LA TÉCNICA DE RECOCIDO SIMULADO Y SUS APLICACIONES

MIGUEL ANGEL GUTIÉRREZ ANDRADE

DIVISIÓN DE ESTUDIOS DE
POSGRADO DE LA
FACULTAD DE INGENIERÍA
de la
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

CIUDAD UNIVERSITARIA, D.F., Agosto, 1991

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

INTRODUCCIÓN

Los problemas de optimización se dividen de manera natural en dos categorías: problemas de optimización con variables *continuas* y problemas de optimización con variables *discretas*. A estos últimos se les llama *problemas de optimización combinatoria*. En problemas de tipo continuo se busca la solución sobre un conjunto de números reales con ciertas propiedades de continuidad y generalmente de convexidad; en los problemas de optimización combinatoria se busca la solución sobre un conjunto finito o infinito numerable de objetos para los que se pueda definir una función que evalúe la "calidad" de cada objeto. Estas dos clases de problemas tienen diferentes retos y los métodos para resolverlos son distintos.

Una característica recurrente en los problemas de optimización combinatoria es el hecho que son muy "fáciles" de entender y enunciar, pero generalmente son "difíciles" de resolver. Podría pensarse que la solución de un problema de optimización combinatoria se restringe únicamente a buscar de manera exhaustiva el valor máximo o mínimo en un conjunto finito de posibilidades y que usando una computadora veloz, el problema carecería de interés matemático, sin pensar por un momento, en el tamaño de este conjunto. En la mayoría de los casos, este conjunto de posibilidades crece de manera alarmante conforme crece el tamaño de la instancia del problema. Es frecuente, que los problemas de este tipo tengan $n!$ o más soluciones factibles, (donde n es el tamaño de la instancia) y si una computadora pudiera ser programada para examinar soluciones a razón de un billón de soluciones por segundo; la computadora terminaría su tarea, para $n = 23$ en alrededor de 820 años, para $n = 24$ en alrededor de 19,674 años y así sucesivamente. ¡No tiene sentido resolver de esa forma un problema si al interesado no le alcanza su vida para ver la respuesta!

Bajo las consideraciones anteriores, el interés de la optimización combinatoria pasa a ser la de desarrollar algoritmos para los cuales el número de etapas computacionales elementales sea aceptablemente pequeño. Equivalentemente, que encuentren la solución en un tiempo razonable o como suele decirse (en términos formales) en tiempo polinomial; esto es, el número de etapas computacionales elementales es una función polinomial del número de datos.

Los problemas de optimización combinatoria surgen como modelos para resolver problemas que tradicionalmente ha estudiado la Investigación de Operaciones y otros recien-

temente generados en Teoría de la Computación y la aplicación de las computadoras a problemas numéricos y no numéricos que han requerido nuevos métodos, enfoques y herramientas matemáticas para su solución. Dichos problemas se caracterizan por las propiedades combinatorias de las restricciones.

En el pasado se han hecho esfuerzos para resolver problemas de optimización combinatoria, hay problemas para los que existen algoritmos rápidos y eficientes; por ejemplo, el problema de programación lineal, flujo en redes, acoplamiento, transporte, asignación, etc. Sin embargo, hay problemas de optimización combinatoria para los que no se conocen algoritmos eficientes que los resuelvan cuando el tamaño del problema es mediano o de gran escala; por ejemplo, el problema del agente viajero, el problema lineal entero, el problema de asignación cuadrático, el problema de reubicación de equipo, el problema de localización de plantas, etc. Estos últimos problemas tienen más de 20 años de ser atacados, en la mayoría de los casos, por algoritmos desarrollados especialmente para el problema específico y usando una diversidad de técnicas tales como planos de corte, ramificación y acotamiento, enumeración implícita, programación dinámica, partición de Benders, relajación lagrangeana, etc. o combinaciones de las técnicas antes mencionadas. Sin embargo, no se ha podido dar soluciones eficientes. Por ejemplo, para el problema del agente viajero se conocen instancias menores de 100 ciudades cuya solución óptima es desconocida [GLO89] y la instancia más grande que se ha resuelto es de 318 ciudades [LAW85]. También existe la necesidad de producir métodos de aplicabilidad general y flexibles para problemas de optimización combinatoria que obtengan soluciones en un tiempo razonable.

En el estudio de la existencia de algoritmos que permitan encontrar la solución buscada en un tiempo polinomial y la construcción de ellos cuando es posible, es muy importante el conocimiento de las propiedades y estructura matemática del problema. En particular, la teoría de Gráficas permite, en muchos casos, el estudio de esta estructura y al aprovechar sus propiedades es posible construir los algoritmos buscados. Pero para otros problemas, no siempre resulta fácil aprovechar la estructura. Hay problemas lineales, como los de flujo en redes y acoplamiento que pueden resolverse de manera muy eficiente. Sin embargo estos problemas están aparentemente muy relacionados con otros problemas que se consideran "intratables". Como ejemplo, los problemas de trayectoria más corta en una gráfica y de acoplamiento, en donde se conocen algoritmos $O(n^2)$ que los resuelven; en contraste con el problema del agente viajero, el cual resuelve la trayectoria más corta para visitar el conjunto de todos los nodos de una red exactamente una vez, que como es bien conocido, es un problema en la denominada clase NP-completa cuyos problemas son ampliamente considerados irresolubles por algoritmos polinomiales. Este fino límite

entre problemas "fáciles" y "difíciles" es un fenómeno recurrente en los problemas de optimización combinatoria. Otra manera de ver los problemas de flujo en redes y de acoplamiento es como casos especiales del problema lineal entero general. Sin embargo el problema lineal entero también es NP-completo.

Para los problemas de optimización en la clase NP-completa (Para una lista actualizada hasta 1980 vea [GAR79]), a la fecha, no se conocen algoritmos que los resuelvan en tiempo polinomial. Es por esto, que en las últimas décadas se han desarrollado algoritmos de tipo heurístico para resolver instancias grandes de problemas que pertenecen a esta clase. Estos algoritmos, no necesariamente encuentran la mejor solución al problema, pero en general se pueden obtener "buenas" soluciones cuando se aplican, aquí se entiende "buena" solución en términos de cercanía al valor óptimo. La ventaja principal de este tipo de algoritmos es que son rápidos y corren en un tiempo polinomial.

El propósito de este trabajo es presentar la línea de investigación seguida en el programa doctoral; así como también el de presentar un algoritmo general para la solución de problemas de optimización combinatoria denominado *recocido simulado*. Este algoritmo tiene la propiedad que, en el límite, converge al conjunto de soluciones óptimas del problema de optimización combinatoria y en un tiempo polinomial, produce buenas soluciones. En este documento se presenta el estado del arte alrededor de la técnica de recocido simulado y se dan aplicaciones concretas que muestran la bondad e importancia de la técnica. Las aportaciones realizadas mediante este trabajo son las siguientes:

- Desarrollo de programas de enfriamiento que conducen a la ejecución del algoritmo en tiempo polinomial.
- Desarrollo de estructuras de vecindades adecuadas para problemas clásicos de optimización combiantoria como son: el problema de conjunto independiente, el problema de asignación cuadrático y el problema de localización de plantas, cuyas matrices de transición resulten irreducibles y aperiódicas.
- Se da solución a un problema real de localización de plantas de gran escala por medio del algoritmo de recocido simulado.
- Se valida el algoritmo de recocido simulado y se presenta experiencia computacional para tres familias de problemas, el problema de conjunto independiente, el problema de asignación cuadrático y el problema de localización de plantas haciendo un análisis comparativo con otros algoritmos hechos expresamente para cada uno de los problemas anteriores.

- **Implantar en la computadora para los problemas de conjunto independiente, asignación cuadrático y localización de plantas, códigos eficientes que los resuelvan.**
- **Popularizar la técnica de recocido simulado.**

La técnica de recocido simulado es una técnica nueva para resolver problemas de optimización combinatoria. Fué propuesta por Kirkpatrick, Gellat y Vecchi [KIR83] y originalmente se basó en una analogía entre la simulación de recocido de sólidos y el reto de resolver problemas de optimización combinatoria de gran escala.

Recocido es el proceso físico de calentar un sólido hasta la temperatura necesaria, mantenerlo caliente durante un cierto tiempo y dejarlo enfriar lentamente hasta un estado de cristalización. Durante este proceso la energía del sólido se minimiza. En la práctica se observa que si el enfriamiento no se hace cuidadosamente, el proceso puede caer en un óptimo local donde la cristalización es imperfecta.

En optimización combinatoria, se puede definir un proceso similar. Este proceso puede formularse como el problema de encontrar, en un conjunto potencialmente muy grande de soluciones, una solución con costo mínimo. Se puede establecer una correspondencia entre la función de costo y la energía libre del sólido y entre las soluciones y los estados físicos.

La característica sobresaliente de la técnica es su aplicación general y la habilidad para obtener soluciones arbitrariamente cercanas a la óptima. Sin embargo, el obtener soluciones de alta calidad puede requerir de mucho esfuerzo computacional.

Una comparación importante que puede hacerse con respecto a si una técnica es general o no es la que se presenta con respecto a la programación lineal y la programación dinámica. La programación lineal es un modelo muy específico para solucionar problemas en donde la función objetivo y las restricciones deben ser lineales; si el problema no cae dentro de este esquema restringido, la programación lineal no puede aplicarse. Por otro lado, la programación dinámica se puede usar para resolver muchos problemas de optimización y 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. En este sentido, recocido simulado resulta ser análogo a la programación dinámica. El uso y la calidad de sus resultados dependen del arte y la habilidad con que se definan sus diferentes componentes como son: las evaluaciones de la función objetivo de la nueva solución con respecto a la anterior, la estructura de vecindades y el programa de enfriamiento; así como también, el grado de refinamiento de la implantación en la computadora.

En este trabajo se presenta la teoría del recocido simulado y se hace referencia a la analogía entre la simulación del proceso físico de recocido y el proceso combinatorio de encontrar el mínimo en un conjunto finito; se da un análisis detallado de los aspectos teóricos de la técnica, basado en conceptos de la teoría de cadenas de Markov, para mostrar convergencia en probabilidad y en distribución del proceso, al conjunto de soluciones óptimas. Se desarrollan con detalle la técnica de recocido para tres familias de problemas, para cada familia de problemas se compara el algoritmo con otros tres algoritmos; un algoritmo exacto, un heurístico y búsqueda local y se reporta la experiencia computacional por familia y por algoritmo. Se especifican brevemente el potencial de aplicaciones para resolver problemas en áreas tales como la ingeniería, computación, física y biología. Además se desarrolla a detalle una aplicación del algoritmo de recocido simulado al problema de localización de plantas.

Este trabajo se desarrolla de la siguiente manera. El Capítulo 1 comienza con la formulación general del problema de optimización combinatoria; también trata acerca de la complejidad computacional de algoritmos, en particular, se discute cuando un algoritmo se considera bueno y cuando no; además se da una discusión acerca de los algoritmos de búsqueda local basado en el concepto de estructura de vecindades. En el Capítulo 2 se usa la analogía con el proceso físico de recocido para introducir el algoritmo de recocido simulado; también se introducen algunos conceptos estadísticos, como el de equilibrio térmico, importantes para analizar el desempeño de la técnica. El Capítulo 3 da los lineamientos generales a seguir para la implementación práctica del algoritmo de recocido simulado; se desarrollan tres familias de problemas de optimización combinatoria: el problema de conjunto independiente, el problema de asignación cuadrático y el problema de localización de plantas, para cada familia, se presentan tres algoritmos adicionales que resuelven el problema; un algoritmo exacto, un algoritmo heurístico y un algoritmo de búsqueda local, se presenta experiencia computacional y se hace un análisis comparativo de los cuatro algoritmos por familia. Se desarrolla un ejemplo de aplicación de gran escala al problema de localización de plantas. Finalmente se hace referencia a algunas aplicaciones del algoritmo en diferentes campos. En el Capítulo 4 se analiza la convergencia asintótica de la técnica de recocido simulado. Este análisis se basa en la teoría de cadenas de Markov homogéneas y no homogéneas y se prueba de manera rigurosa que el algoritmo converge al conjunto de soluciones globalmente óptimas. El análisis incluye también una discusión de la velocidad de convergencia. En el Capítulo 5 se proponen programas de enfriamiento que conducen a la ejecución del algoritmo en tiempo polinomial.

CAPÍTULO 1

PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

El enfoque clásico para estudiar un problema de optimización es proceder a identificar aquellas propiedades, cualitativas o cuantitativas, que conduzcan a uno o varios procedimientos eficientes para implementarlos en una computadora y obtener su solución. Resulta importante aquí evaluar el tiempo que tardará un procedimiento para encontrar la solución ya que no es lo mismo esperar unos cuantos segundos que tener que esperar horas, días o quizá más tiempo para saber la solución del problema. Otra aspecto importante es conocer el comportamiento del algoritmo cuando el tamaño del problema crece, pues se puede tener un procedimiento que resulte adecuado para resolver problemas pequeños o medianos, pero resultar impracticable cuando el tamaño del problema es grande.

En el presente capítulo se trata de manera genérica los problemas de optimización combinatoria y el esfuerzo computacional que se requiere para conocer su solución; así como, discutir cuando es mejor un algoritmo o procedimiento de solución. El propósito del capítulo se centra en la clasificación de los problemas y sus mecanismos de solución, además de definir conceptos importantes que se usarán a lo largo de todo el trabajo.

El capítulo se desarrolla como sigue: en la Sección 1.1 se definen de manera precisa algunos conceptos de optimización combinatoria; la Sección 1.2 trata acerca de algoritmos y su complejidad computacional; y en la Sección 1.3 se trata brevemente un método heurístico para la solución del problema de optimización combinatoria denominado, búsqueda local que es el antecedente del algoritmo de recocido simulado propósito de este trabajo.

1.1 DESCRIPCIÓN DEL PROBLEMA COMBINATORIO

Una *instancia* de un problema de optimización combinatoria puede formalizarse como una pareja (S, f), donde S denota el conjunto finito de todas las soluciones posibles y f la *función de costo*, mapeo definido por

$$f : S \rightarrow R \tag{1}$$

En el caso de minimización, el problema es encontrar $i_{opt} \in S$ que satisfaga

$$f(i_{opt}) \leq f(i) \quad \forall i \in S \tag{2}$$

en el caso de maximización, la i_{opt} que satisfaga

$$f(i_{opt}) \geq f(i) \quad \forall i \in S \tag{3}$$

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

Un problema de *optimización combinatoria* es un conjunto I de instancias de un problema de optimización combinatoria.

En las definiciones anteriores se ha distinguido entre un *problema* y una *instancia* del problema. De manera informal, una instancia está dada 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.

Ejemplo 1 *Sea X un conjunto de n enteros positivos $x_1, x_2, ..., x_n$ distintos. Para n número par, particione el conjunto X en dos subconjuntos de cardinalidad $n/2$ tal que la diferencia entre las sumas de los elementos de los dos subconjuntos sea maximizada. En este caso, el conjunto S está dado por:*

$$S = \{C| \text{ tal que } C \text{ es un subconjunto de } X \text{ y } |C| = n/2\}$$

y la función f queda definida como:

$$f = \sum_{x_k \in C} x_k - \sum_{x_k \in X-C} x_k$$

Cuando se establece el problema de particionar los enteros $x_1, x_2, ..., x_n$; se piensa en todos los valores posibles que puedan tomar los enteros. Si se define el valor de n y se dan valores específicos a $x_1, x_2, ..., x_n$, tenemos una instancia del problema.

Ejemplo 2 *Sea X un conjunto de n enteros positivos distintos $x_1, x_2, ..., x_n$. Particione el conjunto X en dos subconjuntos tales que la diferencia entre las sumas de los elementos de los dos subconjuntos sea minimizada. En este caso, el conjunto S está dado por:*

$$S = \{ C| \text{ tal que } C \text{ es un subconjunto de } X \}$$

y la función f queda definida como:

$$f = |\sum_{x_k \in C} x_k - \sum_{x_k \in X-C} x_k|$$

Como en el ejemplo anterior, una instancia del problema consiste en definir el valor de n y cada uno de los valores x_k, para $k = 1, 2, ..., n$.

Ejemplo 3 *(El Problema del Agente Viajero) Considere n ciudades y una matriz* (d_{pq}) *de orden* $n \times n$*, cuyos elementos denotan la distancia entre cada par* p, q *de ciudades. Se define un* recorrido *como una trayectoria cerrada que visita cada ciudad exactamente una vez. El problema es encontrar el recorrido con longitud mínima.*

En este problema, una solución está dada por una permutación cíclica $\pi = (\pi(1), \pi(2), ..., \pi(n))$*, donde* $\pi(k)$ *denota la ciudad a visitar después de la ciudad* k*, con* $\pi^l(k) \neq k$*,* $l = 1, 2, ..., n-1$ *y* $\pi^n = k$*, para toda* k*. Aquí* $\pi^l(k)$ *se entiende por la aplicación de* l *veces la permutación* π*. Cada solución corresponde a un recorrido. El espacio de soluciones está dado por*

$$S = \{ \text{ todas las permutaciones } \pi \text{ cíclicas de las } n \text{ ciudades} \}$$

y la función de costo se define por

$$f(\pi) = \sum_{i=1}^{n} d_{i,\pi(i)}$$

es decir, $f(\pi)$ *da la longitud del recorrido correspondiente a* π*. Además, se tiene que* $|S| = (n-1)!$.

En general, un *algoritmo* es un procedimiento paso a paso que resuelve un problema. Se dice que un algoritmo *resuelve* un problema, si puede aplicarse a cualquier instancia y siempre garantiza una solución.

Un aspecto importante, en los algoritmos, es el tiempo que tardan en resolver un problema, pues generalmente el tiempo de ejecución del algoritmo es una función del tamaño de la instancia del problema a resolver. Bajo esta consideración, conviene definir de manera precisa la forma en que se considera a la función que mide el tiempo o "esfuerzo computacional" de acuerdo al tamaño de la instancia. Específicamente, se dice que una función real $f(n)$ es $O(g(n))$ (Se lee "es del orden de $g(n)$") siempre que exista una constante positiva m tal que

$$|f(n)| \leq m|g(n)| \tag{4}$$

para todos los valores de $n \geq 1$. Una función $f(n)$ es del orden de un polinomio o polinomial si $|f(n)| \leq m|p(n)|$ donde $p(n)$ es un polinimio en n.

Para describir un problema, en términos de una computadora, se necesita codificar los datos de entrada en una cadena de caracteres binarios (o de otro tipo) y si se supone que esta cadena es de longitud n, entonces la cantidad de tiempo que necesita un algoritmo para su ejecución es una función de n, denominada $f(n)$. Lo que interesa es la forma y comportamiento asintótico de $f(n)$.

En teoría, se debe codificar los datos de entrada y medir la longitud de la cadena codificada. Por ejemplo, en problemas de teoría de gráficas, se puede usar el número de nodos y/o de arcos como una medida del tamaño del problema, ya que son polinomialmente equivalentes al tamaño de la longitud de código, esto es, el número de caracteres codificados que se requieren para describir la gráfica es una función polinomial del número de nodos y arcos.

Es usual que existan varias maneras de codificar los datos de entrada para describir un mismo problema. Por ejemplo, para gráficas existen varias estructuras de datos que las representan, sin embargo, todas son polinomialmente equivalentes. Si la entrada de datos codificada tiene longitud n y el algoritmo requiere un tiempo $f(n)$ para resolverlo, se desea conocer si $f(n)$ es una función polinomial o exponencial. (Una función se considera exponencial si n aparece en el exponente, así $n^{\ln n}$ se considera como función exponencial). En términos generales, la clasificación de $f(n)$ en función polinomial o en función exponencial no depende de la codificación; es decir, la función $f(n)$ nunca puede ser una función polinomial para una codificación razonable y una función exponencial para otra codificación razonable. En este sentido se puede decir que todas las codificaciones razonables son equivalentes.

Una codificación de datos se dice que es "razonable" si la capacidad de memoria necesaria para guardar el entero positivo n es $\lceil log(n+1) \rceil$ donde $\lceil x \rceil$ es el mínimo entero superior o igual a x. Cualquiera que sea la base escogida para el algoritmo (habitualmente se toma la base 2 y a veces base 10). Es claro que la codificación unitaria (que requiere de mil lugares de memoria para escribir el número $n = 1000$) jamás se utilizará. Una manera de comprender la dimensión de un problema cuando se utiliza una codificación razonable, se refiere al empleo de una calculadora; la dimensión de los datos se mide por el número de presiones ejercidas sobre las teclas dadas por el operardor.

Otro problema importante en el tiempo de ejecución de un algoritmo, es pensar que para algún tipo de computadora el tiempo de ejecución de un algoritmo sea polinomial mientras que para otro tipo de computadora el tiempo de ejecución, del mismo algoritmo, resulte exponencial. Este problema se salva refiriendo la cantidad de tiempo o el número de pasos elementales que deberán realizarse para la ejecución de un algoritmo, a la más primitiva de las computadoras, esta es una máquina abstracta denominada Máquina de Turing. Lo interesante en esta operación es que, se puede mostrar [GAR79] que si un algoritmo necesita de tiempo exponencial para su ejecución en una Máquina de Turing, entonces también necesita de tiempo exponencial en cualquier computadora real y viceversa. En otras palabras, el tiempo polinomial o exponencial es independiente de la computadora seleccionada.

Ejemplo 4 *Un algoritmo para resolver el problema del Ejemplo 1 es el siguiente: se ordenan los n enteros de menor a mayor, se toman los n/2 enteros menores en un subconjunto y los n/2 enteros mayores en otro subconjunto. Claramente, este procedimiento puede efectuarse para cada instancia y da una solución al problema.*

Conviene mencionar que se conoce que el número de operaciones necesarias para ordenar n números reales de mayor a menor es proporcional a $n \ln n$ *(Método Quicksort), es decir, este procedimiento puede hacerse en un tiempo* $O(n \ln n)$. *Un camino más rápido es encontrar la mediana de los n enteros y usar este valor para particionar los dos subconjuntos. El algoritmo para encontrar el k-ésimo entero más grande entre los n enteros es* $O(n)$. *Puesto que cualquier algoritmo debe checar cada entero,* $O(n)$ *es la cota inferior. De hecho puede resolverse este problema para dos subconjuntos de tamaño arbitrario, encontrando el k-ésimo entero más grande y entonces particionando el conjunto en dos subconjuntos, uno de tamaño* k *y otro de tamaño* $n - k$.

Ejemplo 5 *Para resolver el problema del Ejemplo 2 considere el siguiente algoritmo: particione el conjunto en un subconjunto de tamaño* k *y otro subconjunto de tamaño* $n - k$ *y observe sus sumas. Si se tratan todas las posibles combinaciones de k, entonces se puede encontrar la mínima diferencia. Claramente el procedimiento anterior se puede efectuar para cada instancia del problema y da la solución.*

Para ejecutar este algoritmo, se necesita buscar exhaustivamente sobre todos los posibles subconjuntos cuyo número total es 2^n, *es decir, este procedimiento puede hacerse en un tiempo de orden* $O(2^n)$.

1.2 ALGORITMOS Y SU COMPLEJIDAD

Se ha establecido que un algoritmo es un procedimiento paso a paso para resolver un problema y resulta natural tratar de investigar si un algoritmo es eficiente o no. Sin embargo, esto arroja varias preguntas tales como:
¿Los problemas tienen distintos grados de dificultad? ¿Pueden los problemas clasificarse por su grado de dificultad? ¿Son algunos problemas intrínsecamente más difíciles que otros? ¿Existen problemas que no tienen algoritmos eficientes que los resuelvan? ¿Qué relación existe entre problemas y algoritmos? ¿Existe alguna medida de eficiencia en los algoritmos? ¿Cómo se pueden clasificar los algoritmos actuales? ¿Qúe preguntas acerca de los problemas interesa resolver? En esta sección se abordarán las preguntas anteriores y se tratará de dar una contestación a cada una de ellas.

Se empezará por decir que existen al menos tres motivos para hacer una clasificación de los problemas. Estos motivos son:

- Si se conoce que un problema está contenido en una clase "difícil", no debemos concentrarnos en encontrar su solución óptima.
- Cuando dos problemas están en una clase (entendiendo esto, por el mismo nivel de dificultad), la técnica para resolver uno, algunas veces puede usarse para resolver el otro (empleando alguna transformación).
- Existen problemas que parecen muy similares, y sin embargo uno es mucho más difícil que otro.

Ejemplo 6 *Dos problemas aparentemente similares son los expuestos en los Ejemplos 1 y 2. Aquí se puede hacer la pregunta: ¿Cuál de estos dos problemas es más difícil de resolver? El primer problema, puede resolverse en forma muy rápida por el algoritmo expuesto anteriormente (Ejemplo 4), sin embargo el algoritmo dado para resolver el Ejemplo 2 (Ejemplo 5) no es eficiente, ya que el tiempo de solución crece de manera exponencial cuando crece el tamaño del problema; existen mejores algoritmos para resolver este problema, pero a la fecha no se ha encontrado un algoritmo en donde n no aparezca en el exponente. Esto hace que el problema del Ejemplo 2 sea más difícil que el problema de Ejemplo 1.*

Quizá para una instancia dada, se tenga un algoritmo muy eficiente, pero lo que interesa es el comportamiento del algoritmo para el peor conjunto de datos posible, es decir, la instancia más mal comportada. Otro punto interesante es pensar en instancias grandes del problema. Esto motiva la clasificación de los problemas 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 (ni existirá) un algoritmo polinomial para resolverlo.

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

- Problemas Polinomiales. Se dice que un problema pertenece a la clase P de problemas resolubles polinomialmente si 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 que no se conozca un algoritmo polinomial que lo resuelva, aunque tal algoritmo exista.

Una pregunta muy importante de resolver es si $P = NP$. Esta pregunta tiene más de 20 años de formulada y a la fecha no se ha podido responder; ha sido tema de investigación de mucha gente brillante en el área de optimización y de teoría de la computación a través de los últimos años. Para entender lo que esto significa, a continuación se establece con más detalle lo que es un algoritmo polinomial, un algoritmo no determinista y finalmente lo que es la clase NP-completa, que es la subclase "difícil" dentro de NP.

