práctica común en programación lineal consiste en relacionar el tiempo de procesamiento de cómputo con el tamaño de problema. Tradicionalmente, el tamaño de un problema de optimización es descrito por su número de variables y de restricciones. Estos dos parámetros, sin embargo, no pueden ser adecuados.

Para el caso de la programación combinatoria, el tiempo de ejecución requerido por un algoritmo para resolver un problema es uno de los parámetros importantes para medir en la práctica la bondad de un algoritmo pues, entre otros factores, el tiempo de ejecución equivale a tiempo de utilización de la computadora y, en consecuencia, su costo económico.

Para que una computadora resuelva un problema es preciso indicarle qué operaciones debe realizar. Es decir, se le debe de construir el procedimiento para resolver el problema. Dicho procedimiento se llama algoritmo. Un algoritmo describe el método mediante el cual se realiza un programa y consiste en una secuencia de instrucciones, las cuales, realizadas adecuadamente, dan lugar al resultado deseado. Cada paso del algoritmo se expresa mediante una instrucción o sentencia en el programa lo ejecuta. Por tanto, un programa de computadora consiste en una serie de instrucciones dictadas en forma sistemática y ordenada, orientadas a la realización de una serie de operaciones lógicas, aritméticas y de decisión, cuyo fin es una salida que corresponda al procesamiento que la computadora hace de dichas instrucciones a partir de ciertos datos de entrada expresados en forma correcta.

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. Por tanto, se supone que todas las operaciones básicas se ejecutan en una misma 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.

En la actualidad el tiempo requerido por un algoritmo para su ejecución, depende fundamentalmente del programa que representa el algoritmo, lenguaje de programación elegido y la computadora que lo ejecuta.

Por otra parte, la complejidad de los problemas de programación está determinada por la existencia o no de un algoritmo que lo resuelva en forma satisfactoria. Quizá para una instancia dada, se tenga un algoritmo eficiente, pero lo que interesa es el comportamiento del algoritmo para el peor conjunto de datos posibles, es decir, la instancia más mal comportada. Otro punto interesante es pensar en instancias grandes del problema. En razón de ello es que los problemas se clasifiquen de acuerdo al trabajo que se tiene en resolverlos. Específicamente, se pueden clasificar los problemas en cuatro clases de acuerdo a su grado de dificultad:

Problemas Indecidibles.
Problemas Polinomiales **(P).**
Problemas intratables.
Problemas no polinomiales determinísticos **(NP).**
Problemas no polinomiales completos **(NP-completos).**

La clasificación anterior ubica la importancia de conocer o descubrir algoritmos polinomiales para el problema que se tenga que resolver. Un problema puede pertenecer a la clase **P** y sin embargo puede ser que no se conozca un algoritmo polinomial que lo resuelva, aunque tal algoritmo exista.

En investigación de operaciones se han presentado diversos algoritmos no determinísticos que en la práctica, han demostrado que son eficientes resolver lo problemas de optimización combinatorios. La mayoría de estos algoritmos se basan en los procedimientos de programación lineal y en la estructura del método simplex, por lo que su mecanismo es iterativo y, dependiendo del tamaño del problema, alcanzaran la respuesta en tiempo polinomial o exponencial.

Los algoritmos de este tipo son los que se basan en los métodos de **planos de corte, de ramificación y acotamiento** y de **enumeración implícita.** La eficiencia de estos métodos está determinada por su convergencia a la solución óptima en problemas de programación entera y mixta. Sin embargo, su procedimiento de búsqueda implica la generación nuevas variables, lo que determina un tiempo de procesamiento cada vez mayor conforme el tamaño del problema se incrementa, llegando a ser este factor, el tiempo, lo que establece muchas veces su imposibilidad de aplicación.

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

Los *métodos de corte*, que se desarrollan principalmente para problemas *lineales* enteros, comienzan con el óptimo continuo. Sumando sistemáticamente restricciones secundarias especiales, que representan básicamente condiciones necesarias de integridad, el espacio de soluciones continuo se modifica de manera gradual hasta que su punto extremo óptimo continuo satisface las condiciones enteras. El nombre métodos de corte surge del hecho que las restricciones secundarias sumadas cortan (o eliminan) efectivamente ciertas partes del espacio

de soluciones que no contienen puntos enteros factibles.