Algoritmos Polinomiales

Se define un algoritmo de *tiempo polinomial* como aquel cuya función de complejidad de tiempo es $O(p(n))$ para alguna función polinomial p, donde n denota el tamaño de la instancia. Cualquier algoritmo cuya función de complejidad de tiempo no pueda ser acotado por un polinomio se dice *algoritmo de tiempo exponencial.* Un problema que requiera $n \ln n$ unidades de tiempo está contenido en la clase P, así como un problema que requiera $(n^{117} + 3n^4 + 7739)$ unidades de tiempo. Aquí, la unidad de tiempo no se especifica y puede ser microsegundo u hora. En este último caso, únicamente esta involucrada la constante $3,600,000,000$. Existen tres razones para concentrarse en la clase P. Primera, la adición o multiplicación de dos polinomios, es un polinomio; los polinomios son invariantes bajo sustitución. Generalmente es fácil contar el número de operaciones que requiere el algoritmo para saber si la cantidad de pasos que necesita es polinomial. Segundo, la función polinomial es insensible a la implementación en computadora. Específicamente la implementación en computadora de un algoritmo, por diferente que sea la estructura de datos usada, a lo más puede bajar (o subir) el grado del polinomio, pero no existen ejemplos en que dos implementaciones del mismo algoritmo requieran, una un tiempo polinomial y la otra un tiempo exponencial. En este sentido, la propiedad de que el esfuerzo del algoritmo sea de tiempo polinomial viene a ser una propiedad intrínseca

del algoritmo. Y la tercera, la más importante para los fines perseguidos en este trabajo, es obtener algoritmos polinomiales pues los algoritmos polinomiales son mejores que los exponenciales cuando el problema es "grande" como es el caso típico de aplicaciones reales.

Con el propósito de ponderar el crecimiento asintótico de los algoritmos, considere que un algoritmo A necesita n^5 operaciones y un algoritmo B requiere de 2^n operaciones para resolver un problema. Si cada operación toma un microsegundo, entonces el algoritmo A necesita 0.1 segundos para $n = 10$ y 13 minutos para $n = 60$. Por otro lado, el algoritmo B necesita 0.001 segundos para $n = 10$ y 36,600 años para $n = 60$. Por esta razón los algoritmos polinomiales se consideran "buenos" algoritmos. Si un algoritmo exponencial es mejor que un algoritmo polinomial para una cierta n, sabemos que el algoritmo polinomial siempre superará en velocidad al exponencial para instancias mayores o iguales a un cierto valor n. Note sin embargo, que la comparación es únicamente para el peor de los casos, puesto que existen algoritmos exponenciales que son más rápidos que los algoritmos polinomiales en el caso promedio.

Finalmente, se introduce lo que se entiende por problema de decisión. Los *problemas de decisión* son problemas para los que su solución es una respuesta "si" o "no". Por ejemplo, el problema del agente viajero se puede convertir a un problema de este tipo haciéndose la pregunta ¿La gráfica tiene un recorrido con distancia total menor que B? Si se puede responder a esta pregunta en tiempo polinomial, entonces se puede hacer esta pregunta para un conjunto de valores distintos de B (usando una búsqueda binaria) y encontrar la solución al problema de optimización en un tiempo polinomial.

Para un problema de decisión (como se definió anteriormente) y un algoritmo que lo resuelva, dando una entrada de datos, la computadora efectuará varios cálculos hasta que finalmente dará una respuesta. Para problemas de decisión, la respuesta será "si" o "no". Uno se puede imaginar el proceso seguido por el algoritmo como un árbol enraizado, en donde cada nodo representa algún cálculo, las ramificaciones representan las posibles alternativas que pueda seguir el proceso dependiendo de la instancia a resolver y los nodos terminales del árbol están etiquetados con "si" o "no". Para un algoritmo que corra en tiempo polinomial, la profundidad de este árbol para cualquier valor del parámetro n debe ser polinomial en n; además, la cantidad de tiempo necesario para alcanzar cada nodo también debe ser polinomial.

Cuando se dan los datos de entrada, el programa de computadora, debe hacer algunos cálculos, el resultado de estos cálculos decidirá cual trayectoria se seguirá dentro del árbol. Un algoritmo polinomial tiene la propiedad que los cálculos a través de todas estas trayectorias son polinomiales en tiempo.

Pedir a un algoritmo que todas sus trayectorias estén acotadas por polinomios es

muy restrictiva y es de gran importancia seleccionar una clase especial del problema y resolverla con un algoritmo eficiente, donde la clase general solamente tiene algoritmos exponenciales.

Algoritmos No Deterministas

En esta parte discutimos los problemas contenidos en la clase NP, donde NP se entiende por no determinista polinomial. Para discutir el concepto de algoritmo no determinista, tomemos nuevamente el problema del agente viajero, un algoritmo determinista especificará por cual arco se viajará a partir de un nodo. En el caso de la pregunta ¿Existe un recorrido con distancia total menor que B? Un algoritmo exhaustivo buscará sobre todas los posibles recorridos y responderá "si" si existe un recorrido con esta característica y "no" en otro caso. Puesto que los recorridos serán evaluados uno por uno y el último puede ser el recorrido con la propiedad deseada, el algoritmo exhaustivo para resolver el problema del agente viajero se considera que es $O((n-1)!)$.

Un algoritmo no determinísta, puede "adivinar" correctamente cual arco deberá usarse enseguida de cada nodo en el recorrido. En otras palabras si existe un recorrido con distancia total menor que B, el algoritmo no determinista tomará $O(n)$ para calcular la distancia total del recorrido y verificar que tal recorrido existe, si la respuesta es "si".

Se dice que el problema del agente viajero se resuelve por un algoritmo no determinista en tiempo polinomial si la respuesta es "si". Equivalentemente, si la respuesta es "si", entonces se considerará que el problema puede ser resuelto por un algoritmo no determinista en tiempo polinomial o que el problema pertenece a la clase NP. Por otro lado, se puede observar que hay una importante ausencia de simetría entre la respuesta "si" y "no". Uno puede ser incapaz, para un problema en la clase NP de probar en tiempo polinomial que la respuesta es "no". Por ejemplo, no se conoce algoritmo no determinista polinomial que permita verificar que el problema de decisión asociado al problema del agente viajero no tiene solución. Es decir, no se sabe como mostrar que el siguiente problema, llamado el problema complemento, pertenezca a la clase NP. ¿Es cierto que en la gráfica G no existen recorridos de costo inferior o igual a B? Esta asimetría se puede comprender bien si se regresa al principio de verificar cuando la respuesta es "si" en el problema original. Resulta fácil convéncerse que la respuesta es "si", únicamente exhibiendo un recorrido con longitud menor o igual a B. Para el otro problema, no se conoce un método eficaz para mostrar la inexistencia de tales recorridos, una forma es exhibir todos los posibles recorridos y verificar que ninguno tiene una longitud menor o igual a B.

. Por lo tanto, si no existe un recorrido con distancia total menor que B, es decir la respuesta es "no" vemos que se requiere más que la habilidad de adivinar correctamente. Por lo tanto, el problema de respuesta "no" probablemente no pertenezca a NP.

La ambiguedad anterior no existe al tratar los problemas en la clase P. Cuando el problema pertenece a P, su complemento también pertenece a P. Se conjetura que esto no es verdadero para problemas en NP (aunque la pregunta técnicamente está abierta). Es inmediato que si un problema esta contenido en la clase P, entonces también está contenido en la NP. De modo que la clase P es una subclase de NP. Una pregunta abierta en la Teoría de la Computación es

$$¿P = NP? \tag{5}$$

En otras palabras, ¿Es la inclusión propia?
Observe que para probar que $P = NP$, se debe probar que todos los problemas en NP pueden resolverse en tiempo polinomial por algoritmos deterministas. O probar que un problema en NP no puede ser resuelto en forma determinista en tiempo polinomial para mostrar $P \neq NP$.

Problemas NP-Completo

Considere el problema de circuito Hamiltoniano consistente en determinar en una gráfica si existe una recorrido que visite cada nodo exactamente una vez. Ahora bien, el problema del agente viajero pregunta si existe una ruta suficientemente corta que visita todo nodo de la gráfica. Un algoritmo para el problema del agente viajero puede usarse para resolver el problema de circuito Hamiltoniano, puesto que este último se puede transformar a un problema del agente viajero como sigue: Asignemos a cada arco de la gráfica una distancia igual a 1, y la distancia entre dos nodos igual a ∞ si no existe un arco que los conecte. Puesto que hay n nodos en la gráfica, la existencia de una ruta de distancia total igual a n significa la existencia de un circuito Hamiltoniano.

Note que la transformación anterior toma tiempo polinomial; es decir, específicamente la distancia para los $\frac{n(n-1)}{2}$ pares. Si uno descubre un algoritmo polinomial para el problema del agente viajero, habrá descubierto también un algoritmo polinomial para el problema de circuito Hamiltoniano.

Se dice que el problema L_1 se reduce al problema L_2 y se escribe

$$L_1 \propto L_2$$

si cualquier algoritmo para L_2 puede usarse para L_1. Aquí se entiende que esta transformación toma tiempo polinomial. Si L_2 puede ser reducido a L_1, entonces se dice que L_1

y L_2 son *polinomialmente equivalentes*; es decir,

$$L_1 \simeq L_2$$

Las relaciones $\propto$ y $\simeq$ son transitivas. Por ejemplo, si $L_1 \propto L_2$ y $L_2 \propto L_3$ entonces $L_1 \propto L_3$. Si $L_1 \propto L_2$, entonces el problema L_2 es al menos tan difícil como L_1.

Existe un subconjunto de problemas en NP que son los problemas más difíciles en el siguiente sentido (los problemas en este subconjunto son polinomialmente equivalentes). Cualquier problema en NP puede reducirse a cualquier problema en el subconjunto difícil. De modo que si tenemos un algoritmo polinomial para un problema en el subconjunto, podemos resolver todos los problemas en NP en tiempo polinomial. Los problemas en este subconjunto se llaman problemas *NP-completos*. En resumen, en la clase NP existen dos subconjuntos de problemas, el subconjunto P para el que existen algoritmos deterministas polinomiales y el subconjunto NP-completo que contiene los problemas más difíciles dentro de NP.

El primer problema NP-completo fue descubierto en un artículo de Cook [COO71], y Karp [KAR75] muestra que hay muchos otros problemas que son también NP-completos. Para probar que $P = NP$, se debe probar que un problema en la clase NP-completo puede resolverse en tiempo polinomial de manera determinista. Para probar que $P \neq NP$, debemos mostrar que un problema en NP no está en P. También se puede mostrar que si $P \neq NP$, entonces existe un problema que no es ni P ni NP-completo. (Vea [LAD77]).

Existe una extensa lista de problemas de optimización combinatoria que están contenidos en la clase NP-completa (vea [GAR79]) para los que 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 de manera alarmante cuando n crece.

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

- Método Simplex.
- Planos de Corte.
- Ramificación y Acotamiento.
- Enumeración Implícita.

- Programación Dinámica.
- Partición de Benders.
- Técnicas de Grupos.

La implantación en código de 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 resolver 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 denominados *heurísticos* para obtener "buenas soluciones" de los problemas de optimización combinatoria. Dentro de las técnicas que existen para resolver problemas de optimización combinatoria de manera heurística se pueden enumerar las siguientes:

- Ramificación y Acotamiento.
- Enumeración Implícita.
- Relajación Lagrangeana.
- Métodos Glotones.
- Búsqueda Tabú.
- Búsqueda Local.
- Recocido Simulado.

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 obvia para los algoritmos de la primera categoría es su limitada aplicabilidad debido a la dependencia del problema; así que es deseable tener algoritmos heurísticos generales que obtengan soluciones cercanas a la óptima para la mayoría de problemas de optimización combinatoria.

Una técnica que a últimas fechas ha acaparado la atención es la de recocido simulado, por tener entre sus cualidades las siguientes: es una técnica general, esto es, es aplicable a

todos los problemas de optimización combinatoria; es robusta, esto es, se puede demostrar convergencia al óptimo sin depender de la solución inicial seleccionada; corre en tiempo polinomial y en la práctica ha dado muy buenos resultados.

1.3 ALGORITMO DE BÚSQUEDA LOCAL

El algoritmo de búsqueda local constituye una clase interesante de los algoritmos heurísticos y está basado en el mejoramiento paso a paso de la función de costo al explorar las vecindades de soluciones cercanas. Este algoritmo tiene una fuerte relación con el algoritmo de recocido simulado. El uso del algoritmo de búsqueda local presupone la definición de una solución, función de costo y una estructura de vecindades.

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

$$\mathcal{N} : S \to 2^S, \tag{6}$$

que define para cada solución $i \in S$ *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* $j \in S_i \Leftrightarrow i \in S_j$.

Ejemplo 7 *En el problema del agente viajero (vea Ejemplo 3) una estructura de vecindades* $\mathcal{N}_k$, *llamada* $k-$*cambio, define para cada solución* i *una vecindad* S_i *que consiste del conjunto de soluciones que pueden obtenerse a partir de la solución* i *cambiando* k *arcos del recorrido correspondiente a la solución* i *y remplazándolos por otros* k *arcos de tal manera que nuevamente se obtenga un recorrido.*

Si $k = 2$, *entonces* $\mathcal{N}_2$ *es la estructura de vecindades de 2-cambio. El 2-cambio* $\mathcal{N}_2(p, q)$ *puede verse como invertir el orden en el que las ciudades* p *y* q *se visitan en el recorrido. Si el mecanismo 2-cambio* $\mathcal{N}_2(p, q)$ *cambia una solución* i *en una solución* j, *entonces las permutaciones cíclicas* π_i *y* π_j *están relacionadas de la siguiente manera. Sea* k *denote el entero tal que* $\pi_i^k(p) = q$, $1 < k < n$, *entonces se tiene*

$$\begin{aligned} \pi_j(p) &= \pi_i^{-1}(q) = \pi_i^{k-1}(p), \\ \pi_j(\pi_i(p)) &= q = \pi_i^k(p), \\ \pi_j(\pi_i^r(p)) &= \pi_i^{r-1}(p), \quad r = 2, \ldots, k-1, \\ \pi_j(s) &= \pi_i(s), \quad \textit{en otro caso.} \end{aligned} \tag{7}$$

Así,

$$\mathcal{S}_i = \{j \in \mathcal{S} | \, \pi_j \textit{ se obtine de } \pi_i \textit{ por } 2-\textit{cambio}\}, \tag{8}$$

y

$$|\mathcal{S}_i| \equiv \Theta = (n-1)(n-2), \quad \textit{para toda } i. \tag{9}$$

Además, cada solución j puede obtenerse a partir de cualquier solución i por $n-2$ 2-cambio, es decir, para toda $i, j \in \mathcal{S}$ existe una sucesión $l_0, l_1, ..., l_{n-2} \in \mathcal{S}$ tal que

$$\pi_{l_0} = \pi_i \;\; y \;\; \pi_{l_{n-2}} = \pi_j, \tag{10}$$

y $\pi_{l_{k+1}}$ se obtiene desde π_{l_k}, $k = 0, ..., n-3$, por un $\mathcal{N}_2(p,q)$ cambio, donde

$$p = \pi_j^k(1) \;\; y \;\; q = \pi_{l_k}(\pi_j^{k+1}(1)). \tag{11}$$

Definición 2 *Sea $(\mathcal{S}, f)$ una instancia de un problema de optimización combinatoria y $\mathcal{N}$ una estructura de vecindades. Entonces un* mecanismo de generación *es un medio para seleccionar una solución j de la vecindad $\mathcal{S}_i$ de la solución i*

Dada una instancia de un problema de optimización combinatoria y una estructura de vecindades. El algoritmo de búsqueda local es un algoritmo que itera sobre un número de soluciones. Comienza con una solución inicial, que a menudo se genera aleatoriamente, después se aplica un mecanismo de generación que continuamente trata de encontrar una mejor solución en la vecindad de la solución actual, una solucion con menor costo. Si se encuentra una solución con estas características, la solución actual se reemplaza por esta solución. De otra manera el algoritmo continua con la solución actual. El algoritmo termina cuando el mecanismo de generación no puede encontrar una mejor solución a partir de la solución actual. La Figura 1 muestra el algoritmo de búsqueda local en pseudo-código.

Comienza
    INICIALIZA ($i_{inicial}$)
    $i := i_{inicial}$
    Repite
        GENERA (j de $\mathcal{S}_i$)
        si $f(j) \leq f(i)$ **entonces** i:=j
    hasta $f(j) \geq f(i)$, para toda $j \in \mathcal{S}_i$
fin

Figura 1. Algoritmo de Búsqueda Local en pseudo-código

Un concepto importante en el análisis de algoritmos de búsqueda local es el de optimalidad local.

Definición 3 *Sea (S, f) denote una instancia de un problema de optimización combinatoria y sea $\mathcal{N}$ una estructura de vecindades, entonces $\hat{i} \in S$ se llama una* solución óptima local *o simplemente un* óptimo local *con respecto a $\mathcal{N}$ si $\hat{i}$ es mejor que o igual a, todas sus soluciones vecinas con respecto al costo. Específicamente, en el caso de minimización, $\hat{i}$ se llama* solución mínima local *o simplemente un* mínimo local *si*

$$f(\hat{i}) \leq f(j), \forall j \in S_{\hat{i}}, \tag{12}$$

y en el caso de maximización, $\hat{i}$ se llama una solución máxima local *o simplemente un* máximo local *si*

$$f(\hat{i}) \geq f(j), \forall j \in S_{\hat{i}}, \tag{13}$$

Definición 4 *Sea (S, f) denote una instancia de un problema de optimización combinatoria y sea $\mathcal{N}$ una estructura de vecindades. Entonces $\mathcal{N}$ se llama* exacta *si para cada $\hat{i} \in S$ que es localmente óptimo con respecto a $\mathcal{N}$, $\hat{i}$ también es globalmente óptimo.*

De aquí se observa que, los algoritmos de búsqueda local terminan con un óptimo local a menos que la estructura de vecindades sea exacta. Generalmente es difícil definir estructuras de vecindades exactas, ya que la definición de una estructura de vecindades exacta frecuentemente involucra hacer una búsqueda exhaustiva en todo el conjunto S.

La calidad del óptimo local obtenido por el algoritmo de búsqueda local, depende fuertemente de la solución inicial y para muchos problemas de optimización combinatoria no se tiene a la mano una solución inicial apropiada. Además la complejidad del problema, en el peor de los casos, no se conoce para muchos problemas.

Entre las ventajas de usar el algoritmo de búsqueda local es que se puede aplicar de manera general, a diferentes problemas específicos de optimización combinatoria, sin alterar su estructura, ya que únicamente se requiere de especificar la función de costo y la estructura de vecindades.

Algunas formas de relajar las desventajas del algoritmo de búsqueda local son las siguientes:

- Ejecutar el algoritmo de búsqueda local para un número grande de diferentes soluciones iniciales. Es claro que de manera asintótica (garantizando que todas las soluciones han sido usadas como inicial) el algoritmo encontrará el óptimo global con probabilidad uno.

- Introducir estructuras de vecindades complejas, de tal forma que se explore una parte grande del conjunto de soluciones. A menudo encontrar una estructura de este tipo requiere de un conocimiento experto del problema de optimización combinatoria tratado.

- Aceptar con ciertos límites transiciones correspondientes a un incremento del valor de la función de costo; note que en el algoritmo de búsqueda local solamente se aceptan transiciones que correspondan a soluciones que decrementen el costo.

Esta última alternativa es la que se explora con profundidad en el resto del trabajo ya que corresponde a la filosofía del algoritmo de recocido simulado.

CAPÍTULO 2

RECOCIDO SIMULADO

En el análisis y clasificación de algoritmos se observó que existen varios problemas de optimización combinatoria en la clase NP-completa. Estos problemas requieren de un gran esfuerzo computacional para resolverlos de manera exacta. Desafortunadamente una gran cantidad de problemas de optimización combinatoria, de mediana y gran escala, que se presentan en la práctica pertenecen a esta clase. Una alternativa que resulta adecuada para esta problemática es el uso de algoritmos heurísticos con los que no se puede garantizar que la solución que encuentran sea la óptima; sin embargo, su tiempo de ejecución puede acotarse por un polinomio en el tamaño de la instancia del problema.

El algoritmo de recocido simulado es una técnica de aplicabilidad general a problemas de optimización combinatoria cuyo principio es sencillo de entender pero su eficacia depende de la habilidad que se tenga para identificar y definir cada una de las componentes que requiere y que en este capítulo se describen. En este sentido, se puede pensar que la técnica del recocido simulado 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.

El concepto de recocido (*annealing*) en optimización combinatoria fué introducido a inicios de la década de los 80 por Kirkpatrick, Gellat y Vecchi [1983] e independientemente Černy [1985]. Este concepto está basado en una fuerte analogía entre el proceso físico de recocido de sólidos y el problema de resolver problemas de optimización combinatoria grandes. En este capítulo se da esta analogía con el fin de introducir el algoritmo de recocido simulado desde un punto de vista intuitivo. El capítulo se desarrolla como sigue: en la Sección 2.1 se describe el proceso de recocido de un sólido en términos físicos y su correspondiente modelación con técnicas de simulación; en la Sección 2.2 se introduce el algoritmo de recocido simulado como una analogía al proceso físico de recocido y se establecen algunos cantidades y resultados similares a la de los sistemas físicos, útiles en el desarrollo del algoritmo de recocido simulado.

2.1 EL PROCESO DE RECOCIDO DE UN SÓLIDO

El algoritmo de recocido simulado está basado en una analogía entre la simulación de recocido de sólidos y la problemática de resolver problemas de optimización combinatoria de gran escala. Por esta razón el algoritmo se conoce como *recocido simulado*. Recocido denota un proceso de calentamiento de un sólido a una temperatura en la que sus granos deformados recristalizan para producir nuevos granos. La temperatura de recocido o de recristalización, depende del tipo de material, del grado de deformación del mismo, además de su uso futuro. Seguida a la fase de calentamiento, viene un proceso de enfriamiento en donde la temperatura se baja poco a poco. De esta manera, cada vez que se baja la temperatura, las partículas se reacomodan en estados de más baja energía hasta que se obtiene un sólido con sus partículas acomodadas conforme a una estructura de cristal. Si se comienza con un valor máximo de 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 alcance en cada etapa el equilibrio térmico, el sólido congelará en un estado cuya estructura será amorfa en lugar de la estructura cristalina de más baja energía. La estructura amorfa está caracterizada por una imperfecta critalización del sólido.

El equilibrio térmico está caracterizado por la distribución de Boltzmann [TOD83]. 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 sólido. $Z(T)$ es una constante de normalización llamada la *función partición*, que esta definida como

$$Z(T) = \sum_j \exp\left(\frac{-E_j}{k_B T}\right), \tag{2}$$

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. Obviamente (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. Se puede observar en (1) que cuando el valor de T disminuye, la distribución de Boltzmann se concentra en los estados de menor energía mientras que si la temperatura se aproxima a cero, únicamente los estados con mínima energía tienen una probabilidad de ocurrencia diferente de cero.

Por lo dicho anteriormente, el proceso de recocido consta de dos pasos fundamentales que son:

- Incrementar la temperatura del baño térmico a un valor máximo.
- Decrementar *cuidadosamente* la temperatura del baño térmico hasta que las partículas se reacomoden por sí mismas en un estado de mínima energía, denominado el estado fundamental del sólido.

En el proceso de elevar la temperatura del sólido, todas las partículas se reacomodan aleatoriamente. En el estado fundamental, las partículas se acomodan en una retícula altamente estructurada y la energía del sistema es mínima. El estado fundamental del sólido se obtiene solamente si la temperatura máxima es suficientemente elevada y el proceso de enfriamiento es suficientemente bajo. De otra manera se obtendrá un estado amorfo denominado meta-estado.

El proceso físico de recocido puede modelarse exitosamente usando métodos de simulación (vea [BIN78] o [MET53]), el algoritmo introducido para tal próposito se basa en técnicas Monte Carlo y genera una sucesión de estados del sólido de la siguiente manera. Dado un estado i del sólido con energía E_i, se genera un estado subsecuente j aplicando un mecanismo de perturbación que transforma el estado actual en el siguiente estado por medio de una pequeña distorsión, por ejemplo, por el desplazamiento de una partícula. La energía del siguiente estado es E_j. Si la *diferencia de energía*, $E_j - E_i$, es menor o igual a cero, el estado j se acepta como el estado actual. Si la diferencia de energía es mayor que cero, el estado j se acepta con una probabilidad que esta dada por

$$\exp\left(\frac{E_i - E_j}{k_B T}\right), \tag{3}$$

donde T denote la temperatura del baño térmico y k_B es la constante de Boltzmann. La regla de decisión descrita arriba se conoce como el *criterio de Metropolis* y al algoritmo se le conoce como *algoritmo de Metropolis*.

Si la temperatura se baja poco a poco, el sólido puede alcanzar su equilibrio térmico en cada temperatura. En el algoritmo de Metropolis esto se lleva a cabo generando un número grande de transiciones para un valor dado de la temperatura.

2.2 EL ALGORITMO DE RECOCIDO SIMULADO

La simulación del proceso de recocido 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, se puede observar una analogía entre el sistema físico y un problema de optimización combinatoria en donde cada solución del problema de optimización combinatoria puede verse como un estado del sólido y los valores de la función objetivo para cada solución del problema de optimización combinatoria como los niveles de energía del sólido. En resumen, se puede pensar en 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.

Además, se introduce un parámetro que juega el papel equivalente de la temperatura. Este parámetro se llama el *parámetro de control.* El algoritmo de recocido simulado puede verse como una iteración del algoritmo de Metropolis, evaluado en valores decrecientes del parámetro de control.

Como en los algoritmos de búsqueda local se supone la existencia de una estructura de vecindades y un mecanismo de generación. Se introducen las siguientes definiciones.

Definición 1 *Sea (S, f) denote una instancia de un problema de optmización combinatoria y denote por i y j 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:*

$$P_c\{aceptar\ j\} = \begin{cases} 1 & si\ f(j) \leq f(i) \\ \exp\left(\frac{f(i)-f(j)}{c}\right) & si\ f(j) > f(i), \end{cases} \tag{4}$$

donde $c \in R^+$ denota el parámetro de control.

Claramente, el mecanismo de generación corresponde al mecanismo de perturbación en el algoritmo de Metropolis, mientras que el criterio de aceptación corresponde al criterio de Metropolis.

Definición 2 *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 denote el valor del parámetro de control y L_k el número de transiciones generadas en la k-ésima iteración del algoritmo de Metropolis. Entonces el algoritmo de recocido simulado puede describirse en pseudo-código como se muestra en la Figura 1.

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 $\exp\left(\frac{f(i)-f(j)}{c_k}\right) >$ numero 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 comienza llamando a un procedimiento de inicialización donde se definen la solución inicial, la temperatura inicial y el número inicial de generaciones necesarias para alcanzar el 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 los criterios de aceptación ya discutidos. 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 pequeñas desviaciones serán aceptadas 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.