La desventaja de los métodos de planos de corte, 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.

Por su parte, el *método de ramificación y acotamiento* redondea y acota variables enteras, resultantes de la solución de los problemas lineales correspondientes. Este proceso de acotamiento y redondeo se hace de una manera secuencial lógica heurística, que permite eliminar con anticipación un buen número de soluciones factibles alejadas del óptimo a medida que se itera. Esta técnica también resuelve el problema entero considerando su versión continua. Pero a diferencia de los métodos de corte, el método de ramificar y acotar se aplica directamente a ambos tipos de problemas, el *puro* y el *mixto*.

La eficiencia de cómputo de este método puede aumentarse introduciendo el concepto de *acotamiento*. Este concepto indica que, si la solución óptima continua de un subproblema proporciona un valor de la función objetivo peor que el asociado a la mejor solución entera disponible, no vale la pena explorar adicionalmente el subproblema. En este caso se dice que el subproblema está *exhausto* y, por consiguiente, puede desecharse. En otras palabras, una vez que se encuentra una solución factible entera, su valor de la función objetivo asociado, puede ser utilizado como una *cota* (superior en el caso de minimización e inferior en el caso de maximización) para descartar subproblemas inferiores.

Los problemas específicos generados dependen de la variable seleccionada para efectuar la ramificación. Desafortunadamente, no existe la forma *mejor* definida; para seleccionar la variable ramificadora o la sucesión específica con que debe examinarse el subproblema. Pero existen reglas empíricas que mejoran el proceso. Estas reglas usualmente están implantadas en la mayoría de los códigos de ramificar y acotar.

Una desventaja básica del algoritmo anterior consiste en que es necesario resolver un programa lineal completo en cada nodo. En problemas grandes, esto podría consumir mucho tiempo, particularmente cuando la única información necesaria en el nodo puede ser su valor óptimo de la función objetivo. Este punto se aclara notando que una vez que se obtiene una *buena* cota, muchos nodos podrían descartarse del conocimiento de sus valores objetivos óptimos.

Los métodos de *enumeración implícita* son métodos heurísticos, basados en la lógica, que como los métodos de ramificación y acotamiento resuelven problemas enteros, sin tener que analizar todas las posibles alternativas. Así como el método de planos de corte resuelve un problema entero mediante la modificación de la región de factibilidad del problema lineal correspondiente, y el método de bifurcación y acotación mediante la ramificación de problemas que obligan a una variable fraccionada a tomar el valor entero inmediato mayor o menor de la fracción, el método de enumeración implícita resuelve problemas enteros mediante la aceptación o rechazo implícito de ciertas alternativas.

Una observación particular acerca de la enumeración implícita es que el tiempo de cómputo depende de los datos. El ordenamiento especifico de las variables y restricciones puede tener un efecto directo sobre la eficiencia del algoritmo. Por ejemplo, las restricciones deberán estar ordenadas con la más restrictiva en la parte superior, mientras que las variables podrían ordenarse según el orden ascendente de sus coeficientes en la función objetivo (coeficientes no negativos). Ambas condiciones son favorables para producir el examen más rápido de las soluciones parciales.

Aunque se han desarrollado varios algoritmos finitos para los problemas combinatorios, ninguno de estos métodos es uniformemente eficiente desde el punto de vista computacional, principalmente cuando aumenta el tamaño del problema. A diferencia de los programas lineales, donde problemas muy grandes se han resuelto en un tiempo razonable, los algoritmos que se aplican a problemas combinatorios han sido erráticos en el comportamiento de resolución.

En la práctica estos problemas de programación pueden ser tan complejos que los modelos que se construyan para abordarlos no se puedan resolver mediante los algoritmos tradicionales. En este caso se hace necesario aplicar la *heurística* para generar procedimientos que ofrezcan *buenas* soluciones.

Una *heurística* es una apelación intuitiva a una regla interna o particular para trabajar con un aspecto del problema que se esté resolviendo. En el contexto de la programación matemática, a menudo se emplea la heurística en conjunción con estrategias de resolución de problemas más rigurosas o generales, o como caso particular de ellas.

El factor principal que motiva a recurrir a procesos alternativos se debe a que no todos los modelos matemáticos en investigación de operaciones poseen algoritmos o métodos de solución que *converjan* siempre al nivel óptimo. Existen dos razones de ser de esta dificultad.

En primer lugar, se puede probar que el algoritmo de solución converge al nivel óptimo, pero solo en sentido técnico. La convergencia técnica señala que hay un límite superior finito para el número de iteraciones, pero no indica cuan alto puede ser este límite. Por lo tanto, se pueden consumir horas de tiempo de la computadora sin llegar a la iteración final, lo que es peor aún que si las iteraciones se detienen en forma prematura antes de llegar al nivel óptimo, generalmente no se puede medir la calidad de la solución obtenida en relación con el nivel óptimo verdadero.

En segundo lugar, la complejidad del modelo matemático puede hacer imposible idear un algoritmo de solución. En este caso el modelo se puede mantener infactible en término de cálculos.

Las dificultades evidentes en los cálculos de los modelos matemáticos ha obligado a los analistas a buscar otros métodos de cálculo. Estos métodos también son de naturaleza iterativa, pero no garantizan la optimalidad de la solución final. En cambio simplemente buscan una buena solución al problema. Tales métodos suelen denominarse *heurísticos* porque su lógica está basada

en reglas o métodos prácticos que lleven a la obtención de una buena solución. La ventaja de los métodos heurísticos es que normalmente implican un menor número de cálculos cuando se comparan con algoritmos exactos. Asimismo, debido a que están basados en reglas prácticas, normalmente son más sencillos de explicar por los usuarios que no están orientados a las matemáticas.

En investigación de operaciones, los métodos heurísticos se pueden utilizar dentro del contexto de un algoritmo de optimización exacto, a fin de aumentar la velocidad del proceso para alcanzar el nivel óptimo. La necesidad de *fortalecer* el algoritmo de optimización se hace más evidente con modelos a gran escala. También, se utilizan para obtener una buena solución al problema. La solución resultante no tiene la garantía de ser óptima y de hecho, su calidad en relación con el nivel óptimo real, puede ser difícil de determinar.

En la actualidad, con los avances tecnológicos en el área de informática se ha propiciado la creación de nuevas propuestas para la solución de problemas de optimización que se resuelven a partir de métodos heurísticos, basados lo que se ha dado llamar *Inteligencia Artificial*.

La Inteligencia Artificial es una tecnología que se ocupa de la comprensión de la inteligencia y del diseño de sistemas inteligentes, entendiendo por tales aquellas que presentan características asociadas al entendimiento humano como el razonamiento, la comprensión del lenguaje hablado y escrito, el aprendizaje, la toma de decisiones y otras similares.

A diferencia de la informática convencional, cuyos procesos son de tipo algorítmico, perfectamente definidos y estructurados *a priori*, con una secuencia de operaciones predefinida y que se repite en su totalidad en las mismas condiciones de partida, la Inteligencia Artificial, por el contrario, actúa bajo un proceso de tipo lógico abierto, en el que el ordenador dispone de unas reglas de inferencia y de una base de conocimientos, y en función de ambas y de la información que adquiere, o se le suministra, inicia un proceso de búsqueda empírico o *heurístico* con notables dosis de *complejidad*, *incertidumbre* y ambigüedad propias.

Tomando como base que la inteligencia artificial se apoya en ciertos mecanismos de procesamiento que los hace apropiados para la solución de problemas que requieren de técnicas heurísticas de búsqueda basadas en operaciones y especialmente en mecanismos de razonamiento lógicos y analógicos, se ha dado lugar a que en la actualidad se propicia un enlace bilateral entre investigación de operaciones y la inteligencia artificial.

Dentro de toda una serie de nuevas técnicas basadas en inteligencia artificial, recientemente se ha puesto especial atención en cuatro métodos de manipulación en la solución de problemas complejos de decisión: **Algoritmos genéticos, redes neuronales, recocido simulado y búsqueda tabú.** Los dos primeros; algoritmos genéticos y redes neuronales, están inspirados por los principios derivados de las ciencias biológicas; el recocido simulado se deriva de las ciencias físicas, principalmente de la termodinámica. La búsqueda tabú se basa en las técnicas de solución de problemas inteligentes.