Note que 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.

Comparando el algoritmo de recocido simulado con el algoritmo de búsqueda local, resulta evidente que recocido simulado puede verse como una generalización de búsqueda local y viene a ser idéntico a búsqueda local, en el caso que el parámetro de control c se tome igual a cero. En lo que respecta a la ejecución de ambos algoritmos, generalmente el algoritmo de recocido simulado obtiene mejores soluciones que el algoritmo de búsqueda local.

Equilibrio Estadístico

En esta sección se discuten algunos conceptos que emergen de la física estadística. Usando estas cantidades como guía, se puede analizar diferentes rasgos distintivos del algoritmo de recocido simulado. La suposición fundamental de la física estadística es la *hipótesis de ergodicidad* [TOD83], que establece que un sistema físico de muchas partículas es compatible con un conjunto estadístico (*statistical ensemble*), y que el correspondiente conjunto promedio determina los promedios de observables del sistema físico, es por ello que un número de cantidades útiles pueden derivarse para el sistema físico en equilibrio térmico. Ejemplos de esto son la energía promedio, el despliegue de la energía y la entropía. Además, si el conjunto es estacionario (que es el caso si el equilibrio térmico se alcanza), su densidad de probabilidad está dada como función de la energía del sistema. Sin embargo, aplicando el principio de probabilidades iguales, puede mostrarse que en equilibrio térmico la probabilidad que el sistema esté en un estado i con energía E_i viene dado por la distribución de Boltzmann (1). Esta distribución permite una evaluación analítica de las cantidades mencionadas al principio.

Se puede extender la analogía entre física estadística y optimización con recocido

simulado en el siguiente camino.

Conjetura. 1 *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 (4), el algoritmo de recocido simulado debe encontrar una solución $i \in S$ con una probabilidad igual a*

$$P_c\{X = i\} \equiv q_i(c) = \frac{1}{N_0(c)} \exp\left(-\frac{f(i)}{c}\right), \tag{5}$$

donde X es una variable aleatoria que denota la actual solución obtenida por el algoritmo de recocido simulado y

$$N_0(c) = \sum_{j \in S} \exp\left(-\frac{f(j)}{c}\right), \tag{6}$$

denota una constante de normalización.

El significado en la conjetura de una estructura de vecindades "adecuada" se explica con mayor detalle en el Capítulo 3. La distribución de probabilidad (5) 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 $\chi_{(A')} : A \to \{0, 1\}$ 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 (5), entonces*

$$\lim_{c \downarrow 0} q_i(c) \equiv q_i^* = \frac{1}{|S_{opt}|} \chi_{(S_{opt})}(i), \tag{7}$$

donde S_{opt} denota el conjunto de soluciones globalmente óptimas.

Prueba Usando el hecho de que para toda $a \leq 0$, $\lim_{x \downarrow 0} e^{\frac{a}{x}} = 1$ si $a = 0$ y 0 en otro caso,

obtenemos

$$\begin{aligned}
\lim_{c\downarrow 0} q_i(c) &= \lim_{c\downarrow 0} \frac{\exp\left(-\frac{f(i)}{c}\right)}{\sum_{j\in S}\exp\left(-\frac{f(j)}{c}\right)} \\
&= \lim_{c\downarrow 0} \frac{\exp\left(\frac{f_{opt}-f(i)}{c}\right)}{\sum_{j\in S}\exp\left(\frac{f_{opt}-f(j)}{c}\right)} \\
&= \lim_{c\downarrow 0} \frac{1}{\sum_{j\in S}\exp\left(\frac{f_{opt}-f(j)}{c}\right)}\chi_{(S_{opt})}(i) \\
&\quad + \lim_{c\downarrow 0} \frac{\exp\left(\frac{f_{opt}-f(i)}{c}\right)}{\sum_{j\in S}\exp\left(\frac{f_{opt}-f(j)}{c}\right)}\chi_{(S-S_{opt})}(i) \\
&= \frac{1}{|S_{opt}|}\chi_{(S_{opt})}(i) + 0,
\end{aligned}$$

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 (5) se obtiene en cada valor de c. Regresaremos a esta propiedad en el Capítulo 4.

Usando la distribución estacionaria de (5), un conjunto de cantidades útiles pueden definirse para problemas de optimización en un camino similar al de sistemas físicos con muchas partículas [TOD83] o conjuntos estadísticos [FEL50]. Aquí definimos lo siguiente:

Definición 3 *El* costo esperado $E_c(f)$ *en equilibrio esta definido como*

$$E_c(f) \equiv \langle f\rangle_c = \sum_{i\in S} f(i)P_c\{X=i\} = \sum_{i\in S} f(i)q_i(c). \tag{8}$$

Definición 4 *El* costo cuadrático esperado $E_c(f^2)$ *en equilibrio esta definido como*

$$E_c(f^2) \equiv \langle f^2\rangle_c = \sum_{i\in S} f^2(i)P_c\{X=i\} = \sum_{i\in S} f^2(i)q_i(c). \tag{9}$$

Definición 5 *La* varianza $Var_c(f)$ *del costo de equilibrio está definida como*

$$Var_c(f) \equiv \sigma_c^2 = \sum_{i\in S}(f(i)-E_c(f))^2P_c\{X=i\} = \sum_{i\in S}(f(i)-\langle f\rangle_c)^2q_i(c) = \langle f^2\rangle_c - \langle f\rangle_c^2. \tag{10}$$

Las notaciones $\langle f \rangle_c$, $\langle f^2 \rangle_c$ y σ_c^2 se introducen como notaciones cortas y se usan en el resto del texto.

Definición 6 *La* entropía *en equilibrio está definida como*

$$S_c \equiv -\sum_{i \in S} q_i(c) \ln q_i(c) \tag{11}$$

La entropía es una medida natural de la cantidad de desorden o información en un "sistema". Es una cantidad importante que posteriormente se analizará con mayor detalle.

Corolario 2 *Sea la distribución estacionaria dada por (5), 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 (8)-(11) y sustituyendo la expresión para la distibución estacionaria dada por (5).

Corolario 3 *Sea la distribución estacionaria dada por (5). Entonces se tiene*

$$\lim_{c \to \infty} \langle f \rangle_c \equiv \langle f \rangle_\infty = \frac{1}{|S|} \sum_{i \in S} f(i), \tag{14}$$

$$\lim_{c \downarrow 0} \langle f \rangle_c = f_{opt}, \tag{15}$$

$$\lim_{c \to \infty} \sigma_c^2 \equiv \sigma_\infty^2 = \frac{1}{|S|} \sum_{i \in S} (f(i) - \langle f \rangle_\infty)^2, \tag{16}$$

$$\lim_{c \downarrow 0} \sigma_c^2 = 0, \tag{17}$$

$$\lim_{c \to \infty} S_c \equiv S_\infty = \ln |S|, \tag{18}$$

$$\lim_{c \downarrow 0} S_c \equiv S_0 = \ln |S_{opt}|. \tag{19}$$

Prueba Las relaciones pueden fácilmente verificarse usando las definiciones de costo esperado (8), varianza (10) y entropía (11) sustituyéndolos en la distribución estacionaria de (5) 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 (11) se conoce en teoría de la información, que puede 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 (12) y (13) 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 $\ln|\mathcal{S}_{opt}|$, respectivamente.

La dependencia de la distribución estacionaria en (5) del parámetro de control c es el propósito del siguiente corolario.

Corolario 4 *Sea $(\mathcal{S}, f)$ denote una instancia de un problema de optimización combinatoria con $\mathcal{S}_{opt} \neq \mathcal{S}$ y sea $q_i(c)$ denote la distribución estacionaria asociada con el algoritmo de recocido simulado y dada por (5). Entonces se tiene*

(i) $\forall i \in \mathcal{S}_{opt}$*:*

$$\frac{\partial}{\partial c} q_i(c) < 0, \tag{20}$$

(ii) $\forall i \notin \mathcal{S}_{opt},\ f(i) \geq \langle f \rangle_\infty$*:*

$$\frac{\partial}{\partial c} q_i(c) > 0, \tag{21}$$

(iii) $\forall i \notin \mathcal{S}_{opt},\ f(i) < \langle f \rangle_\infty,\ \exists \tilde{c}_i > 0$*:*

$$\begin{array}{rcl} \frac{\partial}{\partial c} q_i(c) & > & 0 \quad si \quad c < \tilde{c}_i \\ & = & 0 \quad si \quad c = \tilde{c}_i \\ & < & 0 \quad si \quad c > \tilde{c}_i. \end{array} \tag{22}$$

Prueba De (6) se deriva la siguiente expresión:

$$\frac{\partial}{\partial c} N_0(c) = \sum_{j \in \mathcal{S}} \frac{f(j)}{c^2} \exp\left(\frac{-f(j)}{c}\right).$$

Por lo tanto, se obtiene

$$\begin{aligned}
\frac{\partial}{\partial c} q_i(c) &= \frac{\partial}{\partial c} \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0(c)} \\
&= \left\{ \frac{f(i)}{c^2} \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0(c)} - \frac{\exp\left(\frac{-f(i)}{c}\right)}{N_0^2(c)} \frac{\partial}{\partial c} N_0(c) \right\} \\
&= \frac{q_i(c)}{c^2} f(i) - \frac{q_i(c)}{c^2} \frac{\sum_{j \in S} f(j) \exp\left(\frac{-f(j)}{c}\right)}{N_0(c)} \\
&= \frac{q_i(c)}{c^2} (f(i) - \langle f \rangle_c).
\end{aligned} \tag{23}$$

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 $i \in \mathcal{S}$ y $c > 0$.

De (12), (14) y (15) 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 $\mathcal{S}_{opt} \neq \mathcal{S}$. De lo anterior, la prueba del teorema se sigue directamente.

Si $i \in \mathcal{S}_{opt}$ y $\mathcal{S}_{opt} \neq \mathcal{S}$, entonces $f(i) < \langle f \rangle_c$. Así, $\frac{\partial}{\partial c} q_i(c) < 0$ (vea 23), que completa la prueba de la parte (i).

Si $i \notin \mathcal{S}_{opt}$, 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 (23), se tiene que $\forall i \in \mathcal{S} - \mathcal{S}_{opt}$: $\frac{\partial}{\partial c} q_i(c) > 0$ si $f(i) \geq \langle f \rangle_\infty$, mientras que $\forall i \in \mathcal{S} - \mathcal{S}_{opt}$, donde $f(i) < \langle f \rangle_\infty$, existe $\tilde{c}_i > 0$ en donde $f(i) - \langle f \rangle_\infty$ cambia de signo. Consecuentemente se cumple que

$$\begin{aligned}
\frac{\partial}{\partial c} q_i(c) &> 0 \quad si\ c < \tilde{c}_i \\
&= 0 \quad si\ \ c = \tilde{c}_i \\
&< 0 \quad si\ \ c > \tilde{c}_i.
\end{aligned}$$

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 $\tilde{c}_i$ del parámetro de control, tal que para $c < \tilde{c}_i$, la probabilidad de encontrar esta solución decrece monótonamente cuando c decrece.

CAPÍTULO 3

PROBLEMAS CLÁSICOS Y APLICACIONES

El campo de la optimización combinatoria es extensa e interesante y surgen de ella una gran variedad de problemas prácticos y teóricos, algunos de ellos fáciles de resolver en forma numérica, pero otros con un alto grado de complejidad. Estos problemas, como ya se mencionó en el Capítulo 1, pertenecen a la clase NP-completa y actualmente son objeto de intensas investigaciones.

En este capítulo se discuten aspectos generales relacionados con las aplicaciones del algoritmo de recocido simulado. Usando estos aspectos como guía, se analizan algunos problemas de optimizacion combinatoria como ejemplos de problemas que pueden ser resueltos cuando el algoritmo de recocido simulado se aplica. Los problemas que se discuten aquí son los siguientes: problema del conjunto independiente, problema de asignación cuadrático y el problema de localización de plantas. Para cada una de estas tres familias de problemas, se presenta la bondad de la técnica de recocido simulado, así como los lineamientos a seguir para implantarlo en la computadora. Para tal propósito, se presentan en cada una de las secciones dedicadas a los problemas anteriores tres métodos alternativos de solución del problema: un método exacto, un método heurístico y un método de búsqueda local. El análisis comparativo de los cuatro métodos se hizo en cuanto a tiempo de ejecución así como a su cercanía con el valor óptimo de la función objetivo, cuando se conoce tal solución. También se formula un ejemplo real y se comentan los resultados así como la experiencia computacional.

El capítulo se desarrolla como sigue: en la Sección 3.1 se describen los aspectos generales que deben considerarse cuando se implanta el algoritmo de recocido simulado; en la Sección 3.2 se describe y analiza el problema de conjunto independiente, se presenta un algoritmo de solución exacto, un algoritmo de solución heurístico, el algoritmo de recocido simulado y el algoritmo de búsqueda local, se hace un análisis comparativo de los cuatro métodos y se presenta la experiencia computacional; en la Sección 3.3 se describe y analiza el problema de asignación cuadrático, se presenta un algoritmo de solución exacto, un algoritmo de solución heurístico, el algoritmo de recocido simulado y el algoritmo de búsqueda local, se hace un análisis comparativo de los cuatro métodos y se presenta la experiencia computacional; en la Sección 3.4 se describe y analiza el problema de localización de plantas, se presenta un algoritmo de solución exacto, un

algoritmo de solución heurístico, el algoritmo de recocido simulado y el algoritmo de búsqueda local, se hace un análisis comparativo de los cuatro métodos y se presenta la experiencia computacional; la Sección 3.5 describe un ejemplo de aplicación real de gran escala, que se puede encuadrar en el marco de problemas de localización óptima; y en la Sección 3.6 se da una lista de problemas para los que la técnica de recocido simulado se ha utilizado, así como su respectiva experiencia computacional.

3.1 ASPECTOS GENERALES DEL ALGORITMO

En las aplicaciones del algoritmo de recocido simulado comúnmente se desea implementarlo de manera que la sucesión de soluciones estén generadas a partir de valores decrecientes del parámetro de control. Las soluciones se generan continuamente tratando de transformar la solución actual en una subsecuente por medio de aplicar los mecanismos de generación y el criterio de aceptación. Las aplicaciones del algoritmo de recocido simulado requieren de la especificación de los siguientes puntos: (*i*) Una representación concisa del problema, (*ii*) un mecanismo de transición y (*iii*) un programa de enfriamiento. Cada uno de estos puntos se pueden enumerar con más detalle.

1. Una descripción concisa de la *representación del problema* consiste de una representación del espacio de soluciones y una expresión de la función de costo. La función de costo debe escogerse de manera que represente la efectividad de las soluciones con respecto al objetivo de optimización. La representación del problema y la función de costo deben darse por medio de expresiones simples que permitan una fácil manipulación.

2. La generación de ensayos para transformar la solución actual en una subsecuente consiste de tres pasos. Primero, se debe generar una nueva solución aplicando un mecanismo de generación. Enseguida se debe calcular la diferencia de costo de las dos soluciones, por último, se hace una decisión de aceptar o no, la nueva solución.

   La evaluación de la nueva solución es lo que más tiempo consume en el algoritmo de recocido simulado y por lo tanto debe hacerse lo más eficientemente posible. El mecanismo de generación usualmente se escoge de tal manera que se obtengan modificaciones simples para que puedan ser ejecutadas rápidamente, por ejemplo permutaciones, cambios o inversiones. El cálculo de la diferencia de costo se hace tomando en cuenta las diferencias entre ambas soluciones. Para muchas aplicaciones éste es el camino más rápido para calcular las diferencias en el costo. La decisión

de aceptar la nueva solución se basa en un criterio de aceptación; en la mayoría de los casos se aplica el criterio de Metropolis.

3. Ejecutar el proceso de recocido, requiere de la especificación de los parámetros que determinan el programa de enfriamiento. Estos parámetros son el valor inicial del parámetro de control, una función que especifique el decremento del parámetro de control, la longitud de cada bloque donde permanece constante la temperatura y el criterio de paro.

En lo que sigue se discuten las aplicaciones del algoritmo de recocido simulado a diferentes problemas de optimizacion combinatoria. La discusión se centra en los diferentes aspectos relacionados con los primeros dos puntos ya que los aspectos relacionados con el tercer punto se discuten en el Capílulo 5.

Las vecindades para los problemas que se presentan a continuación se escogieron de modo que los mecanismos de generación induzcan cadenas de Markov irreducibles y aperiódicas, para garantizar la convergencia asintótica al conjunto de soluciones óptimas.

Cabe mencionar que todos los problemas que serán tratados a continuación caen dentro de los problemas pertenecientes a la clase NP-completa.

3.2 EL PROBLEMA DE CONJUNTO INDEPENDIENTE

El objetivo fundamental de esta sección es presentar la bondad de la técnica de recocido simulado para el problema de conjunto independiente, así como los lineamientos a seguir para implantarlo en la computadora. Para tal propósito, se presentan en la sección tres métodos alternativos de solución del problema: un método exacto, un método heurístico y un método de búsqueda local. El método exacto y el heurístico que se seleccionaron, se reportan en la literatura como los más eficientes, vea [SYS83]. El estudio con el método de búsqueda local está enfocado a la investigación de si es bueno aceptar una solución más mala, a partir de la solución que actualmente se tenga.

El análisis comparativo de los cuatro algoritmos se hizo en cuanto a su cercanía con el número de elementos del conjunto independiente máximo. Para cada algoritmo se reporta la experiencia computacional y en la última subsección se hace un análisis comparativo de los cuatro métodos.

3.2.1 Descripción

Considere una gráfica $G(N, A)$ no dirigida, donde N es el conjunto de nodos y A es el conjunto de arcos. Un subconjunto I de N se dice que es un *Conjunto Independiente* de nodos, también conocido como *Conjunto Estable*, de G si no existen dos nodos en I que sean adyacentes; es decir, no existen dos nodos que estén ligados por un arco. Para la gráfica de la Figura 1, por ejemplo, los conjuntos de nodos $\{1, 5, 7\}$, $\{2, 6\}$, $\{2, 4, 6\}$, son conjuntos independientes.

Un conjunto independiente se llama *maximal* si no existe otro conjunto independiente en el que este contenido propiamente. Esto es, un conjunto independiente I es maximal si para todo conjunto H tal que $I \subset H$ con $I \neq H$, se tiene que H no es independiente. Por ejemplo para la gráfica de la Figura 1, los conjuntos $\{1, 5, 7\}$, $\{2, 4, 6\}$, $\{3, 6\}$, $\{1, 8\}$, son conjuntos independientes maximales y los conjuntos $\{1\}$, $\{2, 6\}$, $\{1, 5\}$ no son conjuntos independientes maximales.

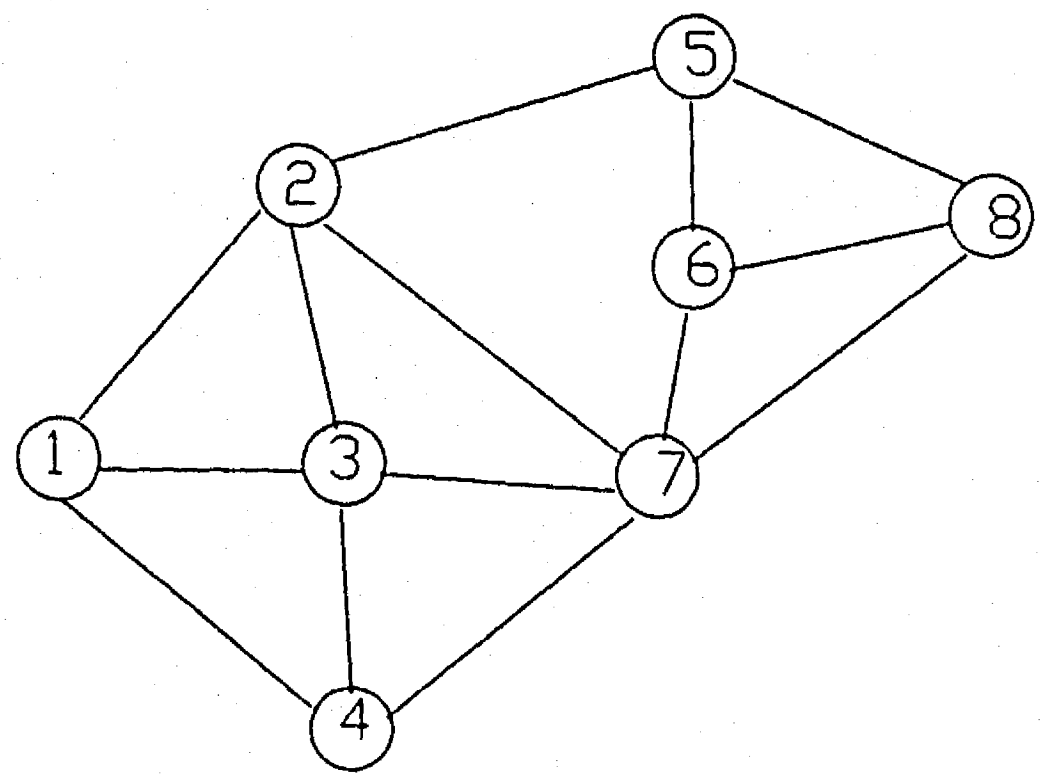


Figura 1. Gráfica.

En general existe más de un conjunto independiente maximal para una gráfica dada. También como se vió en los ejemplos, el número de nodos en los conjuntos independientes maximales no es necesariamente el mismo.

Si Q es la familia de conjuntos independientes asociados a una gráfica G, entonces el número.

$$\alpha[G] = \max_{I \in Q} |I|$$

se llama el *número independiente* o de *estabilidad* de una gráfica G, y el conjunto I para el que se obtiene este número se llama un *conjunto independiente máximo.*
En la gráfica de la Figura 1, la familia de conjuntos maximales está compuesta por:

$$\{\{1,5,7\}, \{1,6\}, \{1,8\}, \{2,4,6\}, \{2,4,8\}, \{3,5\}, \{3,6\}\ \{3,8\}, \{4,5\}\}.$$

El más grande de estos conjuntos tiene 3 elementos y por lo tanto $\alpha[G] = 3$. Cualquier conjunto de la familia maximal con 3 elementos es un conjunto independiente máximo.

Un problema real en donde se utiliza este modelo es el siguiente. Considere n proyectos que deben realizarse y suponga que cada proyecto i requiere de un subcojunto de recursos $R_i \subseteq \{r_1, r_2, ..., r_p\}$ donde r_j denota uno de los p recursos disponibles para su ejecución. Suponga adicionalmente que cada proyecto (dados los recursos que necesita) puede ejecutarse en un sólo periodo de tiempo. Se puede formar una gráfica G en donde cada nodo corresponde a un proyecto y con arcos (i, j) siempre que los proyectos i y j tengan algún requerimiento en común, es decir, cuando $R_i \cap R_j \neq \emptyset$. Un conjunto independiente maximal de nodos en G representa un conjunto de proyectos maximal que pueden ejecutarse simultáneamente durante un periodo de tiempo.

En un sistema dinámico nuevos proyectos se pueden incorporar para su realización en cada periodo de tiempo, de modo que la familia de conjuntos maximales independientes de G tiene que encontrarse a menudo, de forma tal que se escoja el conjunto maximal de proyectos que se ejecuten durante un periodo de tiempo. En un sistema práctico no es suficiente simplemente ejecutar el conjunto de proyectos que corresponden al máximo conjunto independiente en todo periodo, puesto que algunos proyectos pueden ser relegados indefinidamente. Otra forma es dar una penalización p_i a cada proyecto i que se vaya incrementando cada vez que el proyecto no se tome en cuenta y entonces en cada periodo de tiempo se escoja de la familia de conjuntos independientes maximales el que maximice alguna función de penalización sobre cada uno de sus nodos.

Aparentemente el calcular todos los conjuntos maximales de una gráfica G podría pensarse que es una tarea fácil, esta suposición es cierta para gráficas con pocos nodos pero cuando la gráfica sobrepasa a 20 nodos, el número de conjuntos maximales se incrementa en forma alarmante, conforme se incrementa el número de nodos. De la misma forma, encontrar un conjunto independiente máximo es un problema que es NP-completo y por lo tanto los algoritmos conocidos para resolver el problema de manera exacta, son de enumeración implícita.

3.2.2 Algoritmo Exacto

En lo que sigue se discute un método enumerativo debido a Christofides [CHR75] que según estudios de Syslo [SYS83] es de los métodos exactos más "eficientes" con que se cuenta en la actualidad para resolver este problema.

Descripción del Algoritmo

El algoritmo esencialmente es un algoritmo enumerativo de búsqueda sobre un árbol. En la etapa k, un conjunto independiente de nodos S_k se aumenta al agregar otro nodo adecuadamente seleccionado para producir un conjunto independiente S_{k+1}, en la etapa $k+1$ hasta que no sea posible encontrar otro nodo y de esta forma encontrar un conjunto independiente maximal. En la etapa k sea Q_k el conjunto de nodos más grande para el que $S_k \cap Q_k = \emptyset$, es decir, cualquier nodo de Q_k agregado a S_k produce un conjunto S_{k+1} que es independiente.

En algún momento durante el algoritmo, Q_k consiste de dos tipos de nodos: nodos en Q_k^- que ya han sido usados anteriormente en la búsqueda para aumentar S_k y nodos en Q_k^+ que no se han usado. Una ramificación hacia adelante en el árbol de búsqueda consiste en escoger un nodo $x_{i_k} \in Q_k^+$, agregarlo a S_k para producir.

$$S_{k+1} = S_k \cup \{x_{i_k}\} \tag{1}$$

y crear nuevos conjuntos Q_{k+1}

$$Q_{k+1}^- = Q_k^- - \Gamma(x_{i_k}) \tag{2}$$

y

$$Q_{k+1}^+ = Q_k^+ - \Gamma(x_{i_k}) - \{x_{i_k}\} \tag{3}$$

donde $\Gamma(x_{i_k})$ significa el conjunto de nodos x_j tales que $(x_{i_k}, x_j) \in A$.

Un paso hacia atrás del algoritmo en el árbol de búsqueda consiste en quitar x_{i_k} de S_{k+1} para obtener nuevamente S_k y eliminar x_{i_k} del conjunto Q_k^+ y agregarlo al conjunto Q_k^- para formar dos nuevos conjuntos Q_k^+ y Q_k^-.