Cada uno de estos métodos guarda un enfoque diferente para la solución de un mismo problema. Algunos presentan, en ciertos casos soluciones más confiables que los otros, sin que ello demerite los procesos empleados en cada caso.

Un algoritmo genético se define como *una técnica aleatoria de búsqueda que imita la evolución natural*. Esta se inicia seleccionando una población de soluciones generadas aleatoriamente para el problema que se trate. Las soluciones se mueven de una generación de soluciones a otra por crianza de nuevas soluciones usando solo evaluación objetiva y el llamado operador genético. En este sentido, dicho proceso puede ser clasificado como una técnica de iteraciones de asignación y mejoramiento. En general, el algoritmo genético trabaja con un código de las variables en lugar de las variables mismas. Típicamente una solución esta representada por una cadena de *bits*, también llamados *cromosomas*. Cada posición de un bit es llamado un *gene*, y los valores de cada uno que de los genes puede tomar se llaman *alleles*.

El algoritmo genético básico tiene tres operadores principales: *la reproducción*, donde los cromosomas (o soluciones) son copiados para la siguiente generación, con igual probabilidad, con base en la cualidad de el valor de su función objetivo; el *cruzamiento*, donde son seleccionadas aleatoriamente pares de cromosomas que son unidos, creando nuevas parejas y; la *mutación*, que es la alteración aleatoria del allele de un gene. La mutación diversifica el espacio de búsqueda y protege de pérdida de material genético que pueda ser causado por la reproducción y el cruzamiento.

Los algoritmos genéticos ofrecen resultados aceptables a problemas de optimización en los que el objetivo es conseguir una secuencia de objetos que presente un resultado, de entre una gran cantidad de posibles resultados, que satisfaga un determinado número de restricciones. La calidad de los resultados varia de acuerdo con las características del problema en cuanto a su modelo, por lo que existen casos para los cuales su aplicación no aporta importantes avances.

Una de las ventajas de los algoritmos genéticos y quizá la principal, es su capacidad para incluir múltiples restricciones, además de funciones objetivo no lineales y no convexas. Otro aspecto que favorece el uso del algoritmos genéticos es su velocidad de procesamiento, lo que se traduce en tiempos cortos para la obtención de buenos resultados y por tanto la posibilidad de experimentar con múltiples poblaciones. Además, la convergencia al óptimo a través de este método se obtiene con base a la realización de múltiples experimentos con poblaciones diferentes, sin que el aumento en el tamaño de la población sea un factor de mejoramiento de la solución.

Sin embargo, la realización de múltiples corridas para poder presentar una solución que pueda considerarse como satisfactoria representa una de sus principales desventajas. En la mayoría de los casos reportados se hacen arriba de 50 experimentos para poder encontrar un óptimo considerado como bueno, además de que cada experimento es independiente de otros experimentos, por lo que resultados de cada uno pueden ser peores o mejores que los otros.

Los algoritmos genéticos difieren de las técnicas de optimización tradicionales en diversos aspectos. Estos trabajan con una codificación de las variables (típicamente como cadenas) más que con las variables en sí mismas, y utilizan las reglas de transición probabilística para moverse de una población de soluciones a otras más que de una solución sencilla a otra. La más interesante e importante característica de los AG es que utilizan tan solo evaluaciones de la función objetivo. Esto es, no utilizan ninguna información sobre diferenciabilidad, convexidad o alguna otra característica auxiliar. Esto hace que los AG sean fáciles de utilizar y de implantar en gran variedad de problemas de optimización.

Otra técnica importante basada en los principios de inteligencia artificial es la de redes neuronales.

El modelo de redes neuronales hacia el cual se han dirigido las investigaciones sobre su aplicación a problemas combinatorios está basado en la propuesta de Hopfield. La característica de interés de este modelo es la rápida minimización de una función de energía. Aunque se garantiza que la red converja a un mínimo de la función de energía, no se garantiza que vaya a converger al mínimo *más bajo* de energía: la solución será, con toda probabilidad, una buena solución, pero no necesariamente la mejor.

Las investigaciones revisadas en este tranbajo, sugieren que el modelo de redes neuronales de Hopfield no presenta en la práctica soluciones que compitan satisfactoriamente con otros métodos heurísticos en su aplicación a problemas combinatorios que involucren un número grande de datos, en razón de que utiliza un número grande de neuronas y ocupa mucha memoria por las características de la función de energía que debe minimizarse.

Sin embargo, con las avances actuales en los sistemas de información, las aplicaciones en que los modelos de redes neuronales han demostrado ser muy útiles, es en problemas de decisión relacionados con el reconocimiento de patrones. Las redes que se manejan para resolver estos problemas poseen propiedades deseables que parecen ideales para ciertos problemas de asignación de recursos.

El algoritmo más comúnmente usado con redes neuronales para enfrentar estos problemas de decisión es el algoritmo de retropropagación. Este consiste en un conjunto neuronal adaptativo que está configurado como un sistema de retroalimentación integrado por una capa de nodos de entrada, una capa de nodos de salida y algún número de nodos ocultos que representan funciones con característica no lineal. Durante el entrenamiento, los ejemplos de los patrones y su respuesta correcta asociada están representados por la red, la cual trata de minimizar, sobre un conjunto de entrenamiento entero, el error entre la capa de nodos de actividades de salida y la respuesta proporcionada.

Las aplicaciones de este tipo son utilizadas satisfactoriamente en problemas de determinación de carteras de inversión, determinación de riesgo asociado a crédito, clasificación de riesgo, así como a problemas de asignación de trabajos.

Por su parte, el método de **recocido simulado**, como un proceso computacional, se basa en el proceso físico de *recocido*, en el cual algunas sustancias físicas, tales como los metales, son derretidos subiendo la temperatura a niveles elevados y luego enfriados gradualmente hasta que algún estado sólido es alcanzado. El objetivo de este proceso es producir en una función objetivo un estado mínimo de energía final.

El proceso de recocido consta de tres factores que deben determinarse: a) el valor inicial al que es elevada la temperatura; b) el criterio que se utilizará para decidir cuando la temperatura del sistema debe ser reducido y c) la cantidad a la cual la temperatura será reducida en cada tiempo.

En investigación de operaciones el algoritmo de recocido simulado es una técnica aplicable a problemas de optimización combinatoria cuya eficacia depende de la habilidad que se tenga para identificar y definir cada uno de los factores que deben ser involucrados. La caracterización matemática del proceso de recocido se basa en la distribución de Boltzmann, la cual representa una función de densidad de probabilidad de un estado de energía con respecto a una determinada temperatura.

En la mayor parte de las referencias que existen sobre las aplicaciones del recocido simulado se le presenta en combinación con otros métodos heurísticos, formando estrategias híbridas para atacar problemas combinatorios que caen en el grupo de los llamados NP-completos.

En lo que se refiere a la **búsqueda tabú**, ésta puede definirse como un procedimiento heurístico para resolver problemas de optimización cuya característica principal es la de *escapar* de la optimalidad local. La búsqueda tabú en una forma simple presenta dos elementos claves: La de restringir la búsqueda mediante la clasificación de ciertos movimientos como prohibidos (es decir tabú) y la de liberar la búsqueda mediante una función de memoria de término corto que proporciona una "estrategia de olvido".

Para problemas grandes, donde las vecindades pueden tener muchos elementos o para problemas donde esos elementos son muy costosos de examinar, es importante aislar a un subconjunto de la vecindad, y examinar este conjunto en vez de la vecindad completa. Esto puede realizarse a pasos permitiendo al subconjunto de candidatos expandirse si los niveles de aspiración no se encuentran.

Los factores que dan vitalidad a la estrategia de búsqueda del método de BT y que merecen ser resaltados son la *estrategia de oscilación*, la *memoria de término intermedio y largo*, el proceso de *intensificación y diversificación* y el *criterio de aspiración*.

La estrategia de oscilación opera mediante el moverse hasta pegarle a una frontera representada por la factibilidad o un estado de construcción que normalmente puede representarse por un punto donde el método puede parar. En vez de parar, la definición de vecindad se extiende o el criterio de evaluación para seleccionar movimientos se modifica, para permitir que se pueda cruzar esa frontera. La aproximación entonces procede para una profundidad específica más allá de la frontera y entonces se regresa. En este punto la frontera de nuevo se aproxima y se cruza,

esta vez desde la dirección opuesta, procediendo a un nuevo punto en turno.