Un conjunto S_k es un conjunto maximal independiente si no puede ser aumentado, es decir, solamente si $Q_k^+ = \emptyset$. Si $Q_k^- \neq \emptyset$, se sigue que el conjunto actual S_k fue aumentado en algún paso previo por algún nodo en Q_k^- y por lo tanto no es maximal. Si $Q_k^- = \emptyset$, el actual conjunto S_k no ha sido previamente aumentado y como los conjuntos se generan sin duplicarlos, S_k es un conjunto independiente maximal. Por lo tanto, una condición necesaria y suficiente para que S_k sea un conjunto independiente maximal es

$$Q_k^+ = Q_k^- = \emptyset \tag{4}$$

Es evidente que si una etapa se alcanza cuando algún nodo $x \in Q_k^-$ existe para el que $\Gamma(x) \cap Q_k^+ = \emptyset$, entonces sin tomar en cuenta de que nodo de Q_k^+ se use para aumentar S_k en cualquier número de ramificaciones hacia adelante, el nodo x nunca será removido de Q_p^- para cualquier paso $p > k$. Por lo tanto, la condición:

$$\exists x \in Q_k^- \quad \textit{de modo que} \quad \Gamma(x) \cap Q_k^+ = \emptyset \tag{5}$$

es suficiente para ejecutar un paso hacia atrás ya que no puede resultar ningún conjunto maximal independiente de ahí en adelante, a partir de S_k.

Como en todos los métodos de árbol de búsqueda, es bueno hacer pasos hacia atrás lo más pronto posible ya que esto tiende a limitar hacer búsquedas innecesarias por el árbol. Por lo tanto, es bueno forzar la condición (5) lo más pronto posible, escogiendo nodos que aumentan el conjunto S_k. Durante un paso hacia adelante se puede escoger cualquier nodo $x_{i_k} \in Q_k^+$ que aumente el conjunto S_k y en un paso hacia atrás x_{i_k} debe ser removido de Q_k^+ e insertado en Q_k^-. Ahora, si x_{i_k} se escoge como un nodo $\in \Gamma(x)$ para algún $x \in Q_k^-$, entonces en el correspondiente paso hacia atrás, el número

$$\Delta(x) = |\Gamma(x) \cap Q_k^+| \tag{6}$$

debe decrecer en uno (de su valor previo al completar los pasos hacia adelante y hacia atrás) de modo que la condición (5) está más próxima de ser verdadera. Así, un posible camino para escoger el nodo x_{i_k} para aumentar S_k es: primero determinar el nodo $x^* \in Q_k^-$ que da el menor valor de $\Delta(x^*)$ y entonces escoger x_{i_k} del conjunto $\Gamma(x^*) \cap Q_k^+$. Si se escoge x_{i_k} de esta manera, en todo paso debe provocarse que $\Delta(x^*)$ decrezca en uno en todo paso hacia atrás, cuando se hace al nivel k, hasta que eventualmente el nodo x^* satisface la condición (5), al tiempo que un paso hacia atrás ocurre.

Se debe notar que puesto que en un paso hacia atrás x_{i_k} entra a Q_k^- puede ser que esta operación haga que Δ tenga un valor más pequeño que el nodo previamente fijado x^* de modo que este nuevo nodo deber ser el blanco para forzar la condición (5). Esto es de particular importancia en la primera ramificación hacia adelante cuando $Q_k^- = \emptyset$.

El algoritmo puede enunciarse de la siguiente forma:

Algoritmo Exacto para el Problema de Conjunto Independiente Maximal

Propósito: Obtener todos los conjuntos independientes maximales de una gáfica $G(N, A)$ dada.

Descripción:
Inicialización
Paso 1: Sea $S_0 = Q_0^- = \emptyset$, $Q_0^+ = N$ y $k = 0$.
Paso hacia adelante
Paso 2: Escoja un nodo $x_{i_k} \in Q_k^+$ como ya se mencionó anteriormente
y forme S_{k+1}, Q_{k+1}^+ y Q_{k+1}^- dejando Q_k^+ y Q_k^- intactos.
Haga $k = k + 1$
Prueba
Paso 3: Si la condición (5) se cumple vaya al paso 5, de otra manera vaya al paso 4.
Paso 4: Si $Q_k^+ = Q_k^- = \emptyset$, el conjunto S_k es un conjunto independiente maximal, vaya al paso 5. Si $Q_k^+ = \emptyset$ pero $Q_k^- \neq \emptyset$ vaya al paso 5. De otra manera vaya al paso 2.
Paso hacia atrás
Paso 5: Sea $k = k - 1$. Renueva x_{i_k} de S_{k+1} para producir Sk. Recupere Q_k^- y Q_k^+, remueva x_{i_k} de Q_k^+ y agréguelo a Q_k^-. Si $k = 0$ y $Q_0^+ = \emptyset$.
Pare. (Se han generado todos los conjuntos independientes maximales).
De otra manera vaya al paso 3.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria gráficas conectadas. Para valores preestablecidos del número n de nodos y el número m de arcos, se generaron 100 gráficas aleatorias. La Tabla 1 reporta el tiempo promedio (en segundos) y los valores del número de nodos y de arcos considerados. Para cada valor del número

de nodos, se consideraron cuatro casos (en cuanto a número de arcos): gráfica con pocos arcos, gráfica con un número normal de arcos, gráfica medianamente densa y gráfica densa.

Num. Nodos	EXACTO Num. Arcos / Tiempo(seg.)			
	15	20	30	40
10	0.01	0.01	0.01	0.01
	30	50	70	90
15	0.01	0.04	0.05	0.03
	40	80	120	160
20	0.05	0.14	0.33	0.29
	60	120	240	350
30	0.53	3.02	21.13	8.20
	80	240	320	600
40	2.57	336.82	125.00	4648.49
	100	200	400	800
50	40.18	1354.57	7326.73	—

Tabla 1. Tiempos promedio de ejecución del algoritmo exacto para el problema de conjunto independiente.

3.2.3 Algoritmo Heurístico

El algoritmo descrito en la subsección anterior es un método exacto para encontrar todos los conjuntos independientes maximales y en particular los conjuntos independientes máximos. Como todos los métodos de enumeración, tiene el problema que cuando el número de nodos en la gráfica crece, el tamaño del árbol de búsqueda crece de manera no polinomial con respecto al número de nodos. Esto da como consecuencia que para instancias grandes, el algoritmo no sea aplicable. Una alternativa para encontrar "buenas" soluciones al problema es la de desarrollar algoritmos heurísticos que resuelvan el problema dando una solución subóptima lo más cercana posible del óptimo.

En esta subsección se describe un algoritmo heurístico propuesto por Syslo, Deo y Kowalik [SYS83]. Syslo y et. al. reportan que el comportamiento de este algoritmo tiene un mejor

comportamiento que cualquier otro algoritmo de su tipo en el peor de los casos.

Descripción del Algoritmo

El algoritmo es sencillo y se puede describir de la siguiente manera. Sea $G = (N, A)$ una gráfica. Primero se encuentra el nodo $u \in N$ con el grado mínimo (el grado de un nodo se define como el número de arcos que inciden en él). Este nodo se agrega a S. Una vez agregado el nodo a S, se elimina de N y también se eliminan de N todos los nodos adyacentes al nodo u. De esta manera se obtiene una subgráfica $G_1(N_1, A_1)$ de G. Una vez que se tiene la subgráfica $G_1(N_1, A_1)$ nuevamente se selecciona un nodo u de grado mínimo se ejecuta el procedimiento anterior para encontrar una subgráfica $G_2(N_2, A_2)$. Se sigue de esta manera hasta que para alguna iteración k, $N_k = \emptyset$. Obviamente el conjunto S encontrado por medio de este procedimiento es un conjunto independiente maximal cuya cardinalidad es "cercana" al conjunto máximo independiente.

Este proceso se puede realizar mediante el siguiente algoritmo.

Algoritmo Heurístico para el Problema de Conjunto Independiente Máximo

Propósito: Obtiene un conjunto independiente maximal de la mayor cardinalidad posible de la gráfica $G(N, A)$ dada.

Descripción:
Paso 1: Tome $N_1 = N$, $S = \emptyset$, $A_1 = A$
Paso 2: Si $N_1 = \emptyset$, Termine. Si no, encuentre el nodo u de grado mínimo en $G_1(N_1, A_1)$
Paso 3: Haga $S = S \cup \{u\}$
$N_1 = N_1 - \{u\} - \{v \in N_1 : (v, u) \in A_1\}$
$A_1 = A_1 - \{(v, u) : (v, u) \in A_1\}$
vaya al paso 2.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria gráficas conectadas. Para valores preestablecidos del número n de nodos y el número m de arcos, se generaron 100 gráficas aleatorias. La Tabla 2 reporta el tiempo promedio (en segundos)

y los valores del número de nodos y de arcos considerados. Para cada valor del número de nodos, se consideraron cuatro casos (en cuanto a número de arcos): gráfica con pocos arcos, gráfica con un número normal de arcos, gráfica medianamente densa y gráfica densa.

Num. Nodos	HEURÍSTICO Num. Arcos / Tiempo(seg.)			
	15	20	30	40
10	0.00	0.00	0.00	0.00
	30	50	70	90
15	0.01	0.01	0.01	0.01
	40	80	120	160
20	0.00	0.01	0.01	0.01
	60	120	240	350
30	0.01	0.01	0.02	0.01
	80	240	320	600
40	0.02	0.02	0.04	0.39
	100	200	400	800
50	0.06	0.02	0.04	0.43

Tabla 2. Tiempos promedio de ejecución del algoritmo heurístico para el problema de conjunto independiente.

3.2.4 Algoritmo de Recocido Simulado

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 que se conoce la solución.
Para aplicar el algoritmo de recocido simulado se necesitan definir ciertos puntos que a continuación se describen.

Para el caso del problema de conjunto independiente, se tiene que agregar adicionalmente a la función objetivo, un costo para cuando se tengan soluciones infactibles; esto es, subconjuntos de N que no necesariamente son conjuntos independientes. Para este problema es apropiado particionar la familia de conjuntos de todos los subconjuntos de

N en dos subfamilias; la familia de soluciones factibles, es decir, la familia de conjuntos que son independientes y la familia de subconjuntos que no son independientes. La infactibilidad de algunas soluciones puede usarse ventajosamente por el algoritmo de recocido simulado para evitar quedar atrapado en un conjunto independiente maximal de menor cardinalidad a la del máximo.

Descripción del Algoritmo

El algoritmo de recocido simulado puede aplicarse de la siguiente manera.

- El espacio de soluciones S consiste de todas las posibles particiones del conjunto N en subconjuntos N' y $N \backslash N'$. El conjunto S' consiste de todas las particiones donde $\forall u, v \in N' : (u, v) \notin A$.

- La función de costo, que debe ser maximizada, se escoge como

$$f(N') = |N'| - \lambda |A'|, \tag{7}$$

  donde A' denota el conjunto de arcos $(u, v) \in A$ con $u, v \in N'$ y λ denota el *factor de peso* que debe ser mayor que 1 y juega el papel de penalización en el algoritmo. Las soluciones factibles, únicamente contribuyen en el primer término de la función de costo. Soluciones infactibles contribuyen en el segundo término y por lo tanto bajan el costo de la partición para puntos con igual cardinalidad. Así, el segundo término puede verse como una función de penalización que castiga la presencia de arcos entre nodos de la partición.

- La generación de nuevas soluciones se hace de manera aleatoria, escogiendo un nodo $u' \in N$ y cambiándolo de N' a $N \backslash N'$ si $u' \in N'$ o viceversa. La diferencia en costo viene dada por

$$\Delta f = (\chi_{(N \backslash N')}(u') - \chi_{(N')}(u')) \left(1 - \lambda \sum_{(u', v) \in A, v \in N'} 1 \right). \tag{8}$$

Claramente, el espacio de soluciones para el conjunto independiente puede hacerse restrictivo solamente al conjunto de soluciones factibles. Además la presencia de soluciones infactibles permite que se defina la función de costo de manera suave haciendo que el algoritmo de recocido simulado escape fácilmente de óptimos locales ya que existen más soluciones con diferencias pequeñas en costo en la vecindad de un óptimo local y así se

incrementa la probabilidad de escape. Agrandar el conjunto de soluciones factibles con el conjunto de soluciones infactibles en este caso (como en muchos) hace posible el uso de un mecanismo de generación simple que a menudo permite una convergencia rápida.

El algoritmo que describe el procedimiento de recocido puede enunciarse de la manera siguiente.

Algoritmo de Recocido Simulado para el Problema de Conjunto Independiente Máximo

Propósito: Obtiene un conjunto independiente maximal de la mayor cardinalidad posible de la gráfica $G(N, A)$ dada.

Descripción:

Paso 1: Tome c un valor grande apropiado.

Paso 2: Tome $N' \subset \{1, 2, ..., n\}$ arbitrario.
Obtenga el valor correspondiente de la función objetivo $f(N')$.

Paso 3: Asigne el valor verdadero a la variable lógica CHANGE.

Paso 4: Si la variable CHANGE tiene valor falso, entonces pare.

Paso 5: Haga CHANGE igual a falso.

Paso 6: Repita los puntos 1-9 r veces.

1.- Seleccione aleatoriamente un número $u' \in \{1, 2, ..., n\}$.
2.- Evalue el valor Δf de la expresión (8)
3.- Si $\Delta f < 0$ entonces vaya a 7.
4.- Sea $P(\Delta f) = \exp(-\Delta f/c)$.
5.- Genere un número aleatorio x, uniformemente distribuido en $(0, 1)$.
6.- Si $x \geq P(\Delta f)$ entonces vaya a 9.
7.- Haga $N' \leftarrow N' \cup \{u'\}$ si $u' \notin N'$ o $N' \leftarrow N' - \{u'\}$ si $u' \in N'$
8.- Si $\Delta f \neq 0$ entonces CHANGE $\leftarrow$ verdadero.
9.- Fin del paso 6.

Paso 7 Reduzca c e incremente r.

Paso 8 Vaya al paso 4.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria gráficas conectadas. Para valores preestablecidos del número n de nodos y el número m de arcos, se generaron 100 gráficas aleatorias. La Tabla 3 reporta el tiempo promedio (en segundos) y los valores del número de nodos y de arcos considerados. Para cada valor del número de nodos, se consideraron cuatro casos (en cuanto a número de arcos): gráfica con pocos arcos, gráfica con un número normal de arcos, gráfica medianamente densa y gráfica densa.

Num. Nodos	RECOCIDO SIMULADO Num. Arcos / Tiempo(seg.)			
	15	20	30	40
10	2.05	2.64	3.20	4.48
	30	50	70	90
15	2.16	3.15	4.47	5.50
	40	80	120	160
20	3.22	4.83	5.69	6.96
	60	120	240	350
30	3.57	5.78	6.17	7.26
	80	240	320	600
40	4.24	5.89	6.69	7.94
	100	200	400	800
50	6.13	6.78	7.17	8.31

Tabla 3. Tiempos promedio de ejecución del algoritmo de recocido simulado para el problema de conjunto independiente.

3.2.5 Algoritmo de Búsqueda Local

Uno de los puntos importantes del algoritmo de recocido simulado es el hecho de que cada que se ejecuta el **Paso 6** del mismo, no solamente acepta subconjuntos de nodos con mayor cardinalidad que el actual, sino que también se aceptan con cierta probabilidad subconjuntos cuya cardinalidad es menor e incluso subconjuntos que no necesariamente son conjuntos independientes. Esta situación cada vez es menos frecuente conforme el valor del parámetro de control c se decrementa. Sin embargo queda en duda el hecho

de si el aceptar soluciones "malas" da algún beneficio. Con el objeto de poder comparar los resultados del algoritmo de recocido simulado con el algoritmo de búsqueda local correspondiente, también se implantó este último en la computadora.
Siguiendo la misma notación que la ya dada para el algoritmo de recocido simulado y tomando en cuenta que si en el algoritmo de recocido simulado se toma el parámetro de control c desde el principio cercano a cero, el algoritmo que nos queda es un algoritmo de búsqueda local, ya que, la probabilidad de aceptar soluciones más malas que la actual prácticamente se reduce a cero. El algoritmo de búsqueda local para obtener un conjunto independiente máximo, se puede describir de la siguiente manera:

Algoritmo de Búsqueda Local para el Problema de Conjunto Independiente Máximo

Propósito Obtiene un conjunto independiente maximal de la mayor cardinalidad posible de la gráfica $G(N, A)$ dada.

Descripción:
Paso 1: Tome $N' \subset \{1, 2, ..., n\}$ arbitrario.
Obtenga el valor correspondiente de la función objetivo $f(N')$.
Paso 2: Asigne el valor verdadero a la variable lógica CHANGE.
Paso 3: Si la variable CHANGE tiene valor falso, entonces pare.
Paso 4: Haga CHANGE igual a falso.
Paso 5: Repita los puntos 1-6 $2 * n$ veces.
1.- Seleccione aleatoriamente un número $u' \in \{1, 2, ..., n\}$.
2.- Evalue el valor Δf de la expresión (8)
3.- Si $\Delta f \geq 0$ entonces vaya a 6.
4.- Haga $N' \leftarrow N' \cup \{u'\}$ si $u' \notin N'$ o $N' \leftarrow N' - \{u'\}$ si $u' \in N'$
5.- CHANGE $\leftarrow$ verdadero.
6.- Fin del paso 5.
Paso 6 Vaya al paso 3.

Experiencia Computacional
La prueba del algoritmo se llevó a cabo generando de manera aleatoria gráficas conectadas. Para valores preestablecidos del número n de nodos y el número m de arcos, se

generaron 100 gráficas aleatorias. La Tabla 4 reporta el tiempo promedio (en segundos) y los valores del número de nodos y de arcos considerados. Para cada valor del número de nodos, se consideraron cuatro casos (en cuanto a número de arcos): gráfica con pocos arcos, gráfica con un número normal de arcos, gráfica medianamente densa y gráfica densa.

Num. Nodos	BÚSQUEDA LOCAL Num. Arcos / Tiempo(seg.)			
	15	20	30	40
10	0.06	0.08	0.14	0.19
	30	50	70	90
15	0.18	0.34	0.54	0.77
	40	80	120	160
20	0.32	0.77	1.36	2.07
	60	120	240	350
30	0.73	1.74	4.66	2.76
	80	240	320	600
40	1.28	5.41	8.26	22.29
	100	200	400	800
50	1.99	4.82	12.89	32.42

Tabla 4. Tiempos promedio de ejecución del algoritmo de búsqueda local para el problema de conjunto independiente.

3.2.6 Análisis Comparativo de los Métodos

Como ya se mencionó, se generaron en forma aleatoria instancias del problema de conjunto independiente. Para cada una de estas instancias generadas, se corrieron los cuatro algoritmos y se calculó el tiempo promedio de solución. Los resultados aparecen en las Tablas 1, 2, 3 y 4.

Cabe mencionar que 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 incremente el parámetro r. Para el análisis realizado en este

documento, se tomó un valor inicial de c de tal manera que la tasa de aceptación de las permutaciones propuestas fuera mayor o igual a 0.8 y los decrementos fueran del 0.1 (paso 7 del algoritmo). El número r inicial que se tomó en cada caso fué $r = n$ y los incrementos fueron de 0.05 (paso 7 del algoritmo).

También se calculó la eficiencia de cada uno de los algoritmos por medio de la ecuación siguiente:

$$eficiencia = 1 - \frac{z - z_{opt}}{z_{opt}} \tag{9}$$

donde z denota el valor de la función objetivo en la solución obtenida y z_{opt} es el valor óptimo de la instancia del problema.

Los cálculos que aparecen en la Tabla 5 son con base a los promedios obtenidos en las 100 instancias resultas para cada valor del número de arcos m y del número de nodos n de los valores en la función objetivo de la solución correspondiente.

En la Tabla 5, puede observarse que las eficiencias del algoritmo de recocido simulado y las del algoritmo heurístico son las más elevadas. Con base en estas observaciones se puede concluir que ambos algoritmos tienen una eficiencia similar en cuanto a los valores en la función objetivo, sin embargo, los tiempos de ejecución del algoritmo heurístico son mucho menores que los del algoritmo de recocido simulado y esto hace preferible al algoritmo heurístico. Una ventaja que tiene el algoritmo de recocido simulado sobre el algoritmo heurístico es que cada vez que se ejecuta el algoritmo de recocido simulado, da una solución diferente y por lo tanto cuando se ejecuta varias veces se tiene una probabilidad de 1.0 de obtener la solución óptima del problema. Para el algoritmo heurístico, lo anterior no es válido, ya que cada vez que se ejecuta, da la misma solución. En este aspecto recocido simulado tiene ventaja sobre el algoritmo heurístico.

Las pruebas computacionales con el problema de conjunto independiente, revelan un buen 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 obtienen en todos los casos la solución óptima. Además de que la implantación del algoritmo en la computadora, no requiere de gran capacidad de memoria y permite un tiempo razonable para obtener buenas soluciones en las computadoras personales.

		ALGORITMO (eficiencia)			
		EXACTO	HEURISTICO	REC. SIM.	BUS. LOC.
n	m				
10	15	1.00	0.9977	0.9954	0.9002
	20	1.00	0.9975	0.9944	0.8532
	30	1.00	0.9968	0.9256	0.7832
	40	1.00	0.9951	0.9903	0.8447
15	30	1.00	0.9984	0.9830	0.8647
	50	1.00	0.9978	0.9886	0.8170
	70	1.00	1.0000	0.9871	0.7886
	90	1.00	0.9962	0.9827	0.8046
20	40	1.00	0.9988	0.9865	0.8418
	80	1.00	0.9982	0.9813	0.8227
	120	1.00	0.9975	0.9818	0.8064
	160	1.00	0.9966	0.9771	0.8293
30	60	1.00	0.9992	0.9892	0.8496
	120	1.00	0.9988	0.9867	0.8135
	240	1.00	0.9981	0.9810	0.7907
	350	1.00	1.0000	0.9754	0.8182
40	80	1.00	0.9984	0.9834	0.8644
	240	1.00	0.9900	0.9823	0.8365
	320	1.00	0.9981	0.9942	0.8102
	600	1.00	0.9687	0.9725	0.7187
50	100	1.00	0.9982	0.9791	0.7876
	200	1.00	0.9976	0.9821	0.7838
	400	1.00	0.9800	0.9802	0.7641

Tabla 5. Comparación de los algoritmos que resuelven el problema de conjunto independiente con respecto a su eficiencia.

Finalmente se mencioná que todos los programas computacionales y las corridas de los mismos, se realizaron en una computadora IBM-PS/70 de 25 mhz con procesador 80386 y sin coprocesador matemático.

3.3 EL PROBLEMA DE LOCALIZACIÓN DE PLANTAS

El objetivo fundamental de esta sección es presentar la bondad de la técnica de recocido simulado para el problema de localización de plantas, así como los lineamientos a seguir para implantarlo en la computadora. Para tal propósito, se presentan en la sección tres métodos alternativos de solución del problema: un método exacto, un método heurístico y un método de búsqueda local. El método exacto y el heurístico que se seleccionaron, se reportan en la literatura como los más eficientes, vea [LOV88]. El estudio con el método de búsqueda local está enfocado a la investigación de si es bueno aceptar una solución más mala, a partir de la solución que actualmente se tenga.

El análisis comparativo de los cuatro algoritmos se hizo en cuanto a los valores que se obtienen en la función objetivo. Para cada algoritmo se reporta la experiencia computacional y en la última subsección se hace un análisis comparativo de los cuatro métodos.

3.3.1 Descripción

El problema de localización de plantas tiene 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 $i = 1, ..., n$. La demanda tiene que satisfacerse manufacturando el producto dentro de la región. Se necesitan m o menos plantas para manufacturar 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 sitios es un problema de optimización combinatoria. Una vez que se conoce una solución de este problema, es decir, se ha decidido donde colocar las plantas, el problema de determinar la cantidad óptima transportada a lo largo de las rutas es un simple problema de transporte. Así, cada vez que se asignan las plantas a sitios, se debe resolver un problema de transporte para obtener la distribución óptima del producto.

Este problema puede formularse 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.
$y_{ij} = 1$ 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 sitio j.

Entonces el problema de localización de plantas puede formularse en términos matemáticos como:

$$\min \sum_i \sum_j c_{ij} x_{ij} + \sum_i f_i y_i + \sum_i \sum_j f_{ij} y_{ij} \tag{10}$$

$$\begin{aligned}
\sum_j x_{ij} - k_i y_i &\leq 0 \ \forall i \\
x_{ij} - k_{ij} y_{ij} &\leq 0 \ \forall i,j \\
\sum_i x_{ij} &\geq d_j \ \forall j \\
\sum_i y_i &\leq m \\
x_{ij} \geq 0 \ \forall i,j; \ y_i, y_{ij} &= 0 \, o \, 1 \ \forall i,j
\end{aligned}$$

Una aplicación de este problema se reporta en la sección 5 de este capítulo.

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.

3.3.2 Algoritmo Exacto

En esta subsección se introduce un algoritmo exacto para resolver el problema de localización de plantas. Este algoritmo se reporta como una algoritmo eficiente para resolver el problema de manera exacta (vea [LOV88]) y originalmente lo propone Khumawala ([KHU72]). El algoritmo de Khumawala es un algoritmo de ramificación y acotamiento y resuelve el siguiente problema:

Sea m el número de plantas potenciales a instalar (sin límite de capacidad) y n el

número de consumidores:

$$minz = \sum_i \sum_j c_{ij} x_{ij} + \sum_i f_i y_i \tag{11}$$

sujeto a

$$\sum_{i \in N_j} x_{ij} = 1 \qquad j = 1, 2, ..., n,$$

$$0 \leq \sum_{j \in P_i} \leq n_i y_i \qquad i = 1, 2, ..., m,$$

$$y_i = 0 \, o \, 1 \qquad i = 1, 2, ..., m,$$

donde, $c_{ij} = t_{ij} d_j$.

t_{ij} : Costo unitario del producto de la planta i al consumidor j.

d_j : demanda del consumidor j.

f_i : Costo fijo asociado a la planta i.

x_{ij} : Porcentaje de d_j abastecido desde la planta i.

y_i : Es igual a 1 o 0, dependiendo de si la planta i está abierta o cerrada.

N_j : Conjunto de almacenes que pueden abastecer al consumidor j.

P_i : Conjunto de aquellos consumidores que pueden ser abastecidos por la planta i.

n_i : Número de elementos de P_i.

Primero se resuelve el problema (11) de manera lineal, relajando las restricciones de integrabilidad. Sea z_0 la solución a este problema. Si las restricciones de integrabilidad se cumplen, entonces el problema está resuelto, de otra manera, se escoje una variable y_k que viola estas condiciones y se formulan dos nuevos problemas. (a) La restricción $y_k = 0$ se adiciona al problema y este es nuevamente resuelto en forma lineal, sea z_1 la solución; claramente $z_1 \geq z_0$. (b) La restricción $y_k = 1$ se adiciona al problema original y este es nuevamente resuelto en forma lineal, sea z_2 la solución; claramente $z_2 \geq z_0$.

Entonces $\bar{z} = \min\{z_1, z_2\}$ es una cota inferior para z. Este procedimiento resulta en la construcción del árbol de búsqueda en donde cada una de las ramas se corta hasta que se obtiene una mejor solución del problema (11) o cuando la cota inferior asociada al subproblema es mayor o igual al valor de la mejor solución encontrada. El algoritmo para cuando ya no existen nodos que explorar sobre el árbol de búsqueda. En este momento, la mejor solución encontrada, es la solución óptima del problema.

En cada nodo del árbol de búsqueda existen tres conjuntos de índices, K_0, K_1 y K_2, donde K_0 es el conjunto de índices asociados a plantas que están abiertas en forma fija;

K_1 es el conjunto de índices asociados a plantas que están cerradas en forma fija y K_2 es el conjunto de índices asociados a plantas que están libres.

La solución óptima del problema lineal en este nodo es:

$$\begin{aligned} x_{ij} &= 1 \text{ si } c_{ij} + g_i/n_i = \min_{k \in K_1 \cup K_2} [c_{kj} + g_k/n_k], \\ &= 0 \quad \textit{en otro caso}, \\ y_i &= 0 \qquad\qquad i \in K_0, \\ &= \sum_{j \in P_i} x_{ij}/n_i, \qquad i \in K_2, \\ y_i &= 1, \qquad\qquad i \in K_1, \end{aligned}$$

donde

$$\begin{aligned} g_k &= f_k, \qquad k \in K_2, \\ &= 0, \qquad k \in K_1. \end{aligned}$$

Existen algunas simplificaciones para reducir el número de ramas sobre el árbol de búsqueda. Son las siguientes:

1.- Cota mínima para abrir una planta. Para $i \in K_2$, $j \in P_i$, calcule

$$\begin{aligned} \nabla_{ij} &= \min_{k \in N_j \cap (K_1 \cup K_2); k \neq i} [\max(c_{kj} - c_{ij}, 0)], \\ \Delta_i &= \sum_{j \in P_i} \nabla_{ij} - f_i. \end{aligned}$$

si $\Delta_i > 0$, entonces $y_i = 1$ para todas las ramas que salgan del nodo. ∇_{ij} mide el costo mínimo que el consumidor j puede tener si la planta i se abre, cuando se considera con respecto a todas las plantas no cerradas en este nodo. Claramente, si la suma de este mínino para la planta i sobre todos los consumidores que puede proveer, excede su costo fijo f_i, entonces la planta i siempre debe estar abierta en este nodo.

2.- La segunda simplificación se refiere a la reducción de n_i. Si para $i \in K_2, j \in P_i$

$$\min_{k \in K_1 \cap N_j} (c_{kj} - c_{ij}) < 0,$$

entonces n_i se reduce en uno. De hecho, si la desigualdad se cumple para toda $j \in P_i$, entonces $P_i = \emptyset$, $n_i = 0$ y $y_i = 0$ para todas las ramas que salgan de este nodo.

Claramente, si una planta que ya este abierta puede suministrar al consumidor j más barato que cualquiera de las que están libres en el nodo, entonces este consumidor debe ser abastecido por cualquiera de las plantas abiertas del nodo. Por lo tanto, este consumidor, no se debe considerar como consumidor potencial de las plantas libres en este nodo.

3.- La tercera simplificación determina una cota máxima en la redución de costo por abrir una planta. Si esta cota es negativa, la planta debe permanecer fija en cerrada.

Para $i \in K_2, j \in P_i$

$$\begin{aligned} \omega_{ij} &= \min_{k \in K_1 \cap N_j} [\max(c_{kj} - c_{ij}, 0)], \\ \Omega_i &= \sum_{j \in P_i} \omega_{ij} - f_i. \end{aligned}$$

si $\Omega_i < 0$, entonces $y_i = 0$ para todas las ramas que salgan de este nodo. La ω_{ij} es similar a la ∇_{ij} de la primera simplificación, excepto que aquí las comparaciones se hacen únicamente sobre todas las plantas fijas en estar abiertas; es decir, ω_{ij} es el ahorro mínimo que puede hacerse por suministrar al consumidor j si la planta i está abierta cuando se considera sobre todas las plantas abiertas fijas en este nodo. Es claro que si la suma de estos ahorros para la planta i sobre todos los consumidores que puede proveer es menor que el costo fijo f_i, entonces esta planta debe estar cerrada y eliminarla para consideraciones adicionales.

Estas consideraciones deben hacerse a cada nodo tantas veces como sea necesario hasta que ya no se puedan hacer operaciones de cerrar o abrir plantas en forma definitiva.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria instancias del problema de localización de plantas. Para valores preestablecidos del número m de plantas y el número n 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).

La Tabla 6 reporta el tiempo promedio (en segundos) y los valores del número de plantas y de consumidores considerados. Para cada valor del número de plantas, se

consideraron cuatro casos (en cuanto al número de consumidores).

Num. Plantas	EXACTO Num. Cons. / Tiempo(seg.)			
	5	10	15	20
5	0.95	1.16	1.28	1.36
	10	15	25	30
10	2.20	2.68	3.93	5.05
	15	25	35	40
15	4.64	9.24	15.52	17.33
	20	25	35	40
20	15.89	17.89	21.23	30.41
	25	30	35	40
25	32.27	76.26	84.48	95.47
	30	40	50	60
30	60.14	99.55	164.97	235.60
	35	40	50	60
35	209.48	214.28	237.80	437.75
	40	45	50	60
40	396.18	436.76	1050.76	1504.63

Tabla 6. Tiempos promedio de ejecución del algoritmo exacto para el problema de localización de plantas

3.3.3 Algoritmo Heurístico

El algoritmo descrito en la subsección anterior es un método exacto para la solución al problema de localización de plantas. Como todos los métodos de ramificación y acotamiento, tiene el problema que cuando el número de plantas en la instacia crece, el tamaño del árbol de búsqueda crece de manera no polinomial con respecto a este número. Esto da como consecuencia que para instancias grandes, el algoritmo no sea aplicable. Una alternativa para encontrar "buenas" soluciones al problema es la de desarrollar algoritmos heurísticos que resuelvan el problema dando una solución subóptima lo más cercana posible del óptimo. En esta subsección se describe una modificación al algoritmo heurístico propuesto por Grange (vea [SHA75]).

Descripción del Algoritmo

El algoritmo es sencillo y se puede describir de la siguiente manera.
Suponga que el número de sitios en donde se pueden construir plantas potenciales es n y se requiere construir m o menos plantas entonces, para cada consumidor j seleccione de las n plantas potenciales que pueden cubrir su demanda, la planta que le suministre con el costo mínimo de transporte. Una vez hecho esto, se tendrá un número k de plantas abiertas.

Para cada planta abierta i, calcule su costo de eliminación; este costo de eliminación es la suma de las diferencias entre el costo menor de suministrar su demanda por alguna otra de las $k-1$ plantas abiertas y el costo de transporte a sus consumidores asignados. Si este costo es menor que el costo fijo f_i de la planta, entonces ciérrela y reasigne sus demandas conforme se hayan obtenido los costos menores.

Este proceso se sigue hasta que todas las plantas tengan un costo fijo menor que su costo de eliminación.

Una vez terminado este proceso se checa si $k \leq m$. Si esta condición se cumple pare con una solución subóptima. En caso contrario, seleccione la planta con el valor mínimo del costo de eliminación menos su costo fijo. Ciérrela y reasigne demanda.
Repita este proceso hasta que $k \leq m$

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria instancias del problema de localización de plantas. Para valores preestablecidos del número m de plantas y el número n 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).

La Tabla 7 reporta el tiempo promedio (en segundos) y los valores del número de plantas y de consumidores considerados. Para cada valor del número de plantas, se consideraron cuatro casos (en cuanto al número de consumidores).

Num. Plantas	HEURÍSTICO Num. Arcos / Tiempo(seg.)			
	5	10	15	20
5	0.01	0.01	0.02	0.02
	10	15	25	30
10	0.02	0.03	0.04	0.04
	15	25	35	40
15	0.04	0.05	0.07	0.09
	20	25	35	40
20	0.05	0.07	0.09	0.09
	25	30	35	40
25	0.07	0.10	0.12	0.13
	30	40	50	60
30	0.12	0.13	0.15	0.16
	35	40	50	60
35	0.13	0.16	0.19	0.18
	40	45	50	60
40	0.16	0.21	0.21	0.22

Tabla 7. Tiempos promedio de ejecución del algoritmo heurístico para el problema de localización de plantas.

3.3.4 Algoritmo de Recocido Simulado

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 que se conoce la solución.

Descripción del Algoritmo

En este caso el algoritmo de recocido simulado puede aplicarse de la siguiente manera:

- El espacio de soluciones $\mathcal{S}$, para el problema combinatorio, está representado por todos los vectores binarios $(y_1, y_2, ..., y_n)$ tal que $y_i = 0$ o 1 y $\sum_{i=1}^{n} y_i \leq m$. 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 de costo a minimizar, se escoge como:

$$f = \sum_i \sum_j c_{ij} x_{ij} + \sum_i f_i y_i \tag{12}$$

donde x_{ij} es la cantidad (en unidades) del producto transportado del sitio i al sitio j y f es el costo de la localización más transporte.

- Las nuevas soluciones se generan 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, tomado 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 costo 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.

El algoritmo que describe el procedimiento de recocido puede enunciarse como aparece abajo.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando de manera aleatoria instancias del problema de localización de plantas. Para valores preestablecidos del número m de plantas y el número n 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 de manda de consumidores, se tomó un número aleatorio con distribución uniforme entre (10-100).

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: Asigne valores 0 o 1 a y_i con $i \in \{1, 2, ..., n\}$ de manera arbitraria.
Obtenga el valor correspondiente de la función objetivo f.

Paso 3: Asigne el valor verdadero a la variable lógica CHANGE.

Paso 4: Si la variable CHANGE tiene valor falso, entonces pare.

Paso 5: Haga CHANGE igual a falso.

Paso 6: Repita los puntos 1-11 r veces.

1.- Seleccione aleatoriamente un número $u \in \{1, 2, ..., n\}$.
2.- Genere un número aleatorio x, uniformemente distribuido en $(0, 1)$.
3.- Si $x \geq p$ abra (o cierre) la planta u, en caso contrario intercámbiela por una cerrada (abierta).
4.- Evalue el valor Δf.
5.- Si $\Delta f < 0$ entonces vaya a 9.
6.- Sea $P(\Delta f) = \exp(-\Delta f/c)$.
7.- Genere un número aleatorio x, uniformemente distribuido en $(0, 1)$.
8.- Si $x \geq P(\Delta f)$ entonces vaya a 11.
9.- Acepte el cambio hecho y reactualice el valor de f.
10.- Si $\Delta f \neq 0$ entonces CHANGE $\leftarrow$ verdadero.
11.- Fin del paso 6.

Paso 7 Reduzca c e incremente p y r.

Paso 8 Vaya al paso 4.

La Tabla 8 reporta el tiempo promedio (en segundos) y los valores del número de plantas y de consumidores considerados. Para cada valor del número de plantas, se consideraron cuatro casos (en cuanto al número de consumidores).

Num. Plantas	RECOCIDO SIMULADO Num. Arcos / Tiempo(seg.)			
	5	10	15	20
5	0.01	0.78	0.75	0.83
	10	15	25	30
10	2.04	2.73	2.20	2.67
	15	25	35	40
15	4.82	5.82	9.15	9.55
	20	25	35	40
20	6.92	7.30	13.73	10.34
	25	30	35	40
25	12.85	10.83	12.46	12.58
	30	40	50	60
30	12.53	12.74	11.92	13.44
	35	40	50	60
35	18.66	26.87	27.80	29.86
	40	45	50	60
40	20.33	30.54	32.75	32.32

Tabla 8. Tiempos promedio de ejecución del algoritmo de recocido simulado para el problema de localización de plantas.

3.3.5 Algoritmo de Búsqueda Local

Uno de los puntos importantes del algoritmo de recocido simulado es el hecho de que cada que se ejecuta el **Paso 6** del mismo, no solamente acepta soluciones con menor costo sino que también se aceptan con cierta probabilidad soluciones con mayor costo. Esta situación cada vez es menos frecuente conforme el valor del parámetro de control c se decrementa. Sin embargo queda en duda el hecho de si el aceptar soluciones "malas" da algún beneficio. Con el objeto de poder comparar los resultados del algoritmo de recocido simulado con el algoritmo de búsqueda local correspondiente, también se implantó este último en la computadora.
Siguiendo la misma notación que la ya dada para el algoritmo de recocido simulado y tomando en cuenta que si en el algoritmo de recocido simulado se toma el parámetro de control c desde el principio cercano a cero, el algoritmo que nos queda es un algoritmo

de búsqueda local, ya que, la probabilidad de aceptar soluciones más malas que la actual prácticamente se reduce a cero. El algoritmo de búsqueda local se puede describir de la siguiente manera:

Algoritmo de Búsqueda Local 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: Asigne valores 0 o 1 a $y_i \in \{1, 2, ..., n\}$ de manera arbitraria.
Obtenga el valor correspondiente de la función objetivo f.
Paso 2: Asigne el valor verdadero a la variable lógica CHANGE.
Paso 3: Si la variable CHANGE tiene valor falso, entonces pare.
Paso 4: Haga CHANGE igual a falso.
Paso 5: Repita los puntos 1-8 $2 * n$ veces.
1.- Seleccione aleatoriamente un número $u \in \{1, 2, ..., n\}$.
2.- Genere un número aleatorio x, uniformemente distribuido en $(0, 1)$.
3.- Si $x \geq p$ abra (o cierre) la planta u, en caso contrario intercámbiela por una cerrada (abierta).
4.- Evalue el valor Δf.
5.- Si $\Delta f \geq 0$ entonces vaya a 9.
6.- Acepte el cambio hecho y reactualice el valor de f.
7.- Si $\Delta f \neq 0$ entonces CHANGE $\leftarrow$ verdadero.
8.- Fin del paso 5.
Paso 6 Incremente el valor de p.
Paso 7 Vaya al paso 3.

Experiencia Computacional
La prueba del algoritmo se llevó a cabo generando de manera aleatoria instancias del problema de localización de plantas. Para valores preestablecidos del número m de plantas y el número n 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 de manda de consumidores, se tomó un número aleatorio con distribución uniforme entre (10-100).

La Tabla 9 reporta el tiempo promedio (en segundos) y los valores del número de plantas y de consumidores considerados. Para cada valor del número de plantas, se consideraron cuatro casos (en cuanto al número de consumidores).

Num. Plantas	BÚSQUEDA LOCAL Num. Arcos / Tiempo(seg.)			
	5	10	15	20
5	0.00	0.07	0.07	0.08
	10	15	25	30
10	0.21	0.23	0.22	0.27
	15	25	35	40
15	0.42	0.58	0.93	0.95
	20	25	35	40
20	0.69	0.73	1.53	1.04
	25	30	35	40
25	1.15	1.18	1.58	1.66
	30	40	50	60
30	1.25	1.26	1.92	1.97
	35	40	50	60
35	1.17	2.77	2.78	2.96
	40	45	50	60
40	2.05	3.17	3.25	4.49

Tabla 9. Tiempos promedio de ejecución del algoritmo de búsqueda local para el problema de localización de plantas.

3.3.6 Análisis Comparativo de los Métodos

Como ya se mencionó, 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. Los resultados aparecen en las Tablas 6, 7, 8 y 9.

Como ya se mencionó en la sección anterior, 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. Para el análisis realizado en este documento, se tomó 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 del 0.1 (paso 7 del algoritmo). El número r inicial que se tomó en cada caso fué $r = n$ y los incrementos fueron de 0.05 (paso 7 del algoritmo).

Para el calculó la eficiencia de cada uno de los algoritmos con respecto al valor óptimo de la función objetivo se tomó como referencia la ecuación (9)

Los cálculos que aparecen en la Tabla 10 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 la Tabla 10, 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 revelen 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 obtienen en todos los casos la solución óptima.

Los programas computacionales y las corridas de los mismos, se realizaron en una computadora IBM-PS/70 de 25 mhz con procesador 80386 y sin coprocesador matemático.

		ALGORITMO (eficiencia)			
m	n	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
	25	1.00	0.8920	0.9957	0.8671
	30	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
	35	1.00	0.8690	0.9963	0.8651
	40	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
	35	1.00	0.7982	0.9863	0.8126
	40	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

Tabla 10. Comparación de los algoritmos que resuelven el problema de localización de plantas con respecto a su eficiencia.

3.4 EL PROBLEMA DE ASIGNACIÓN CUADRÁTICO

El objetivo fundamental de este sección es presentar la bondad de la técnica de recocido simulado para el problema de asignación cuadrático, así como los lineamientos a seguir para implantarlo en la computadora. Para tal propósito, se presentan en esta sección tres métodos alternativos de solución del problema: un método exacto, un método heurístico y un método de búsqueda local. El método exacto y el heurístico que se seleccionaron, se reportan en la literatura como los más eficientes, vea [BUR90]. El estudio con el método de búsqueda local está enfocado a la investigación de si es bueno aceptar una solución más mala, a partir de la solución que actualmente se tenga.

El análisis comparativo de los cuatro algoritmos se hizo en cuanto a tiempo de ejecución, así como a su cercanía con el valor óptimo de la función objetivo, cuando se conoce tal solución. Adicionalmente se prueba para las instancias de Nugent. Para cada algoritmo se reporta la experiencia computacional y en la última subsección se hace un análisis comparativo de los cuatro métodos.

3.4.1 Descripción

Una variante importante del problema de localización clásico (ya descrito en la sección 3.3) es cuando el costo asociado por colocar una planta en un determinado lugar depende no únicamente de las distancias entre plantas y de las demandas respectivas, sino también de la interacción con otras plantas.

Una formulación matemática de esta problematica fué inicialmente planteada por Koopmans y Beckmann ([KOO57]) y consiste en minimizar

$$f(\pi) = \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)} \tag{13}$$

donde π es una permutación del conjunto $N = \{1, 2, ..., n\}$ y c_{ik}, a_{ik}, b_{ik} para $i, k = 1, ..., n$, son números reales. Equivalentemente, se desea encontrar una permutación π del conjunto N que mínimice el valor en (13).

Una forma sencilla de interpretar el problema planteado es suponer que existen n sitios disponibles y se van ha construir n edificios en estos lugares. Sea a_{ik} la distancia entre el sitio i y el sitio k y suponga que las cantidades b_{jl} denoten el número de gentes por semana que viajarán entre el edificio j y l. El problema es asignar los sitios de

construcción de manera que la distancia recorrida se minimice. Cada asignación puede describirse matemáticamente como una permutación π del conjunto $N = \{1, 2, ..., n\}$. Mientras que el producto $a_{ik}b_{\pi(i)\pi(k)}$ describe la distancia semanal recorrida por la gente que viaja entre los edificios $j = \pi(i)$ y $l = \pi(k)$, si la construcción j se realiza en el sitio i y la construcción l se realiza en el sitio k.

Considere un ejemplo número co donde se desea localizar cuatro edificios A, B, C y D en cuatro sitios a, b, c, d. Suponga que las distancias (en metros) entre los cuatro edificios están dadas por la matriz de distancias.

$$(a_{ik}) = \begin{bmatrix} 0 & 340 & 320 & 400 \\ 340 & 0 & 360 & 200 \\ 320 & 360 & 0 & 180 \\ 400 & 200 & 180 & 0 \end{bmatrix}$$

El edificio A es la biblioteca, el edificio B son aulas y cubículos, el edificio C son oficinas administrativas y el edificio D son dormitorios. Las "conecciones" entre estos edificios se describen por medio de la matriz de conección (b_{jl}), donde b_{jl} es una medida para el número de personas y la frecuencia con la que viajan del edificio j al edificio l.

$$(b_{ik}) = \begin{bmatrix} 0 & 80 & 40 & 30 \\ 80 & 0 & 30 & 20 \\ 40 & 30 & 0 & 10 \\ 30 & 20 & 10 & 0 \end{bmatrix}$$

Observe que si se construye el edificio A en el sitio a, el edificio B en el sitio b, el edificio C en el sitio c y el edificio D en el sitio d, entonces se obtendría el siguiente valor de la función objetivo:

$$z = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik}b_{ik} = 137,200.$$

que corresponde a la permutación $\pi(i) = i$ para $i = 1, 2, 3, 4$. Ahora suponga que A se construye en el sitio d, B en b, C en c y D en a, se tiene la permutación $\pi(1) = 4$, $\pi(2) = 2$, $\pi(3) = 3$, $\pi(4) = 1$ y el valor de la función objetivo es ahora.

$$z = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik}b_{\pi(i)\pi(k)} = 112,000.$$

Otra interpretación del problema planteado consiste en suponer que n módulos se han colocado en un tablero y se han conectado por medio de "alambres". Los módulos deben colocarse de manera que la longitud total del alambre que los conecta sea mínima.

Denote la distancia entre las posiciones i y k en el tablero por a_{ik}. Sea b_{jl} el número de conecciones entre el módulo j y el módulo l. Entonces una permutación π del conjunto $N = \{1, 2, .., n\}$ asigna una posición en el tablero a todo módulo. Si j se situa en la posición i y l está situado en la posición k, el producto $a_{ik}b_{\pi(i)\pi(k)}$ es la longitud del alambre entre los módulos $j = \pi(i)$ y $l = \pi(k)$. Por lo tanto la longitud total del alambrado es

$$\sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik}b_{\pi(i)\pi(k)},$$

y el problema se reduce a

$$\underset{\pi}{\text{minimizar}}\ z = \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik}b_{\pi(i)\pi(k)}$$

El problema anterior, se conoce comúnmente en la literatura como *el problema de asignación cuadrático* (QAP). Una formulación similar del QAP es aplicable para el diseño de paneles de control y teclados de máquinas. Existen muchas otras aplicaciones del QAP por ejemplo:

- Problemas de ubicación de edificios, planeación de hospitales.
- En deportes, en problemas de calendarización.
- En sistemas de información, un orden óptimo para interrelacionar datos en una cinta magnética.
- En química, para analizar reacciones químicas para componentes orgánicas.
- En balancear turbinas para autos de carreras.
- En arqueología, modelos para ordenar datos arqueológicos.

3.4.2 Algoritmo Exacto

Se conoce que el problema de asignación cuadrático es NP-completo. Por lo tanto, hasta ahora, únicamente se conocen algoritmos de enumeración implícita que resuelven el problema de manera óptima. En esta subsección se describe un algoritmo para encontrar

las soluciones óptimas que según experiencia computacional de Burkard (vea [BUR90]) es de los algoritmos más eficientes que se conocen para resolver el problema de manera exacta. Este método se conoce como el método de la perturbación.

Descripción del algoritmo

El algoritmo de ramificación y acotamiento se puede describir como sigue: Sea $M \subseteq N$ donde $N = \{1, 2, ..., n\}$. Denote por S_n el conjunto de todas las permutaciones π de N. La restricción de π a M se llama *permutación parcial* (π, M). Además sea $\phi_M = \{\pi \in S_n | \pi_M = (\pi, M)\}$ denote el conjunto de todas las permutaciones que son extensiones de la permutación parcial (π, M). Se define $\pi(M) = \{\pi(i) | i \in M\}$.

Sea una instancia de un problema de asignación cuadrático con matrices A y B y una permutación parcial (π, M) dada. Una permutación $\pi \in \phi_M$ tiene la función objetivo siguiente:

$$z(\pi) = \sum_{i,p \in M} a_{ip} b_{\pi(i)\pi(p)} + \sum_{i \in M \; p \notin M} [a_{ip} b_{\pi(i)\pi(p)} + a_{pi} b_{\pi(p)\pi(i)}] + \sum_{i,p \notin M} a_{ip} b_{\pi(i)\pi(p)}. \qquad (14)$$

la primera suma en (14) puede calcularse de manera exacta conociendo la permutación parcial (π, M). Para la segunda y tercera sumatoria en (14) se puede encontrar una cota inferior resolviendo un problema de asignación lineal con una matriz de costos. $C = (c_{ij})$ $(i \notin M, j \notin \pi(M))$. Sin embargo la solución óptima de este problema de asignación lineal da sugerencias de como debe hacerse una asignación simple $(i \rightarrow j)$ en el paso siguiente. Las coeficientes c_{ij} se escogen como cotas inferiores para la contribución de la asignación simple i a j en el valor de la función objetivo $z(\pi)$. Estas cotas inferiores c_{ij} pueden determinarse de la siguiente manera.
En principio

$$a_{ii} b_{jj} + \sum_{p \in M} (a_{ip} b_{j\pi(p)} + a_{pi} b_{\pi(p)j}$$

es la contribución a c_{ij}, que depende de la permutación parcial (π, M). Este valor puede mejorarse por un paso de reducción.

Reducción de la matriz A:

$$\tilde{a}_i = \min_{p \notin M, p \neq i} a_{ip} \qquad (i \notin M);$$

$$\begin{aligned}
\bar{a}_{ip} &= a_{ip} - \bar{a}_i \qquad (i, p \notin M; i \neq p);\\
\tilde{a}_p &= \min_{i \notin M, i \neq p} \bar{a}_{ip} \qquad (p \notin M);\\
a_{ip} &= \bar{a}_{ip} - \tilde{a}_p \qquad (i, p \notin M; i \neq p).
\end{aligned}$$

Es fácil mostrar que

$$\bar{a}_i \sum_{q \notin \pi(M) q \neq j} b_{jq} + \tilde{a}_i \sum_{q \notin \pi(M) q \neq j} b_{qj} \quad (i \notin M, j \notin \pi(M))$$

es la contribución adicional que aparece en c_{ij}, si la asignación simple $(i \rightarrow j)$ se hace. En forma análoga la matriz B puede reducirse y esto permite un mejoramiento de la cota. La cota puede aún mejorarse. La idea es que se puede obtener una cota inferior para todos los productos escalares $\sum_{i=1}^{n} a_i b_{\pi(i)}$ de dos vectores **a** y **b** no negativos, si el orden de los elementos del vector **a** se toma en forma creciente y el de los elementos del vector **b** se toma en forma decreciente y formamos el producto escalar $\langle a, b \rangle$ de estos dos vectores ordenados.