La memoria de término intermedio opera para registrar y comparar características de las mejores soluciones generadas durante un período particular de búsqueda. Las características que son comunes o que competen a mayoría de esas soluciones se toman como atributo regional. El método entonces procura nuevas soluciones que tengan esas características regionales.

La función de memoria de término largo, emplea principios que son inversos a los de la función de término intermedio. *La memoria de término largo* se utiliza para producir un punto inicial de búsqueda en una nueva región, mediante el penalizar las características que la memoria de término intermedio encuentra que prevalecen en la región actual de búsqueda.

La fase de *intensificación* proporciona una forma simple para enfocar la búsqueda alrededor de la mejor solución hasta el momento. La *diversificación* se restringe para operarse sólo en ocasiones particulares. En este caso, se seleccionan las ocasiones donde ningún movimiento de mejora admisible existe. Por lo general, se utiliza la información de la frecuencia para penalizar a los movimientos que no mejoran la búsqueda mediante el asignar una penalización grande a intercambio de pares con mayores contadores de frecuencia.

Los *criterios de aspiración* se introducen en la búsqueda tabú para determinar cuando las restricciones tabú pueden sobrellevarse para remover una clasificación tabú que de otra manera se aplicaría a un movimiento. El uso apropiado de tales criterios puede ser muy importante para que un método de BT alcance sus mejores niveles de realización.

Al igual que cualquier otro método heurístico, la BT genera soluciones que dependerán de la naturaleza del problema que se deba resolver, su planteamiento, así como la estructura del programa que realice el proceso iterativo de búsqueda y la capacidad del computador que se use. Su perspectiva tiende hacia su implementación en la mayoría de los problemas de programación lineal entera y combinatoria en razón de su factibilidad como un método eficiente en cuanto a resultados, uso de memoria y rapidez de procesamiento.

Un aspecto importante que debe considerarse al elaborar y aplicar los programas de Inteligencia Artificial es el dominio del área del conocimiento en que se van a presentar los problemas que habrán de resolverse. El dominio de conocimientos debe estar claramente acotado y estructurado para poder ser manejado. Por ello en Inteligencia Artificial es usual tener por separado los conocimientos y el mecanismo que controla la búsqueda de soluciones (al contrario de lo que ocurre con los programas convencionales). Como consecuencia los programas de Inteligencia Artificial son normalmente *fáciles de modificar, actualizar y ampliar*, ya que se puede cambiar la estructura de una instancia cambiando la base de conocimientos y utilizar el mismo método de búsqueda. Debido a esta organización y al hecho de que el sistema informático puede inferir nuevas reglas, en el caso de IA, el determinismo algorítmico que caracteriza a la programación convencional desaparece en buena parte.

Con los avances en los sistemas de procesamiento de información, la investigación de operaciones estará tomando nuevos rumbos. La creación de técnicas basadas en los procesos *inteligentes* que se utilizan en la estructuración de los sistemas expertos, permitirá contar con nuevas estrategias para la solución de los casos que en la actualidad se encuentran en una situación conflictiva.

En el campo de la optimización combinatoria queda aún mucho por hacer. Los programas basados en procedimientos heurísticos tienen siempre la limitación de estar restringidos a un manejo parcial de la información de un problema. Además cada problema guarda complejidades diferentes respecto con el método que se utilice para resolverlo, que no puede hacerse una generalización al respecto.

Las facilidades que brindan los nuevos equipos de cómputo y el desarrollo de los lenguajes de alto nivel permiten generar nuevas propuestas con alternativas de procesamiento y de uso de memoria cada vez más amplias.

BIBLIOGRAFIA

CAPITULO I.

Balinski, M. L. y Gomory, R. E. *A primal method for the assignment and transportation problems.* En Management Science, Vol. 10. No. 3, abril 1964.

Bazaraa, M. S., Jarvis, J. J. y Sherali, H. D. *Linear programing and network flows.* U. S. A., John Wiley & sons, segunda edición, 1990.

Bellmore, M. y Nemhauser, G. L. *The traveling salesman problem: a Survey.* En Operations Research, Vol. 16, No. 3, mayo-junio, 1968.