Si se aplica lo anterior para reducir los elementos a_{ip} $(i, p \notin M; i \neq p)$ y b_{jq} $(j, q \notin \pi(M); j \neq q)$ y se definen los vectores.

$$\begin{aligned}
a_i &= (a_{ip} | p \notin M, p \neq i) \qquad i \notin M,\\
b_j &= (b_{jq} | q \notin \pi(M), q \neq j) \qquad j \notin \pi(M)
\end{aligned}$$

Entonces $\langle a, b \rangle$ es una contribución adicional a la cota c_{ij} (llamada la "cota de Gilmore").

Ahora se resuelve un problema de asignación lineal con matriz de costos $C = (c_{ij})$ $(i \notin M, j \notin \pi(M))$. La experiencia computacional (vea [BUR90]) muestra que estos problemas deben resolverse en forma óptima. Sea z_L el valor de la función objetivo y $(\bar{c}_{ij})$ los costos reducidos en la solución óptima del problema de asignación lineal con coeficientes de costo c_{ij}. Entonces una cota inferior para el valor de la función objetivo en (14) de las permutaciones $\pi \in \phi_M$ es

$$z(\phi_M) = \sum_{i,p \in M} a_{ip} b_{\pi(i)\pi(p)} + z_L.$$

La ramificación en el método de la perturbación viene dada de la siguiente forma: Se inicializa con la permutación parcial, en la que M es el conjunto vacío. Sea z una cota

superior para el valor óptimo de la función objetivo. Si $z(\phi_M) < z$ se escoge una pareja de índices (i,j) $i \notin M$, $j \notin \pi(M)$ por las reglas de costos alternativos, reemplace M por $M \cup \{i\}$ y extienda la permutación parcial por la asignación simple $(i \rightarrow j)$. Para esta nueva permutación parcial $(\bar{\pi}, \bar{M})$ se calcula una nueva cota $z(\phi_{\bar{M}})$. Los coeficientes c_{ij} nuevos pueden obtenerse a partir de la solución óptima del último problema de asignación lineal. Si $|M| = n-2$, el valor de las dos permutaciones correspondientes se calcula de manera exacta y eventualmente la cota z se mejora.

Si $z(\phi_{\bar{M}}) \geq z$, se regresa a la última permutación parcial encontrada (π, M) con $z(\phi_{\bar{M}}) < z$. La asignación simple $(i \rightarrow j)$, seleccionada en la última transición de (π, M) a su sucesor, se elimina y la correspondiente entrada (i,j) de la matriz $\bar{C} = (\bar{c}_{ij})$ se bloquea. Si el elemento bloqueado es el primer elemento bloqueado en la matriz $\bar{C} = (\bar{c}_{ij})$ correspondiente a (π, M), una nueva asignación simple se escoge en el mismo renglón o columna. Si el renglón i (o columna j) contiene más elementos bloqueados, la nueva asignación simple se toma sobre este renglón índice (columna índice). Si esto no es posible se regresa a la permutación parcial predecesora de la permutación parcial (π, M).

La solución óptima tiene el valor z cuando el conjunto M se hace vacío por segunda ocasión.

Las reglas de costos alternativas usados son las siguientes. Sea π^* una solución óptima del problema de asignación lineal con costos c_{ij} y sean $(\bar{c}_{ij})$ denoten los costos reducidos en el tableau óptimo. Calcule para todo $(i, \pi^*(i))$ las expresiones.

$$\alpha_i = \min_{j, j \neq \pi^*(i)} \bar{c}_{ij}; \qquad \beta_i = \min_{p, p \neq i} \bar{c}_{p\pi^*(i)};$$

El valor $\alpha_i + \beta_i$ constituye una cota inferior para el incremento de la cota z_L, si la asignación simple $(i \rightarrow \pi^*(i))$ no se realiza. Se puede escoger la pareja $(i, \pi^*(i))$ para la siguiente asignación simple que tiene costos alternativos maximal. Por lo tanto se determina el índice i que resuelve el problema:

$$\max_{i \notin M} (\alpha_i + \beta_i)$$

y a partir de aquí se define la nueva permutación parcial $(\bar{\pi}, \bar{M})$:

$$\bar{M} = M \cup \{i\}$$
$$\bar{\pi}(i) = \pi^*(i)$$
$$\bar{\pi}(p) = \pi(p) \quad \text{para toda } p \in M.$$

Experiencia Computacional
La prueba del algoritmo se llevó a cabo generando instancias del problema de asignación cuadrático de manera aleatoria. Para la matriz A se generaron números aleatorios entre 0 - 100 y para la matriz B se generaron números aleatorios entre 0 - 50. Para cada valor de $n = 5, 6, 7, 8, 12$ se generaron 100 instancias. La Tabla 11 reporta el tiempo promedio (en segundos). Cabe mencionar que para $n = 13$ el programa tardó más de 6 horas en una instancia generada aleatoriamente, por lo que se decidió cortar la tabla hasta el valor de $n = 12$.

n	ALGORITMO EXACTO tiempo (seg.)
5	0.08
6	0.25
7	1.03
8	3.98
10	73.45
12	1713.99

Tabla 11.

n	ALGORITMO EXACTO tiempo (seg.)
5	0.06
6	0.22
7	0.61
8	2.70
12	520.31

Tabla 12. Problemas de Nugent.

Adicionalmente, para cada uno de los valores anteriores de n se corrieron los problemas de prueba de Nugent (vea [NUG68]). Los resultados de estas corridas aparecen en la Tabla 12.

3.4.3 Algoritmo Heurístico

Como ya se ha mencionado, el problema de asignación cuadrático es NP-difícil y desde el punto de vista práctico, es imposible resolver instancias "grandes" del problema. Existen instancias con $N = 20$ para las que hasta la fecha no se conoce la solución óptima (vea [NUG68] y [BUR90]), por lo que cuando se requiere resolver problemas de mayor tamaño, se debe recurrir a algoritmos heurísticos que no necesariamente obtienen la solución óptima del problema, pero dan una "buena" solución.

En esta sección se tratará nuevamente el problema de asignación cuadrático dado por

$$\min_{\pi \in S_n} \sum_{i=1}^{n} \sum_{p=1}^{n} a_{ip} b_{\pi(i)\pi(p)} \tag{15}$$

puesto que el problema (15) puede resolverse de manera óptima únicamente para instancias pequeñas. Burkard y Stratmann ([BUR78] y [BUR90]) proponen un algoritmo para encontrar soluciones subóptimas al problema que denominan el método de incrementar grado de libertad. Este algoritmo se describe a continuación.

Usando la notación de la sección anterior, se comienza con el conjunto vacío M y se escoge de manera arbitraria un índice $i \notin M$ y se asigna este índice i a un índice $j \in \{1, 2, ..., n\}$, tal que $z(j)$ sea minimal. Para $j \notin \pi(M)$ se define $z(j)$ por

$$z(j) = a_{ii} b_{jj} + \sum_{p \in M} [a_{ip} b_{j\pi(p)} + a_{pi} b_{\pi(p)j}] + \sum_{p,q \in M} a_{pq} b_{\pi(p)\pi(q)} \tag{16}$$

y para $j = \pi(i_0), i_0 \in M$; $k \notin \pi(M)$ se define $z(j)$ por

$$\begin{aligned} z(j) \; &= a_{ii} b_{jj} + a_{i_0 i_0} b_{kk} + a_{i i_0} b_{jk} + a_{i_0 i} b_{kj} \\ &+ \textstyle\sum_{p \in M \setminus \{i_0\}} [a_{ip} b_{j\pi(p)} + a_{pi} b_{\pi(p)j} + a_{i_0 p} b_{k\pi(p)} + a_{p i_0} b_{\pi(p)k}] \\ &+ \textstyle\sum_{p,q \notin M \setminus \{i_0\}} a_{pq} b_{\pi(p)\pi(q)} \end{aligned} \tag{17}$$

Se define $m = |M| + 1$. Para determinar para $i \notin M$ un índice $j \in \{1, ..., n\}$ con valor minimal $z(j)$, se tienen que evaluar $m(n - m + 1)$ sumas de la forma (16) o (17). Como $m(n - m + 1)$ se incrementa con $m = 1, 2, ..., \left[\frac{n}{2}\right]$ este método se llama el método de incrementar grados de libertad.

La permutación final que se obtiene por este método depende de la sucesión de índices i que se escojan. Por lo tanto, si se realiza este procedimiento con sucesiones diferentes de índices i (es decir, diferentes permutaciones de la sucesión básica $i = 1, 2, ..., n$), se obtendrán diferentes soluciones. Se puede tomar ventaja de esta observación y ejecutar el método para selecciones diferentes de los índices libres $i \notin M$.

En el programa, la subrutina de asignación se repite REP veces comenzando con permutaciones generadas aleatoriamente para los índices i.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando instancias del problema de asignación cuadrático de manera aleatoria. Para la matriz A se generaron números aleatorios entre

0 - 100 y para la matriz B se generaron números aleatorios entre 0 - 50. Para cada valor de $n = 5, 6, 7, 8, 10, 12, 15, 20, 25, 30$ se generaron 100 instancias de manera aleatoria. La Tabla 13 reporta el tiempo promedio (en segundos).

n	ALGORITMO HEURÍSTICO tiempo (seg.)
5	0.08
6	0.16
7	1.28
8	3.44
10	1.01
12	2.03
15	4.75
20	14.55
25	35.02
30	71.80

Tabla 13.

n	ALGORITMO HEURÍSTICO Mejor Solución Conocida	Valor	Tiempo (seg.)
5	50 (óptima)	50	0.11
6	86 (óptima)	86	0.17
7	148 (óptima)	148	0.28
8	214 (óptima)	226	0.44
12	578 (óptima)	612	2.03
15	1150 (óptima)	1202	4.78
20	2570	2774	14.56
30	6124	6630	71.84

Tabla 14. Problemas de Nugent.

Adicionalmente, para $n = 5, 6, 7, 8, 12, 15, 20, 30$ se corrieron los problemas de prueba de Nugent (vea [NUG68]). Los resultados de estas corridas aparecen en la Tabla 14.

3.4.4 Algoritmo de Recocido Simulado

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 que se conoce la solución, así como para aquellas instancias que no se conoce la solución óptima pero se conoce una mejor solución.

Para aplicar el algoritmo de recocido simulado se necesitan definir ciertos puntos que a continuación se describen.

Descripción del algoritmo

Como ya se ha mencionado, el espacio de soluciones consiste de todas las permutaciones de n elementos. Para cada permutación π, se define la función de costo como

$$f = \sum_{i=1}^{n} \sum_{p=1}^{n} a_{ip} b_{\pi(i)\pi(p)} \tag{18}$$

La generación de nuevas soluciones se hace a partir de tomar permutaciones que difieran de la actual en únicamente dos cambios. Se comienza con una permutación π y se seleccionan dos índices i y k y a partir de estos índices se define la nueva permutación $\bar{\pi}$ por

$$\bar{\pi}(i) \equiv \pi(k), \qquad \bar{\pi}(k) \equiv \pi(i) \tag{19}$$

$$\bar{\pi}(l) \equiv \pi(l), \quad l = 1, 2, ...n, \; l \neq i, k.$$

La diferencia en la función de costo entre la solución actual y la nueva solución generada a partir del cambio i, k puede calcularse de la siguiente expresión.

$$\Delta f = \sum_{l=1, l\neq i,k} [(a_{il} - a_{kl})(b_{\pi(k)\pi(l)} - b_{\pi(i)\pi(l)}) + (a_{li} - a_{lk})(b_{\pi(l)\pi(k)} - b_{\pi(l)\pi(i)})] \tag{20}$$

$$+(a_{ii} - a_{kk})(b_{\pi(k)\pi(k)} - b_{\pi(i)\pi(i)})$$
$$+(a_{ik} - a_{ki})(b_{\pi(k)\pi(i)} - b_{\pi(i)\pi(k)})$$

donde Δf denota la diferencia en el costo entre la solución propuesta menos la solución actual al hacer el intercambio. Si $\Delta f < 0$, entonces $\bar{\pi}$ tiene un valor en la función objetivo menor que π. Usando estas observaciones se puede establecer el algoritmo de recocido simulado.

Experiencia Computacional

La prueba del algoritmo se llevó a cabo generando instancias del problema de asignación cuadrático de manera aleatoria. Para la matriz A se generaron números aleatorios entre 0 - 100 y para la matriz B se generaron números aleatorios entre 0 - 50. Para cada valor de $n = 5, 6, 7, 8, 10, 12, 15, 20, 25, 30$ se generaron 100 instancias de manera aleatoria. La Tabla 15 reporta el tiempo promedio (en segundos).

Algoritmo de Recocido Simulado para el Problema de Asignación Cuadrático

Propósito: Obtener una permutación π que minimice el valor de la función en (13).

Descripción:
Paso 1: Tome c un valor grande apropiado.
Paso 2: Obtenga una permutación aleatoria π del conjunto $\{1, 2, ..., n\}$.
Obtenga el valor correspondiente de la función objetivo $f(\pi)$.
Paso 3: Asigne el valor verdadero a la variable lógica CHANGE.
Paso 4: Si la variable CHANGE tiene valor falso, entonces pare.
Paso 5: Haga CHANGE igual a falso.
Paso 6: Repita los puntos 1-9 r veces.
1.- Tome una permutación arbitraria de dos cambios $\bar{\pi}$
2.- Evalue el valor Δf de la expresión (20)
3.- Si $\Delta f < 0$ entonces vaya a 7.
4.- Sea $P(\Delta f) = \exp(-\Delta f/c)$.
5.- Genere un número aleatorio x, uniformemente distribuido en $(0, 1)$.
6.- Si $x \geq P(\Delta f)$ entonces vaya a 9.
7.- Acepte la nueva permutación $\pi \leftarrow \bar{\pi}$, $f(\pi) \leftarrow f(\pi) + \Delta f$.
8.- Si $\Delta f \neq 0$ entonces CHANGE $\leftarrow$ verdadero.
9.- Fin del paso 6.
Paso 7 Reduzca c e incremente r.
Paso 8 Vaya al paso 4.

Adicionalmente, para $n = 5, 6, 7, 8, 12, 15, 20, 30$ se corrieron los problemas de prueba de Nugent (vea [NUG68]). Los resultados de estas corridas aparecen en la Tabla 16.

n	ALGORITMO REC. SIM. tiempo (seg.)
5	1.73
6	2.25
7	2.98
8	3.79
10	6.40
12	12.73
15	15.21
20	20.04
25	28.62
30	43.65

Tabla 15.

ALGORITMO RECOCIDO SIMULADO			
n	Mejor Solución Conocida	Valor	Tiempo (seg.)
5	50 (óptima)	50	0.27
6	86 (óptima)	86	1.27
7	148 (óptima)	148	1.58
8	214 (óptima)	214	1.76
12	578 (óptima)	578	2.86
15	1150 (óptima)	1150	14.17
20	2570	2570	36.80
30	6124	6150	76.13

Tabla 16. Problemas de Nugent.

3.4.5 Algoritmo de Búsqueda Local

Uno de los puntos importantes del algoritmo de recocido simulado es el hecho de que cada que se ejecuta el **Paso 6** del mismo, no solamente acepta permutaciones π para las que se tenga un mejor valor objetivo, sino que también se aceptan con cierta probabilidad permutaciones cuyo valor objetivo es más malo. Con el objeto de poder comparar los resultados del algoritmo de recocido simulado con el algoritmo de búsqueda local correspondiente, también se implantó este último en la computadora.

Siguiendo la misma notación que la ya dada para el algoritmo de recocido simulado, se puede describir el algoritmo de búsqueda local como aparece abajo.

Experiencia Computacional
La prueba del algoritmo se llevó a cabo generando instancias del problema de asignación cuadrático de manera aleatoria. Para la matriz A se generaron números aleatorios entre 0 - 100 y para la matriz B se generaron números aleatorios entre 0 - 50. Para cada valor de $n = 5, 6, 7, 8, 10, 12, 15, 20, 25, 30$ se generaron 100 instancias de manera alcatoria. La Tabla 17 reporta el tiempo promedio (en segundos).

Algoritmo de Búsqueda Local para el Problema de Asignación Cuadrático

Propósito: Obtener una permutación π que minimice el valor de la función en (13).

Descripción:

Paso 1: Obtenga una permutación aleatoria π del conjunto $\{1, 2, ..., n\}$.
Obtenga el valor correspondiente de la función objetivo $f(\pi)$.
Paso 2: Asigne el valor verdadero a la variable lógica CHANGE.
Paso 3: Si la variable CHANGE tiene valor falso, entonces pare.
Paso 4: Haga CHANGE igual a falso.
Paso 5: Repita los puntos 1-6 $n \times (n-1)$ veces.
1.- Tome una permutación arbitraria de dos cambios $\bar{\pi}$
2.- Evalue el valor Δf de la expresión (20)
3.- Si $\Delta f \geq 0$ entonces vaya a 6.
4.- Acepte la nueva permutación $\pi \leftarrow \bar{\pi}$, $f(\pi) \leftarrow f(\pi) + \Delta f$.
5.- CHANGE $\leftarrow$ verdadero.
6.- Fin del paso 5.
Paso 6 Vaya al paso 3.

Adicionalmente, para $n = 5, 6, 7, 8, 12, 15, 20, 30$ se corrieron los problemas de prueba de Nugent (vea [NUG68]). Los resultados de estas corridas aparecen en la Tabla 18.

3.4.6 Análisis Comparativo de los Métodos

Como ya se mencionó, se generaron en forma aleatoria instancias del problema de asignación cuadrático. Para cada una de estas instancias generadas, se corrieron los cuatro algoritmos y se calculó el tiempo promedio de solución. Los resultados aparecen en las Tablas 11, 13, 15 y 17 respectivamente y aparecen para los cuatro algoritmos en la Tabla 19.

n	ALGORITMO BUS. LOC. tiempo (seg.)
5	0.13
6	0.21
7	0.29
8	0.43
10	0.79
12	1.07
15	2.40
20	5.64
25	11.11
30	20.96

Tabla 17.

ALGORITMO BÚSQUEDA LOCAL			
n	Mejor Solución Conocida	Valor	Tiempo (seg.)
5	50 (óptima)	50	0.11
6	86 (óptima)	86	0.22
7	148 (óptima)	148	0.28
8	214 (óptima)	214	0.33
12	578 (óptima)	592	1.32
15	1150 (óptima)	1178	2.41
20	2570	2644	5.27
30	6124	6354	15.81

Tabla 18. Problemas de Nugent.

Cabe mencionar que 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 incremente el parámetro r. Para el análisis realizado en este documento, se tomó un valor inicial de c de tal manera que la tasa de aceptación de las permutaciones propuestas fuera mayor o igual a 0.8 y los decrementos fueran del 0.1 (paso 7 del algoritmo). El número r inicial que se tomó en cada caso fué $r = 2 \times n$ y los incrementos fueron de 0.05 (paso 7 del algoritmo).

También se calculó la eficiencia de cada uno de los algoritmos por medio de la ecuación (9)

La ecuación (9) se usó para los valores de $n = 5, 6, 7, 8, 10, 12$. Para los valores mayores de n, como no se puede calcular la solución óptima, la eficiencia se calculó tomando como base el valor z_{rec}, esto es, el valor objetivo que da el algoritmo de recocido simulado y se usó la ecuación equivalente:

$$eficiencia = 1 - \frac{z - z_{rec}}{z_{rec}} \qquad (21)$$

Los calculos que aparecen en la Tabla 19 son con base en los promedios obtenidos en las 100 instancias resultas para cada valor de n de los valores en la función objetivo de la solución correspondiente.

En la Tabla 19, puede observarse que las eficiencias del algoritmo de recocido simulado son las más elevadas y los tiempos de ejecución del algoritmo de recocido simulado, crece de manera razonable; incluso, para instancias pequeñas se obtienen tiempos de ejecución más grandes que para el algoritmo heurístico. Esta situación cambia para valores de n grandes.

Otra característica importante del algoritmo de recocido simulado es que las variaciones en las soluciones obtenidas al resolver una misma instancia con soluciones iniciales diferentes, es mucho menor que las variaciones en las soluciones que dan los otros dos algoritmos, esto es, la solución del algoritmo de recocido simulado es menos dependiente de la solución inicial proporcionada. En cambio para el algoritmo heurístico y más aún, para el algoritmo de búsqueda local, las soluciones dependen en gran medida de la solución con que se inicialice. Este comportamiento se puede observar en la Tabla 20 en donde se muestran los resultados al ejecutar cada uno de los algoritmos 100 veces con soluciones iniciales generadas de manera aleatoria para cada uno de los problemas de Nugent.

	ALGORITMO							
	EXACTO		HEURISTICO		REC. SIM.		BUS. LOC.	
n	eficien.	tiempo (seg.)	eficien.	tiempo (seg.)	eficien.	tiempo (seg.)	eficien.	tiempo (seg.)
5	1.00	0.08	0.9891	0.08	0.9999	1.73	0.9747	0.13
6	1.00	0.25	0.9803	0.16	0.9982	2.25	0.9666	0.21
7	1.00	1.03	0.9767	0.28	0.9936	2.98	0.9676	0.29
8	1.00	3.98	0.9737	0.44	0.9899	3.79	0.9630	0.43
10	1.00	73.45	0.9684	1.01	0.9905	6.40	0.9629	0.79
12	1.00	1713.99	0.9674	2.03	0.9940	12.73	0.9512	1.07
15	—	—	0.9693	4.75	1.0000	15.21	0.9640	2.40
20	—	—	0.9512	14.55	1.0000	20.04	0.9518	5.64
25	—	—	0.9527	35.02	1.0000	28.62	0.9551	11.11
30	—	—	0.9543	71.80	1.0000	43.65	0.9580	20.96

Tabla 19. Comparación de los algoritmos para resolver el QAP con respecto al tiempo de ejecución y a su eficiencia.

	HEURÍSTICO			REC. SIM.			BUS. LOC.		
n	Mejor	Media	Peor	Mejor	Media	Peor	Mejor	Media	Peor
5	50	51.6	58	50	50.4	58	50	55.2	62
6	86	88.5	94	86	86.3	92	86	90.2	94
7	148	153.9	164	148	148.7	156	148	154.5	172
8	214	228	252	214	215.3	222	214	224.5	248
12	578	620.3	662	578	600.2	640	578	610.8	650
15	1188	1231	1294	1150	1172	1214	1150	1204	1290
20	2692	2749	2794	2570	2657	2728	2604	2680	2742
30	6488	6636	6766	6124	6333	6442	6242	6364	6496

Tabla 20. Comparación entre los algoritmos heurístico, recocido simulado y búsqueda local con respecto a la dependencia de la solución inicial

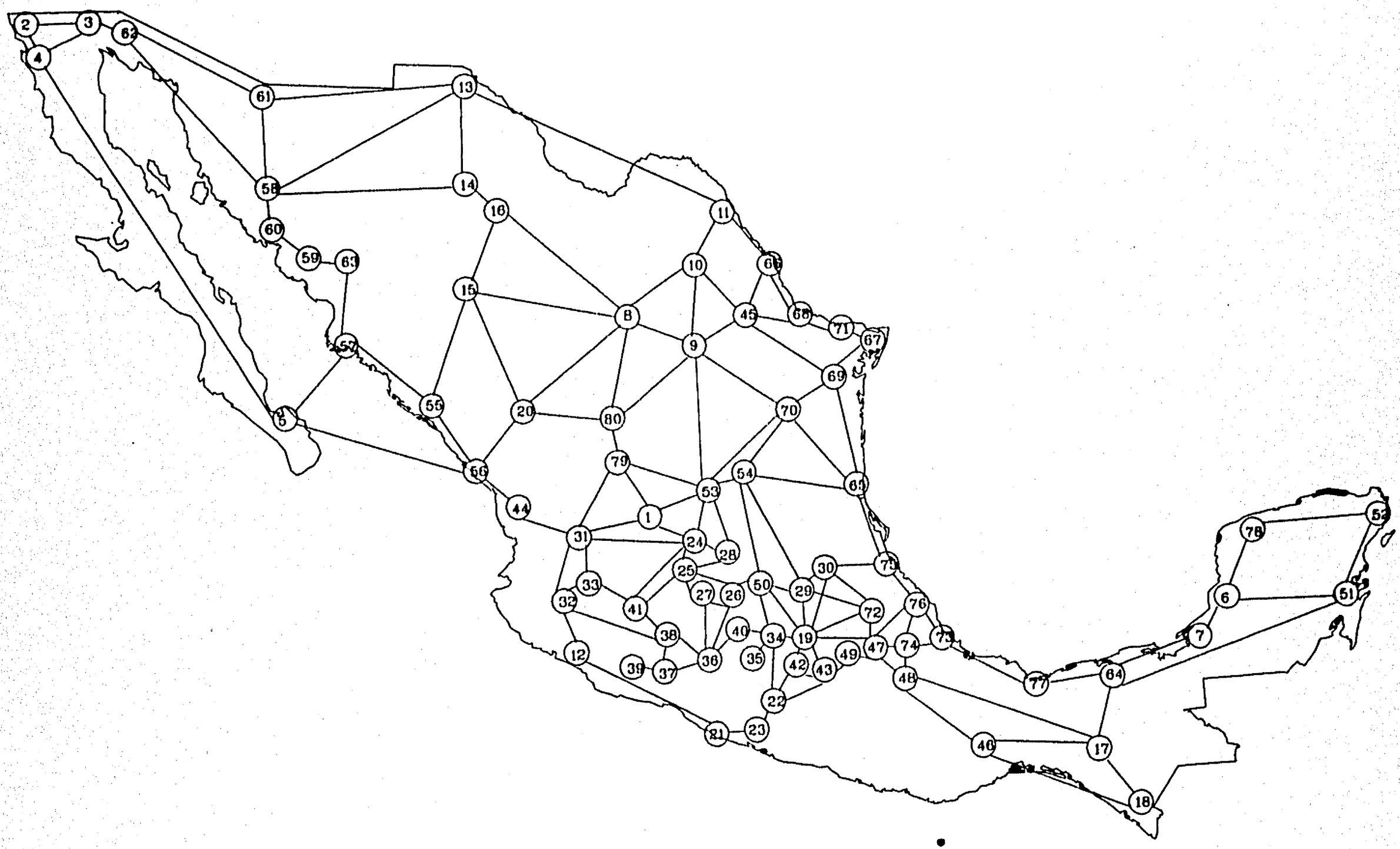
Las pruebas computacionales con el problema de asignación cuadrático, revelan un excelente comportamiento del algoritmo. Las soluciones subóptimas difieren en 1-2% de la mejor solución conocida. Además si se ejecuta el procedimiento varias veces con diferentes valores iniciales, se obtienen en todos los casos la solución óptima o las mejores soluciones que se conocen (según sea el caso). Además de que la implantación del algoritmo en la computadora, no requiere de gran capacidad de memoria y permite un tiempo razonable para obtener buenas soluciones en las computadoras personales.

Finalmente se hará mención que todos los programas computacionales y las corridas de los mismos, se realizaron en una computadora IBM-PS/70 de 25 mhz con procesador 80386 y sin coprocesador matemático.

3.5 UN EJEMPLO DE APLICACIÓN

Considere un problema real de gran escala perteneciente a los denominados problemas de Localización de Plantas (Sección 3.3). Específicamente, el problema consiste en la localización de plantas productoras de tortilla empacada y tiene la siguiente estructura: se consideran 80 centros de demanda en la República Mexicana (Vea Figura 2) que corresponden a 85 ciudades con más de 50,000 habitantes que requieren del producto. (Vea Tabla 21, en donde se especifica las ciudades y las demandas para cada uno de los centros.). Se tiene que satisfacer la demanda, para lo anterior se requiere construir plantas de tortilla empacada en algunas zonas. El costo por construir una planta en

Figura 2. Red Asociada al Problema de Localizacion de Plantas



un sitio, depende del tamaño de la planta (para propósitos de las corridas del modelo se consideraron tres tamaños: 2 toneladas diarias, 5.34 toneladas diarias, 16 toneladas diarias). Además se consideró que el costo de los insumos (harina de maíz) depende de la distancia de la planta a la fábrica de harina más cercana a la misma (Tablas 22 y 23).

El problema es determinar un conjunto óptimo de sitios de localización de las plantas, número de plantas y sus respectivas capacidades; así como la zona de influencia de las mismas. Las zonas de influencia se reflejan en un plan óptimo de transporte en el horizonte de planeación (a 10 años).

Se desea minimizar el costo total de construcción anualizado de las plantas y el costo total de transporte de la tortilla de maíz empacada.

Entre los supuestos usados tenemos:

Costos.- Como ya se mencionó, existen dos componentes principales en los costos del problema a minimizar: Costos de transporte y costos fijos.

Costos de Transporte.- Para obtener los costos de transporte se consideraron las distancias existentes en la red carretera entre cada par de sitios. Estas distancias se obtuvieron del Atlas de Carreteras de México editado por Guía Roji (1990) en donde se tomaron en cuenta las rutas comerciales. Cabe mencionar que existen dos rutas marítimas que son: Mazatlán-La Paz y Mochis-La Paz. en donde se tomó un costo de transporte del doble del considerado por vía terrestre. El costo de transporte se obtiene multiplicando la distancia por el costo por kilómetro-tonelada transportada. Las distancias en kilómetros entre cada par de puntos se encuentran en la Tabla 22. El valor del costo por kilómetro por tonelada transportada se consideró homogéneo en todo el territorio e igual a 445.5 pesos.

Costos Fijos.- Los costos fijos consisten de varios rubros: Costos de inversión (costo de depreciación de equipo), costos de insumos, costos de operación, costos de distribución, gastos financieros, utilidad y varios. Todos los costos fijos se tomaron anualizados. Los costos fijos se encuentran en la Tabla 24.

Demandas.- Con base en las estimaciones de consumo potencial de tortilla (Vea Tabla 21.) y de la distribución del mismo por entidad federativa y ciudad, se obtuvo una demanda potencial en cada uno de los sitios considerados. Se tomaron en cuenta dos escenarios de acuerdo a que el consumo de la tortilla empacada, en principio, se limita

únicamente a las clases media y alta. Uno de los escenarios supone la captación de un 10% del mercado potencial de demanda y el otro del 20% del mercado potencial de demanda.

Número de Plantas.- Debido a la gran diversidad en cuanto al tamaño de la demanda, se dividieron los centros de demanda en dos clases: (1) Zona Metropolitana, Guadalajara, Monterrey y Puebla. (2) los 76 centros restantes.

Para la primera clase, es claro que en la solución óptima se deben colocar plantas de al menos el tamaño de la demanda en las mismas y de capacidad máxima además de dejar la opción al modelo de colocar plantas adicionales para satisfacer la demanda de sitios cercanos a la misma. En la segunda clase se tomaron como posibles opciones de solución, la construcción de dos plantas de 2 toneladas diarias, 2 plantas de 5.34 toneladas diarias, 4 plantas de 16 toneladas diarias; dando un total de ocho posibilidades. Esto es, la solución puede ser una combinación de las ocho posibilidades.

Centros de Insumo.- Se consideraron 19 puntos de insumos de harina de maíz, estos lugares corresponden a la existencia de fábricas de harina de maíz ocho de MICONSA (localizadas en Los Mochis, Sin.; Guadalajara, Jal.; Tlalnepantla, Edo. de Mex.; Arriaga, Chis.; Jaltipán, Ver.; Atlacomulco, Edo. de Mex.; Acapulco, Gro. y Monterrey, N.L.) y once de MASECA (localizadas en Monterrey, N.L.; Río Bravo, Tamps.; Tampico, Tamps.; Chinameca, Ver.; Teotihuacan, Edo. de Mex.; Zamora, Mich.; Guadalajara, Jal.; Acaponeta, Nay.; Culiacán, Sin.; Obregón, Son. y Chihuahua, Chi.) Para cada posible lugar de localización se considera la distancia mínima entre dicho lugar y todos los puntos de insumo.

Enfoques clásicos de solución

El problema se trató de solucionar de manera exacta mediante dos técnicas clásicas que se han venido utilizando: descomposición de Benders [MIR90] y ramificación y acotamiento [KHU72 y MIR86]. Ambas técnicas resultan adecuadas y eficientes para resolver problemas de tamaño chico, pero para el problema anteriormente descrito, donde se tienen 640 variables binarias estas técnicas son impracticables, pues para instancias mayores de 60 variables binarias presentan problemas de tiempo de ejecución excesivo.

Solución con recocido simulado

Como ya se comentó el algoritmo de recocido simulado puede usarse para resolver el problema expuesto. Para ello se procedió a implantar un código de computadora que resolviera el problema. El lenguaje que se usó en la programación fué FORTRAN 77;

el programa consta de alrededor de 1800 lineas de código y corre en una computadora IBM-PS/70 de 25 mhz. En este programa se especificó un mecanismo de generación de vecindades y un programa de enfriamiento. El mecanismo de generación de vecindades que se usó en la solución, es el ya especificádo en la Sección 3.3 y el programa de enfriamiento usado se hizo a partir de la reducción del valor del parámetro de control c usando diferentes factores; tales como, 0.80, 0.85, 0.90 y 0.95. El periodo entre reducciones del parámetro de control se tomó constante e igual a 200, 400, 600 y 1000. Se hicieron varias corridas para cada combinación del periodo y reducción del parámetro de control.

Resultados obtenidos

1. En ambas corridas (10% y 20% de la demanda potencial), las carreteras donde hay flujo de tortillas empacadas tienen distancias pequeñas. Esto es, sólo existe transferencia entre ciudades cercanas lo que asegura el plan de distribución óptimo. Existen pocos movimientos en la red carretera. Entre 140 ligas interciudades existentes, sólo la mitad (aproximadamente 70) de ellas tienen flujo de mercancía. Se puede observar que los costos de transporte (flete) son muy pequeños.

2. Dado que el costo de colocar una planta es muy superior al costo de transporte, optimizar las inversiones es predominante, entonces es lógico que la capacidad total debe ser ligeramente mayor que la demanda total. En las corridas, las capacidades instaladas son 0.44% superior a la demanda total para el escenario de 10% de la demanda potencial, y únicamente 0.15% para el escenario de 20%. Esto implica que se aprovecha casi totalmente las capacidades de las plantas. (Vea Tabla 31.)

   En la mayoría de los casos, la capacidad instalada en cada ciudad es similar a la demanda de la misma, lo cual es razonable ya que se evitan transferencias de la mercancía entre las ciudades. Esto significa que el modelo de optimización tiende a satisfacer las demandas locales.

3. Para el caso de 10% de demanda, el costo de transporte óptimo es de 5,700 millones de pesos anuales mientras que el costo de inversión anualizado es de 455,100 millones de pesos. El costo de transporte es sólo 1.25% de la inversión total (vea las Tablas 25 y 26). Para el caso de 20% de demanda, el costo de transporte óptimo es de 7,466 millones de pesos anuales mientras que el costo de inversión anualizado es de 906,650 millones de pesos. El costo de transporte es sólo 0.82% de la inversión total (vea las Tablas 27 y 28).

Tabla 21. CONSUMO POTENCIAL URBANO
DEMANDA EN TONELADAS ANUALES

NUM.	ENTIDAD FEDERATIVA	DEMANDA POTENCIAL	10% DEMANDA POTENCIAL	20% DEMANDA POTENCIAL
1	AGUASCALIENTES	43742	4374	8748
2	TIJUANA	52177	5218	10435
3	MEXICALI	44927	4493	8985
4	ENSENADA	12867	1287	2573
5	LA PAZ	9489	949	1898
6	CAMPECHE	16533	1653	3307
7	CARMEN	8672	867	1734
8	TORREON, G.PAL.,LER	80608	8061	16122
9	SALTILLO	27611	2761	5522
10	MONCLOVA	13478	1348	2696
11	PIEDRAS NEGRAS	6882	688	1376
12	COLIMA	14519	1452	2904
13	JUAREZ	49565	4956	9913
14	CHIHUAHUA	31419	3142	6284
15	PARRAL	6988	699	1398
16	DELICIAS	6344	634	1269
17	TUXTLA GUTIERREZ	43559	4356	8712
18	TAPACHULA	39512	3951	7902
19	DISTRITO FEDERAL	1497593	149759	299519
20	DURANGO	38131	3813	7626
21	ACAPULCO	60727	6073	12145
22	IGUALA	15707	1571	3141
23	CHILPANCINGO	12587	1259	2517
24	LEON	86702	8670	17340
25	IRAPUATO	26564	2656	5313
26	CELAYA	18114	1811	3623
27	SALAMANCA	14045	1404	2809
28	GUANAJUATO	8361	836	1672
29	PACHUCA	75415	7542	15083
30	TULANCINGO	31940	3194	6388
31	GUADALAJARA	325291	32529	65058
32	GUZMAN	11533	1153	2307
33	OCOTLAN	8449	845	1690
34	TOLUCA	80270	8027	16054
35	OTROS MUNICIPIOS	0	0	0
36	MORELIA	65236	6524	13047
37	URUAPAN	32617	3262	6523
38	ZAMORA	22741	2274	4548
39	APATZINGAN	17780	1778	3556
40	ZITACUARO	14487	1449	2897
41	LA PIEDAD	13830	1383	2766
42	CUERNAVACA	56309	5631	11262
43	CUAUTLA	17819	1782	3564
44	TEPIC	41360	4136	8272
45	MONTERREY	214626	21463	42925
46	OAXACA	73652	7365	14730

Cont. Tabla 21. CONSUMO POTENCIAL URBANO
DEMANDA EN TONELADAS ANUALES

NUM.	ENTIDAD FEDERATIVA	DEMANDA POTENCIAL	DEMANDA 10% POTENCIAL	DEMANDA 20% POTENCIAL
47	PUEBLA	199602	19960	39920
48	TEHUACAN	18708	1871	3742
49	ATLIXCO	16491	1649	3298
50	QUERETARO	65586	6559	13117
51	CHETUMAL	8760	876	1752
52	CANCUN	8760	876	1752
53	SAN LUIS POTOSI	95316	9532	19063
54	VALLES	18267	1827	3653
55	CULIACAN	63188	6319	12638
56	MAZATLAN	46399	4640	9280
57	MOCHIS	25458	2546	5092
58	HERMOSILLO	32938	3294	6588
59	OBREGON	21422	2142	4284
60	GUAYMAS	10674	1067	2135
61	NOGALES	9583	958	1917
62	SAN LUIS R. COLORADO	9340	934	1868
63	NAVOJOA	8115	812	1623
64	VILLAHERMOSA	65136	6514	13027
65	TAMPICO	46195	4620	9239
66	LAREDO	25436	2544	5087
67	MATAMOROS	23498	2350	4700
68	REYNOSA	23465	2346	4693
69	VICTORIA	14347	1435	2869
70	MANTE	8734	873	1747
71	RIO BRAVO	6664	666	1333
72	APIZACO	16601	1660	3320
73	VERACRUZ	96091	9609	19218
74	ORIZABA Y CORDOBA	94668	9467	18934
75	POZA RICA Y TUXPAN	74169	7417	14834
76	JALAPA	55690	5569	11138
77	COATZACOALCOS	63574	6357	12715
78	MERIDA	69804	6980	13961
79	ZACATECAS	37895	3790	7579
80	FRESNILLO	33445	3344	6689
***	Total ***	4804797	480480	960959

Tabla 22. DISTANCIA EN KILOMETROS ENTRE PAREJAS DE CIUDADES

CIUDADES		DIS.	CIUDADES		DIS.	CIUDADES		DIS.
1	31	258	19	75	94	38	41	50
1	24	127	19	47	127	42	43	34
1	53	169	19	43	100	43	49	96
1	79	130	19	42	85	44	56	293
2	4	108	19	34	64	45	68	157
2	3	189	19	50	24	45	69	297
3	4	256	20	56	318	45	66	230
3	62	71	20	80	190	46	48	246
4	5	1348	21	23	132	47	48	127
5	56	860	22	23	100	47	49	34
5	57	412	22	34	185	47	72	47
6	7	213	22	43	92	47	76	185
6	51	420	24	53	197	47	74	148
6	78	192	24	28	58	48	74	65
7	64	168	24	25	46	50	54	401
8	9	280	24	41	112	51	52	379
8	10	252	24	31	244	51	64	570
8	15	313	25	28	45	52	78	322
8	16	456	25	26	57	53	70	338
8	20	253	25	27	20	53	54	272
8	80	285	25	41	87	53	79	188
9	10	203	26	27	57	54	70	96
9	45	85	26	50	50	54	65	138
9	70	572	26	36	145	55	57	206
9	53	450	27	36	111	55	56	224
10	11	236	28	53	212	57	63	156
10	45	194	29	30	83	58	61	279
11	13	1151	29	50	226	58	60	136
11	66	160	29	54	383	58	62	618
12	21	1093	30	72	121	59	60	127
12	32	70	30	75	120	59	63	67
13	14	375	31	79	318	61	62	556
13	61	2593	31	33	82	64	77	169
13	58	2314	31	32	120	65	69	290
14	16	70	31	44	227	65	75	251
14	58	1939	32	33	160	65	70	154
15	16	301	32	38	158	66	68	250
15	20	409	33	41	100	67	69	318
15	55	400	34	35	0	67	71	74
17	18	395	34	40	97	68	71	25
17	46	542	34	50	193	69	70	136
17	48	853	36	38	140	73	76	151
17	64	293	36	37	114	73	77	247
18	46	794	36	40	150	73	74	152
19	29	94	37	39	78	74	76	244
19	30	125	37	38	107	75	76	297
						79	80	100

Tabla 23. DISTANCIA DE CENTROS DE INSUMO A CIUDADES

CIUDAD	INSUMO	DISTANCIA	CIUDAD	INSUMO	DISTANCIA
1	31	258.00	41	38	50.00
2	59	1141.00	42	19	85.00
3	59	952.00	43	19	100.00
4	59	1208.00	44	82	150.00
5	57	412.00	45	45	.00
6	77	550.00	46	74	311.00
7	77	337.00	47	19	127.00
8	45	365.00	48	74	65.00
9	45	85.00	49	19	161.00
10	45	194.00	50	19	24.00
11	45	390.00	51	77	739.00
12	31	190.00	52	77	1064.00
13	14	375.00	53	38	359.00
14	14	.00	54	65	138.00
15	14	371.00	55	55	.00
16	14	70.00	56	82	143.00
17	17	.00	57	57	.00
18	17	395.00	58	59	263.00
19	19	.00	59	59	.00
20	82	461.00	60	59	127.00
21	21	.00	61	59	542.00
22	34	185.00	62	59	881.00
23	21	132.00	63	59	67.00
24	38	162.00	64	77	169.00
25	19	131.00	65	65	.00
26	19	74.00	66	45	230.00
27	19	131.00	67	71	74.00
28	19	176.00	68	71	25.00
29	19	94.00	69	65	290.00
30	19	125.00	70	65	154.00
31	31	.00	71	71	.00
32	31	120.00	72	19	174.00
33	31	82.00	73	74	152.00
34	34	.00	74	74	.00
35	35	.00	75	19	94.00
36	38	140.00	76	74	244.00
37	38	107.00	77	77	.00
38	38	.00	78	77	742.00
39	38	185.00	79	31	318.00
40	34	97.00	80	31	418.00

Tabla 24. COSTOS FIJOS DE PLANTAS

	TAMAÑOS DE PLANTA		
	2 TON/DIA	5.4 TON/DIA	16 TON/DIA
INVERSION (ANUAL)	1.3×10^8	1.8×10^8	3.6×10^8
MATERIA PRIMA	1.3×10^8	2.3×10^8	4.4×10^8
OPERACION	3.2×10^8	8.4×10^8	2.5×10^9
VARIOS	2.3×10^8	4.7×10^8	1.4×10^9
TOTAL	8.1×10^8	1.72×10^9	4.7×10^9

Tabla 25. SOLUCION OPTIMA PARA EL ESCENARIO 10% DEMANDA

UBICACION, TIPO Y CAPACIDAD DE LAS PLANTAS

	P L A N T A S			
CIUDAD	2 TON.	5.34 TON.	16 TON.	CAPACIDAD
1	0	0	1	4992
2	0	0	1	4992
3	0	0	1	4992
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	2	9984
9	0	0	1	4992
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	1	4992
14	0	0	1	4992
15	0	0	0	0
16	0	0	0	0
17	0	0	1	4992
18	0	0	1	4992
19	0	0	31	154752
20	0	0	1	4992
21	0	0	1	4992
22	0	0	1	4992
23	0	0	0	0
24	0	0	2	9984
25	0	0	0	0
26	0	0	0	0
27	0	0	1	4992
28	0	0	0	0
29	0	0	1	4992
30	0	0	1	4992
31	0	2	6	33284
32	0	0	0	0
33	0	0	0	0
34	0	0	2	9984
35	0	0	0	0
36	0	0	1	4992
37	0	0	1	4992
38	0	0	1	4992
39	0	0	0	0
40	0	0	0	0
41	0	0	0	0
42	0	0	1	4992
43	0	0	0	0
44	0	0	1	4992

Cont. Tabla 25. SOLUCION OPTIMA PARA EL ESCENARIO 10% DEMANDA

UBICACION, TIPO Y CAPACIDAD DE LAS PLANTAS

	P L A N T A S			
CIUDAD	2 TON.	5.34 TON.	16 TON.	CAPACIDAD
45	0	1	4	21634
46	0	0	1	4992
47	0	0	5	24960
48	0	0	1	4992
49	0	0	0	0
50	0	0	1	4992
51	0	0	0	0
52	0	0	0	0
53	0	0	2	9984
54	0	0	1	4992
55	0	0	1	4992
56	0	0	1	4992
57	0	0	1	4992
58	0	0	1	4992
59	0	0	1	4992
60	0	0	0	0
61	0	0	0	0
62	0	0	0	0
63	0	0	0	0
64	0	0	2	9984
65	0	0	1	4992
66	0	2	0	3332
67	0	0	0	0
68	0	0	0	0
69	0	0	0	0
70	0	0	0	0
71	0	0	1	4992
72	0	0	0	0
73	0	0	2	9984
74	0	0	1	4992
75	0	0	1	4992
76	0	0	1	4992
77	0	0	1	4992
78	0	0	2	9984
79	0	0	1	4992
80	0	0	0	0
TOTALES	0	5	95	482570

Tabla 26. FLUJO OPTIMO PARA EL ESCENARIO 10% DE DEMANDA
CON SU RESPECTIVA ZONA DE INFLUENCIA

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
1	24	618.	127.	3.50
3	2	225.	189.	1.89
3	4	1286.	256.	14.67
7	6	161.	213.	1.53
8	80	1924.	285.	24.43
9	10	1347.	203.	12.18
9	45	884.	85.	3.35
14	16	1332.	70.	4.15
16	15	698.	301.	9.36
17	46	637.	542.	15.38
18	46	1041.	794.	36.82
19	75	1691.	94.	7.08
19	42	1335.	85.	5.06
19	50	2475.	24.	2.65
20	80	1179.	190.	9.98
22	23	2338.	100.	10.42
22	43	1084.	92.	4.44
23	21	1080.	132.	6.35
24	28	383.	58.	.99
24	25	1501.	46.	3.08
24	41	48.	112.	.24
27	25	1155.	20.	1.03
27	26	902.	57.	2.29
27	36	1531.	111.	7.57
30	29	1434.	83.	5.30
30	72	364.	121.	1.96
31	33	844.	82.	3.08
31	32	1268.	120.	6.78
32	12	1451.	70.	4.52
34	19	509.	64.	1.45
34	40	1448.	97.	6.26
37	39	1778.	78.	6.18
38	32	1336.	158.	9.40
38	37	47.	107.	.22
38	41	1335.	50.	2.97
42	43	697.	34.	1.06
44	31	1150.	227.	11.63
45	68	268.	157.	1.87
45	69	616.	297.	8.15
47	49	1649.	34.	2.50
47	72	1296.	47.	2.71
47	74	2047.	148.	13.50
48	46	695.	246.	7.62
48	74	2427.	65.	7.03
50	26	909.	50.	2.02
53	28	453.	212.	4.28
54	29	1115.	383.	19.02

Cont. Tabla 26. FLUJO OPTIMO PARA EL ESCENARIO 10% DE DEMANDA CON SU RESPECTIVA ZONA DE INFLUENCIA

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
54	70	1691.	96.	7.23
54	65	360.	138.	2.21
56	44	294.	293.	3.84
56	55	59.	224.	.59
57	5	948.	412.	17.40
57	55	1267.	206.	11.63
57	63	232.	156.	1.61
58	61	958.	279.	11.91
58	62	1945.	618.	53.55
59	60	2271.	127.	12.85
59	63	579.	67.	1.73
60	58	1204.	136.	7.29
62	3	1011.	71.	3.20
64	7	1028.	168.	7.69
64	51	876.	570.	22.24
64	77	1567.	169.	11.80
65	75	733.	251.	8.20
66	11	688.	160.	4.90
66	68	101.	250.	1.12
70	69	818.	136.	4.96
71	67	2349.	74.	7.74
71	68	1977.	25.	2.20
73	76	577.	151.	3.88
77	73	202.	247.	2.22
78	6	1492.	192.	12.76
78	52	876.	322.	12.57
79	31	962.	318.	13.63
79	80	241.	100.	1.07
TOTAL				570.00

Tabla 27. SOLUCION OPTIMA PARA EL ESCENARIO 20% DEMANDA

UBICACION, TIPO Y CAPACIDAD DE LAS PLANTAS

	P L A N T A S			
CIUDAD	2 TON.	5.34 TON.	16 TON.	CAPACIDAD
1	0	0	2	9984
2	0	0	2	9984
3	0	0	1	4992
4	0	0	1	4992
5	0	0	0	0
6	0	0	1	4992
7	0	0	0	0
8	0	0	4	19968
9	0	0	1	4992
10	0	0	1	4992
11	0	0	0	0
12	0	0	1	4992
13	0	0	2	9984
14	0	0	1	4992
15	0	0	0	0
16	0	0	0	0
17	0	0	2	9984
18	0	0	2	9984
19	0	0	60	299520
20	0	0	2	9984
21	0	0	2	9984
22	0	0	1	4992
23	0	0	1	4992
24	0	0	3	14976
25	0	0	1	4992
26	0	0	1	4992
27	0	0	1	4992
28	0	0	0	0
29	0	0	3	14976
30	0	0	1	4992
31	1	0	13	65520
32	0	0	0	0
33	0	0	0	0
34	0	0	3	14976
35	0	0	0	0
36	0	0	3	14976
37	0	0	1	4992
38	0	0	1	4992
39	0	0	1	4992
40	0	0	0	0
41	0	0	1	4992
42	0	0	2	9984
43	0	0	1	4992

Cont. Tabla 27. SOLUCION OPTIMA PARA EL ESCENARIO 20% DEMANDA

UBICACION, TIPO Y CAPACIDAD DE LAS PLANTAS

	P L A N T A S			
CIUDAD	2 TON.	5.34 TON.	16 TON.	CAPACIDAD
44	0	0	2	9984
45	0	2	8	43268
46	0	0	3	14976
47	0	0	8	39936
48	0	0	0	0
49	0	0	1	4992
50	0	0	3	14976
51	0	0	0	0
52	0	0	0	0
53	0	0	3	14976
54	0	0	1	4992
55	0	0	3	14976
56	0	0	1	4992
57	0	0	1	4992
58	0	0	1	4992
59	0	0	1	4992
60	0	0	1	4992
61	0	0	0	0
62	0	0	1	4992
63	0	0	0	0
64	0	0	3	14976
65	0	0	2	9984
66	0	0	1	4992
67	0	0	1	4992
68	0	0	1	4992
69	0	0	1	4992
70	0	0	0	0
71	0	0	0	0
72	0	0	1	4992
73	0	0	4	19968
74	0	0	3	14976
75	0	0	3	14976
76	0	0	2	9984
77	0	0	3	14976
78	0	0	3	14976
79	0	0	2	9984
80	0	0	1	4992
TOTALES	1	2	192	962420

Tabla 28. FLUJO OPTIMO PARA EL ESCENARIO 20% DE DEMANDA CON SU RESPECTIVA ZONA DE INFLUENCIA

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
1	53	1236.	169.	9.31
4	2	451.	108.	2.17
4	3	1968.	256.	22.44
6	51	950.	420.	17.78
6	78	736.	192.	6.30
8	15	1523.	313.	21.24
8	16	2488.	456.	50.54
10	8	164.	252.	1.84
10	9	530.	203.	4.79
10	11	1471.	236.	15.47
10	45	132.	194.	1.14
11	66	95.	160.	.68
12	32	2089.	70.	6.51
13	14	71.	375.	1.19
15	55	126.	400.	2.25
16	14	1220.	70.	3.80
17	46	686.	542.	16.56
17	64	587.	293.	7.66
18	46	1413.	794.	49.98
19	29	107.	94.	.45
19	30	1111.	125.	6.19
20	56	2358.	318.	33.41
22	34	2046.	185.	16.86
22	43	119.	92.	.49
23	21	2161.	132.	12.71
23	22	314.	100.	1.40
25	24	1912.	46.	3.92
25	28	1962.	45.	3.93
26	25	2011.	57.	5.11
27	25	2183.	20.	1.95
28	53	290.	212.	2.74
31	44	217.	227.	2.19
33	31	217.	82.	.79
34	40	968.	97.	4.18
36	40	1929.	150.	12.89
38	32	217.	158.	1.53
38	37	95.	107.	.45
38	41	132.	50.	.29
39	37	1436.	78.	4.99
41	24	452.	112.	2.26
41	33	1906.	100.	8.49
43	42	1277.	34.	1.93
43	49	271.	96.	1.16
44	56	1929.	293.	25.18
45	68	132.	157.	.92
46	48	2345.	246.	25.70