Bruce, Schneier. *NP - Completeness.* En Dr. Dobb's Journal, No. 9, Septiembre, 1994. pp. 119-121.

Segal, M. *The operator - Scheduling Problem: A network - flow aproach.* En Operations Research, Vol.22, No. 4, julio - agosto, 1974.

Dantzing, G. *Linear programming and extensions.* Princenton, N. J. Princenton University Press, 1963.

Nemhauser, G. y Wolsey L. *Integer and combinatorial optimization.* New York, Wiley, 1988.

Garfinkel, R. y Nemhauser. *Integer programming.* New York, Wiley, 1972.

Aho, Hopcropt & Ullman. *Estructura de datos y algoritmos.* México, Ed. Sitesa, 1988.

Christofides, N. *Combinatorial optimization.* U.S.A., John Wiley, 1979.

De la Mota Flores, Idalia. *Apuntes de programación entera.* UNAM. Facultad de Ingeniería. División de estudios de posgrado.

CAPITULO II.

Prawda Juan. *Métodos y modelos de investigación de operaciones.* Vol. I Modelos determinísticos. México, Limusa. 1992.

Winston L. Wayne. *Investigación de operaciones: Aplicaciones y algoritmos.* México, Grupo Editorial Iberoamerica. 1994.

Hillier, S. F. y Lieberman, G. J. *Introducción a la investigación de operaciones.* México, Mc Wraw Hill. 1993.

Shamblin, J. E. y Stevens, G. T. *Investigación de operaciones: Un enfoque fundamental.* México, Mc Wraw Hill. 1982.

Gass, Saúl. *Programación Linear.* México, CECSA. 1983.

Taha, H. A. *Investigación de operaciones,* Segunda edición, México, Alfaomega, 1991.

CAPITULO III.

Díaz Caballero, José Ricardo. *Heurística filosófica.* La Habana. Mimeo. 1992.

Jacobson, S. K. *Heuristics for the capacited plant location model.* European Journal of Operation Research, 1993.

Rich, Elaine. *Artificial intelligence.* Segunda edición, U.S.A. Mc Wraw Hill. 1991.

Gould, F. J., Eppen, G. D. y Schmidt, C. P. *Investigación de operaciones en la ciencia administrativa.* México, Prentice All. 1994.

Martino, P. J. *The detrimental wire exclusion heuristic: A new approach to combinatorial optimization.* Dr. Dobb's Journal. Abril, 1995.

Pearson, Peter. *Biochemical techniques take on combinatorial problems.* En Dr. Dobb's Journal. abril, 1995 pp. 121-125.

CAPITULO IV.

Richard, Morin. *Artificial intelligence: sources for information about artificial intelligence research on the Internet.* En UNIX Review. Vol. 12, No. 7, july 1994. pp. 91 - 94.

Rich, Elaine. *Artificial intelligence.* Segunda edición, U.S.A. Mc Wraw Hill. 1991.

Glober, Fred y Harvey J. Greenberg. *New approaches for heuristic search: A bilateral linkage with artificial intelligence.* En European Journal of Operational Research, No. 39, 1989. pp. 119 - 130.

Ferrer, Antonio. *Dentro y fuera del ordenador.* Ed. Ingelek, Colección biblioteca básica de informática. Chile, 1985.

Algotimos genéticos.

Atiel Ben Hadj-Alouane, James C. Bean y Katta G. Murty. *A hibrid genetic: Optimization algorithm for a task a allocation problem.* Department of industrial and operations engineering, University of Michigan. Noviembre 1993.

Pirkul, Hasan y Rolland, Erik. *New heuristic solution procedures for the uniform graph partitioning problem: Extensions and evaluation.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 895 - 907.

Conway, Daniel G. y Venkataramanam, M. A. *Genetic search and the dynamic facility layout problem.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 955 - 960.

Price U., Kenneth. *Genetic annealing: Functioning and use of the hybryd random-search technique.* En Dr. Dobb's Journal, Octubre 1994.

Enrado, Patty. *Investing in genetic algorithms.* En AI Expert. Vol. 9, No.9, Sep. 1994. pp. 48 - 56.

Hedberg, Sara. *Emerging genetic algorithms: successful applications for real world problems.* En AI Expert. Vol. 9, No.9, Sep. 1994. pp. 24 - 30.

Recocido simulado.

Johnson, D. S., Aragón, C. R. Et. Al. *Optimization by simulated annealing: An experimental evaluation; part I, Graph partitioning.* Operation Research, 1989. pp 865-892.

Gutiérrez Andráde, Miguel Angel. *La técnica de recocido simulado y sus aplicaciones.* Tesis doctoral en investigación de operaciones. México, UNAM. División de Estudios de Posgrado de la Facultad de Ingeniería. 1991.

Kirkpatricks S., Gelatt C. y Veachi M. *Optimization by simulated annealing.* En Science, No. 220, 1993. pp. 671 - 680.

Hassan Osman, Ibrahim. *Metaestrategy simulated annealing and tabu search algorithmsfor the vehicle routing problem.* En Annals of operations research, No. 41, 1993. pp. 421 - 451.

Woodruft, David L. *Simulate annealing and tabu search; Lessons from a line search.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 823 - 839.

Redes neuronales.

Varios autores. *Comunicación neural y el proceso de toma de decisiones.* En Psicología de la conducta, Tomo I. Biblioteca de Psicología, México, CECSA. 1993.

James A. Freenman y David M. Skapura. *Neural networks algorithms. Aplications and programming techniques.* Wesley Publishing Co. Massachusetts, E.U.A. 1991.

Levine, Robert I., Drang, Diane E. y Edelson, Barry. *AI and expert sistems: A comprensive guide.* Segunda edición, México, Mc Graw Hill. 1990.

Elaine Rich y Kevin Knight. *Artificial intelligence.* Segunda edición. U.S.A. Mc Graw Hill. 1991.

Burke I. Laura y James P. I. *Neural Networks and operation research an overview.* En

Computer Operations Research. Vol. 19, No. 3/4. 1992. pp. 179 - 189.

Chee-Kit Looi. *Neural network methods in combinatorial optimization.* En Computers Ops. Res. Vol. 19, No.3/4, 1992. pp. 191 - 208.

Kempka, Anthony A. *Activating neural networks: Behavior of the activation function in neural networks.* En AI Expert. Vol. 9, No. 6, Jun. 1994. pp. 33 - 38.

Murray, Dan. *Turnin neural networks with genetic algorithms.* En AI Expert. Vol. 9, No. 6, Jun. 1994. pp. 27 - 32.

Barr, Dean S. y Mani, Genesh. *Using neural nets to manage investments.* En AI Expert. Vol. 9, No. 2, Feb. 1994. pp. 16 - 22.

Búsqueda Tabú.

Glober, Fred. *A user's guide to tabu search.* En Annals of Operations Reseach. Vol. 41. 1993. pp 3 - 28.

Laguna M. y Glober F. *Integrating target analysis and tabu search for improved scheduling systems.* En Expert systems with aplication, vol. 6, 1993. pp. 287 - 297.

De los Cobos, Sergio. *La técnica de búsqueda tabú y sus aplicaciones. Tesis doctoral en investigación de operaciones.* México, UNAM. División de Estudios de Posgrado de la Facultad de Ingeniería. 1994.

Barnes, J. Wesley y Laguna, Manuel. *A tabu search experience in production scheduling.* En Annals of Operations Reseach. Vol. 41. 1993. pp 141 - 156.

Knox, John. *Tabu search performance on the symmetric traveling salesman problem.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 867 - 876.

Icmeli, Oya y Erenguc, Selcuk. *A tabu search procedure for the resourse constrainted project scheduling problem with discounted cash flows.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 840 - 846.

Skorin-Kapov, Jadranka. *Extensions of a tabu search adaptation on the quadratic assignment problem.* En Computers Ops. Res. Vol. 21, No. 8, 1994. pp. 855 - 865.

Laguna, Manuel y Jaimes P., Kelly. *Master production scheduling in a single facility with sequence-dependent changeover times.* Mimeo. Septiembre, 1993.

Semet, Frédéric y Taillard, Eric. *Solving real-life vehicle routing problems efficiently using tabu search.* En Annals of Operations Reseach. Vol. 41. 1993. pp 469 - 488.