Cont. Tabla 28. FLUJO OPTIMO PARA EL ESCENARIO 20% DE DEMANDA CON SU RESPECTIVA ZONA DE INFLUENCIA

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
47	48	1396.	127.	7.90
47	74	2099.	148.	13.84
49	47	1965.	34.	2.98
50	19	1218.	24.	1.30
50	26	641.	50.	1.43
54	53	1853.	272.	22.45
55	57	2465.	206.	22.62
57	5	1897.	412.	34.82
57	63	469.	156.	3.26
58	61	817.	279.	10.15
59	63	1154.	67.	3.44
60	58	2412.	136.	14.61
60	59	446.	127.	2.52
62	3	2025.	71.	6.41
62	61	1099.	556.	27.22
64	7	1734.	168.	12.98
64	51	802.	570.	20.37
65	54	514.	138.	3.16
65	70	231.	154.	1.58
67	71	901.	74.	2.97
68	71	431.	25.	.48
69	67	608.	318.	8.61
69	70	1515.	136.	9.18
72	30	142.	121.	.77
72	47	1530.	47.	3.20
73	76	1154.	151.	7.76
73	74	1858.	152.	12.58
75	30	143.	120.	.76
77	73	2262.	247.	24.89
78	52	1752.	322.	25.13
79	53	708.	188.	5.93
79	80	1697.	100.	7.56
TOTAL				746.64

Tabla 29. FLUJO OPTIMO PARA EL ESCENARIO 10% DE DEMANDA ORDENADO POR DESTINO

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
3	2	225.	189.	1.89
62	3	1011.	71.	3.20
3	4	1286.	256.	14.67
57	5	948.	412.	17.40
7	6	161.	213.	1.53
78	6	1492.	192.	12.76
64	7	1028.	168.	7.69
9	10	1347.	203.	12.18
66	11	688.	160.	4.90
32	12	1451.	70.	4.52
16	15	698.	301.	9.36
14	16	1332.	70.	4.15
34	19	509.	64.	1.45
23	21	1080.	132.	6.35
22	23	2338.	100.	10.42
1	24	618.	127.	3.50
24	25	1501.	46.	3.08
27	25	1155.	20.	1.03
27	26	902.	57.	2.29
50	26	909.	50.	2.02
24	28	383.	58.	.99
53	28	453.	212.	4.28
30	29	1434.	83.	5.30
54	29	1115.	383.	19.02
44	31	1150.	227.	11.63
79	31	962.	318.	13.63
31	32	1268.	120.	6.78
38	32	1336.	158.	9.40
31	33	844.	82.	3.08
27	36	1531.	111.	7.57
38	37	47.	107.	.22
37	39	1778.	78.	6.18
34	40	1448.	97.	6.26
24	41	48.	112.	.24
38	41	1335.	50.	2.97
19	42	1335.	85.	5.06
22	43	1084.	92.	4.44
42	43	697.	34.	1.06
56	44	294.	293.	3.84
9	45	884.	85.	3.35
17	46	637.	542.	15.38
18	46	1041.	794.	36.82
48	46	695.	246.	7.62
47	49	1649.	34.	2.50
19	50	2475.	24.	2.65
64	51	876.	570.	22.24

Cont. Tabla 29. FLUJO OPTIMO PARA EL ESCENARIO 10% DE DEMANDA ORDENADO POR DESTINO

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
78	52	876.	322.	12.57
56	55	59.	224.	.59
57	55	1267.	206.	11.63
60	58	1204.	136.	7.29
59	60	2271.	127.	12.85
58	61	958.	279.	11.91
58	62	1945.	618.	53.55
57	63	232.	156.	1.61
59	63	579.	67.	1.73
54	65	360.	138.	2.21
71	67	2349.	74.	7.74
45	68	268.	157.	1.87
66	68	101.	250.	1.12
71	68	1977.	25.	2.20
45	69	616.	297.	8.15
70	69	818.	136.	4.96
54	70	1691.	96.	7.23
30	72	364.	121.	1.96
47	72	1296.	47.	2.71
77	73	202.	247.	2.22
47	74	2047.	148.	13.50
48	74	2427.	65.	7.03
19	75	1691.	94.	7.08
65	75	733.	251.	8.20
73	76	577.	151.	3.88
64	77	1567.	169.	11.80
8	80	1924.	285.	24.43
20	80	1179.	190.	9.98
79	80	241.	100.	1.07
TOTAL				570.00

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
4	2	451.	108.	2.17
4	3	1968.	256.	22.44
62	3	2025.	71.	6.41
57	5	1897.	412.	34.82
64	7	1734.	168.	12.98
10	8	164.	252.	1.84
10	9	530.	203.	4.79
10	11	1471.	236.	15.47
13	14	71.	375.	1.19
16	14	1220.	70.	3.80
8	15	1523.	313.	21.24
8	16	2488.	456.	50.54
50	19	1218.	24.	1.30
23	21	2161.	132.	12.71
23	22	314.	100.	1.40
25	24	1912.	46.	3.92
41	24	452.	112.	2.26
26	25	2011.	57.	5.11
27	25	2183.	20.	1.95
50	26	641.	50.	1.43
25	28	1962.	45.	3.93
19	29	107.	94.	.45
19	30	1111.	125.	6.19
72	30	142.	121.	.77
75	30	143.	120.	.76
33	31	217.	82.	.79
12	32	2089.	70.	6.51
38	32	217.	158.	1.53
41	33	1906.	100.	8.49
22	34	2046.	185.	16.86
38	37	95.	107.	.45
39	37	1436.	78.	4.99
34	40	968.	97.	4.18
36	40	1929.	150.	12.89
38	41	132.	50.	.29
43	42	1277.	34.	1.93
22	43	119.	92.	.49
31	44	217.	227.	2.19
10	45	132.	194.	1.14
17	46	686.	542.	16.56
18	46	1413.	794.	49.98
49	47	1965.	34.	2.98
72	47	1530.	47.	3.20
46	48	2345.	246.	25.70
47	48	1396.	127.	7.90

Cont. Tabla 30. FLUJO OPTIMO PARA EL ESCENARIO 20% DE DEMANDA ORDENADO POR DESTINO

CIUDAD ORIG.	CIUDAD DEST.	FLUJO TON. ANUAL	DISTANCIA KM.	COSTO DECENAS DE MILLONES
43	49	271.	96.	1.16
6	51	950.	420.	17.78
64	51	802.	570.	20.37
78	52	1752.	322.	25.13
1	53	1236.	169.	9.31
28	53	290.	212.	2.74
54	53	1853.	272.	22.45
79	53	708.	188.	5.93
65	54	514.	138.	3.16
15	55	126.	400.	2.25
20	56	2358.	318.	33.41
44	56	1929.	293.	25.18
55	57	2465.	206.	22.62
60	58	2412.	136.	14.61
60	59	446.	127.	2.52
58	61	817.	279.	10.15
62	61	1099.	556.	27.22
57	63	469.	156.	3.26
59	63	1154.	67.	3.44
17	64	587.	293.	7.66
11	66	95.	160.	.68
69	67	608.	318.	8.61
45	68	132.	157.	.92
65	70	231.	154.	1.58
69	70	1515.	136.	9.18
67	71	901.	74.	2.97
68	71	431.	25.	.48
77	73	2262.	247.	24.89
47	74	2099.	148.	13.84
73	74	1858.	152.	12.58
73	76	1154.	151.	7.76
6	78	736.	192.	6.30
79	80	1697.	100.	7.56
TOTAL		746.64		

Tabla 31. EXCEDENTE DE LA CAPACIDAD DE PRODUCCION EN PLANTAS

ESCENARIO 10% DEMANDA

CIUDAD	NUM.	EXCEDENTE EN TON. ANUALES
CIUDAD JUAREZ	13	36.
CHIHUAHUA	14	519.
GUADALAJARA	31	755.
MONTERREY	45	171.
PUEBLA	47	8.
MERIDA	78	636.
TOTAL		2125.

ESCENARIO 20% DEMANDA

CIUDAD	NUM.	EXCEDENTE EN TON. ANUALES
TAPACHULA	18	669.
GUADALAJARA	31	462.
MONTERREY	45	343.
PUEBLA	47	16.
TOTAL		1490.

4. Las zonas de influencia se reduce a regiones pequeñas. Estas regiones se muestran en las Tablas 26 y 27. En dichas tablas se listan los flujos por la numeración de la ciudad origen, mientras que en las Tablas 29 y 30, se listan por la numeración de la ciudad destino. Con estas tablas, se pueden conocer fácilmente las interacciones de flujos entre diferentes ciudades.

5. El número, tipo, capacidad de plantas se muestra en las tablas 25 y 27, así como su capacidad de producción anual instalada. Se puede notar las ciudades que requieren mayor inversión en el orden de magnitud son: Distrito Federal, Guadalajara, Monterrey y Puebla. Existen varias ciudades que no requieren ninguna inversión ya que sus demandas son pequeñas y pueden satisfacerse por zonas aledañas. A pesar de considerar sólo el 10% de la demanda potencial, se requiere muchas plantas en que invertir, lo cual implica un mercado futuro atractivo.

Se puede observar que el modelo introduce, por lo general plantas grandes para satisfacer las demandas, esto se debe a que los costos marginales son menores que los costos en que se incurriría si se intalaran plantas más chicas. El modelo únicamente introduce plantas chicas o medianas para ajustar la demanda a la oferta. Por otro lado, como el costo de inversión es muy superior al costo de transporte, es preferible instalar plantas grandes y hacer transferencias de flujo para satisfacer demandas en los centros donde la demanda es menor.

CAPÍTULO 4

CONVERGENCIA ASINTÓTICA

El algoritmo de recocido simulado ha sido explicado, haciendo una analogía con el proceso físico de recocido. En esta descripción se conjeturó acerca de la existencia de distribuciones estacionarias para el algoritmo de recocido simulado, equivalente a la existencia de la distribución de Boltzmann en equilibrio térmico en física estadística. Usando el resultado positivo de esta conjetura, se demostró la convergencia asintótica del algoritmo de recocido simulado a un conjunto de soluciones globalmente óptimas (Vea Corolario 2.1). En este capítulo se prueba la validez de esta conjetura. Asimismo, se introduce una formulación del algoritmo de recocido simulado basado en cadenas de Markov y se discuten las condiciones para las cuales se garantiza la convergencia asintótica.

El capítulo se desarrolla como sigue: en la Sección 4.1 se dan algunos resultados básicos de la teoría de cadenas de Markov homogéneas; la Sección 4.2 encuadra cada iteración del algoritmo de recocido simulado como un proceso markoviano homogéneo y con base en esto, se prueba la convergencia asintótica al conjunto de soluciones óptimas; en la Sección 4.3 se dan algunos resultados de la teoría de cadenas de Markov no homogéneas; en la Sección 4.4 se ve el proceso de recocido como una cadena de Markov no homogénea y se prueba convergencia al conjunto de soluciones óptimas y en la Sección 4.5 se dan algunos resultados del comportamiento asintótico del algoritmo.

4.1 CADENAS DE MARKOV

Como ya se mencionó anteriormente, el algoritmo de recocido simulado puede modelarse matemáticamente usando la teoría de cadenas de Markov. El propósito de esta sección es introducir los conceptos sobre cadenas de Markov que se usarán posteriormente.

Definición 1 *Una* cadena de Markov *es una sucesión* de ensayos, *donde la probabilidad del resultado de un ensayo dado, depende únicamente del ensayo previo. Sea* $X(k)$ *la variable aleatoria que denota el resultado del* k*-ésimo ensayo. Entonces la probabilidad*

de transición del k-ésimo ensayo para cada pareja i, j de resultados, está definida como

$$P_{ij} = P\{X(k) = j | X(k-1) = i\} \tag{1}$$

La matriz $\mathbf{P}(k)$ *cuyos elementos están dados por (1) se denomina la* matriz de transición.

Sea $a_i(k)$ denote la probabilidad del resultado i en el k-ésimo ensayo, es decir,

$$a_i(k) = P\{X(k) = i\}. \tag{2}$$

Entonces $a_i(k)$ está dada por la fórmula de recursión siguiente

$$a_i(k) = \sum_l a_l(k-1) P_{li}(k) \tag{3}$$

Definición 2 *Una cadena de Markov se llama* finita *si está definida en un conjunto finito de resultados.*

Definición 3 *Una cadena de Markov se llama* no homogénea *si las probabilidades de transición dependen del ensayo k. Si las probabilidades de transición son independientes del ensayo, la cadena de Markov se llama* homogénea.

Definición 4 *Un vector* **a** *se llama* estocástico *si sus componentes* a_i *satisfacen las siguientes condiciones:*

$$a_i \geq 0, \quad \textit{para toda } i, \quad \textit{y} \quad \sum_i a_i = 1. \tag{4}$$

Una matriz **P** *se llama* estocástica *si sus componentes* P_{ij} *satisfacen las condiciones siguientes:*

$$P_{ij} \geq 0, \quad \textit{para toda } i, j \quad \textit{y} \quad \sum_j P_{ij} = 1, \quad \textit{para toda } i. \tag{5}$$

En lo que sigue se hablará de la distribución estacionaria de una cadena de Markov homogénea y finita y se darán algunos resultados importantes que se utilizarán para probar la convergencia asintótica del algoritmo de recocido simulado a una distribución estacionaria única dada en (2.5).

Definición 5 (Feller,1950) *La distribución estacionaria de una cadena de Markov homogénea y finita con matriz de transición* **P** *está definida como el vector* **q**, *cuya i-ésima componente está dada por*

$$q_i = \lim_{k \to \infty} P\{X(k) = i | X(0) = j\}, \quad \textit{para toda } j. \tag{6}$$

Si la distribución estacionaria $\mathbf{q}$ existe, se tiene

$$\begin{aligned} \lim_{k\to\infty} a_i(k) &= \lim_{k\to\infty} P\{X(k)=i\} \\ &= \lim_{k\to\infty} \sum_j P\{X(k)=i|X(0)=j\}P\{X(0)=j\} \qquad (7) \\ &= q_i \sum_j P\{X(0)=j\} = q_i. \end{aligned}$$

Así, la distribución estacionaria es la distribución de probabilidad de las soluciones después de un número infinito de ensayos. Además se tiene que

$$\begin{aligned} \mathbf{q}^T &= \lim_{k\to\infty} \mathbf{a}^T(0) \prod_{l=1}^{k} \mathbf{P}(l) \\ &= \lim_{k\to\infty} \mathbf{a}^T(0)\mathbf{P}^k \\ &= \lim_{k\to\infty} \mathbf{a}^T(0)\mathbf{P}^{k-1}\mathbf{P} \qquad (8) \\ &= \lim_{l\to\infty} \mathbf{a}^T(0)\mathbf{P}^{l}\mathbf{P} \\ &= \mathbf{q}^T\mathbf{P} \end{aligned}$$

Así, $\mathbf{q}$ es un vector propio izquierdo de $\mathbf{P}$ con valor propio igual a 1.

Definición 6 *Una cadena de Markov con matriz de transición* $\mathbf{P}$ *se llama* irreducible, *si para cada par de soluciones* $i, j \in \mathcal{S}$ *existe una probabilidad positiva de alcanzar* j *a partir de* i, *en un número finito de ensayos, es decir,*

$$\forall i, j \;\; \exists n \geq 1 : (\mathbf{P}^n)_{ij} > 0. \qquad (9)$$

Definición 7 *Una cadena de Markov con matriz de transición* $\mathbf{P}$ *se llama* aperiódica, *si para cada solución* $i \in \mathcal{S}$ *el máximo común divisor* $mcd(\mathcal{D}_i) = 1$, *donde el conjunto* $\mathcal{D}_i$ *se define como*

$$\mathcal{D}_i = \{n|\ n \text{ es entero con } (\mathbf{P}^n)_{ii} > 0.\} \qquad (10)$$

El entero $\mathrm{mcd}(\mathcal{D}_i)$ se llama el *periodo* de la solución i. Así, aperiodicidad requiere que todas las soluciones tengan periodo uno.

Lema 1 *Una cadena de Markov irreducible con matriz de transición* $\mathbf{P}$ *es aperiódica si*

$$\exists j \in \mathcal{S} : \;\; P_{jj} > 0. \qquad (11)$$

Prueba El hecho que la cadena sea irreducible implica que

$$\forall i, j \in \mathcal{S},\ \exists k, l \geq 1\ :\ (P^k)_{ij} > 0\ y\ (P^l)_{ji} > 0.$$

Así, $(P^n)_{ii} \geq (P^k)_{ij}(P^l)_{ji} > 0$, donde $n = k + l$. Sin embargo, se tiene $(P^{n+1})_{ii} \geq (P^k)_{ij} P_{jj} (P^l)_{ji} > 0$. Así, $n, n+1 \in \mathcal{D}_i$. Por lo tanto $mcd(\mathcal{D}_i) = 1$, puesto que $1 \leq mcd(\mathcal{D}_i) \leq mcd(n, n+1)$ y $mcd(n, n+1) = 1$, para toda $n \geq 1$.

Teorema 1 (Feller,1950) *Sea* **P** *la matriz de transición asociada con una cadena de Markov homogénea finita y sea la cadena de Markov irreducible y aperiódica. Entonces existe un vector aleatorio* **q** *cuyas componentes* q_i *se determinan de manera única por medio de las siguientes ecuaciones:*

$$\sum_j q_j P_{ji} = q_i,\ \ para\ toda\ i. \tag{12}$$

Claramente, el vector **q** es la distribución estacionaria de la cadena de Markov ya que satisface (8).

Lema 2 *Sea* **P** *una matriz de transición asociada con una cadena de Markov homogénea, finita, irreducible y aperiódica. Entonces dada una distribución* $\{q_i\}$*, si sus componentes satisfacen la siguiente ecuación:*

$$q_i P_{ij} = q_j P_{ji},\ \ para\ toda\ i, j \in \mathcal{S}. \tag{13}$$

Entonces es la distribución estacionaria de la cadena de Markov

Prueba. Como resultado del Teorema 1, la existencia de una distribución estacionaria única **q** está garantizada. La prueba del teorema se completa probando que (13) implica (12) de la siguiente forma: para todo $i, j \in \mathcal{S}$ se tiene

$$\begin{aligned} & q_i P_{ij} = q_j P_{ji} \\ \Rightarrow & \sum_{j \in \mathcal{S}} q_i P_{ij} = \sum_{j \in \mathcal{S}} q_j P_{ji} \\ \Rightarrow & \qquad q_i = \sum_{j \in \mathcal{S}} q_j P_{ji} \end{aligned}$$

ya que **P** es estocástica.

Por lo tanto, para cualquier distribución se puede checar si es la distribución estacionaria, únicamente observando si satisface (13) y verificando que la cadena de Markov es irreducible y aperiódica.

4.2 CONVERGENCIA ASINTÓTICA I

En seguida se aplican los resultados de la sección anterior para probar convergencia asintótica del algoritmo de recocido simulado. En el algoritmo de recocido simulado, un ensayo corresponde a una transición y el conjunto de resultados está dado por un conjunto finito de soluciones. Claramente, en el caso del algoritmo de recocido simulado, el resultado de un ensayo únicamente depende del resultado del ensayo previo (vea Sección 2.2). Esto hace que podamos usar el concepto de cadenas de Markov finitas.

Definición 8 *(Probabilidades de transición) Sea* (S, f) *una instancia de un problema de optimización combinatoria. Entonces las* probabilidades de transición *para el algoritmo de recocido simulado están definidas como sigue:*

$$\forall i, j \in S: \quad P_{ij}(k) = P_{ij}(c_k) = \begin{cases} G_{ij}(c_k)A_{ij}(c_k) & si\ i \neq j \\ 1 - \sum_{l \in S, l \neq i} P_{il}(c_k) & si\ i = j, \end{cases} \tag{14}$$

donde $G_{ij}(c_k)$ *denota la* probabilidad de generación, *es decir, la probabilidad de generar la solución* j *a partir de la* i *y* $A_{ij}(c_k)$ *denota la* probabilidad de aceptación, *es decir, la probabilidad de aceptar la solución* j*, generada a partir de la solución* i*.*

Las probabilidades $G_{ij}(c_k)$ y $A_{ij}(c_k)$ son probabilidades condicionales. Sus correspondientes matrices $\mathbf{G}(c_k)$ y $\mathbf{A}(c_k)$ se llaman la *matriz de generación* y la *matriz de aceptación*, respectivamente.

Definición 9 *(Probabilidad de generación)*

$$\forall i, j \in S: \quad G_{ij}(c_k) = G_{ij} = \frac{1}{\Theta}\chi_{(S_i)}(j), \tag{15}$$

donde $\Theta = |S_i|$*, para toda* $i \in S$*.*

Definición 10 *(Probabilidad de aceptación)*

$$\forall i, j \in S: A_{ij}(c_k) = \exp\left(-\frac{(f(j) - f(i))^+}{c_k}\right), \tag{16}$$

donde, para toda $a \in R$, $a^+ = a$ *si* $a > 0$, *y* $a^+ = 0$ *en otro caso y* $f(i)$ *denota el costo de la solución* i.

Observe que, las probabilidades de generación se escogen de manera independiente del parámetro de control c_k y uniformemente sobre todas las vecindades de igual tamaño, es decir, $|S_i| = \Theta$, para toda $i \in S$. Las probabilidades de aceptación están dadas por el criterio de aceptación (2.4) y por lo tanto idéntico al criterio de Metropolis. Note además que las matrices de transición y de generación para el algoritmo de recocido simulado son estocásticas y $\mathbf{P}$ depende de c, por lo tanto $\mathbf{q}$ depende de c, es decir, $\mathbf{q} = \mathbf{q}(c)$. La matriz de aceptación no es estocástica.

Las definiciones de las probabilidades de generación y aceptación, dadas por (15) y (16), corresponden a la definición original del algoritmo de recocido simulado y siguen de manera cerrada la analogía discutida en el Capítulo 2. Estas definiciones pueden usarse de manera general en todo problema de optimización combinatoria y un examen de la literatura revela que en las aplicaciones prácticas estas definiciones se usan (con alguna mínima variación que se discutirá al final de esta sección). También es posible formular un conjunto de condiciones que tratan la convergencia asintótica para una clase más general de probabilidades de aceptación y generación, (vea [LAA88]).

Enseguida se dan las propiedades asintóticas del algoritmo de recocido simulado y se muestra que converge con probabilidad 1 a una solución óptima, después de posiblemente un número grande de ensayos, es decir.

$$\lim_{k \to \infty} P\{X(k) \in S_{opt}\} = 1. \tag{17}$$

Para probar convergencia asintótica del algoritmo de recocido simulado es esencial la existencia de una distribución estacionaria única. Esta distribución estacionaria solamente existe bajo ciertas condiciones en las cadenas de Markov que están asociadas con el algoritmo. A continuación se discuten dichas condiciones y se prueba que, para cadenas de Markov homogéneas asociadas al algoritmo de recocido simulado, la distribución estacionaria toma la forma exponencial de (2.5), es decir, se prueba la Conjetura 2.1. Para lograr esto, suponemos que el valor del parámetro de control es independiente de k, es decir, $c_k = c$, para toda k. Entonces se tiene que $\mathbf{P}(k) = \mathbf{P}$, para toda k, que corresponde a una cadena de Markov homogénea.

Teorema 2 *Sea* (S, f) *denote una instancia de un problema de optimización combinatoria y* $\mathbf{P}(c)$ *denote la matriz de transición asociada con el algoritmo de recocido simulado definida por (14),(15) y (16). Además, se supone que las siguientes condiciones se satisfacen:*

(1)

$$\forall i, j \in \mathcal{S} \;\; \exists p \geq 1, \;\; \exists\, l_0, l_1, ..., l_p \in \mathcal{S},$$

$$con \;\; l_0 = i, \;\; l_p = j, \;\; y$$

$$G_{l_k l_{k+1}} > 0, \;\; k = 0, 1, ..., p-1. \tag{18}$$

Entonces la cadena de Markov tiene una distribución estacionaria $\mathbf{q}(c)$*, cuyas componentes están dadas por*

$$q_i(c) = \frac{1}{N_0(c)} \exp\left(-\frac{f(i)}{c}\right), \;\; para\, toda\, i \in \mathcal{S}, \tag{19}$$

donde

$$N_0(c) = \sum_{j \in \mathcal{S}} exp\left(-\frac{f(j)}{c}\right). \tag{20}$$

Prueba Se procederá como sigue: primero se demostrará que la cadena de Markov definida es irreducible y aperiódica. Después, se aplica el Lema 2 y se verifica que la distribución es estacionaria mostrando que las componentes dadas por (19) y (20) satisfacen la ecuación de balance de (13).

Irreducibilidad: Usando la condición (1) se tiene

$$\begin{aligned}
(P^p)_{ij}(c) &= \textstyle\sum_{k_1 \in \mathcal{S}} \sum_{k_2 \in \mathcal{S}} \cdots \sum_{k_{p-1} \in \mathcal{S}} P_{ik_1}(c) P_{k_1 k_2}(c) \cdots P_{k_{p-1} j}(c) \\
&\geq G_{il_1} A_{il_1}(c) G_{l_1 l_2} A_{l_1 l_2}(c) \cdots G_{l_{p-1} j} A_{l_{p-1} j}(c) \\
&> 0,
\end{aligned}$$

puesto que $A_{ij} > 0$, para toda $i, j \in \mathcal{S}$; vea (16)

Aperiodicidad: Sea $i, j \in \mathcal{S}$ con $f(i) < f(j)$ y $G_{ij} > 0$. De la condición (1) el par i, j

siempre existe con tal que $S \neq S_{opt}$. Entonces $A_{ij}(c) < 1$, y por lo tanto se tiene

$$\begin{aligned} P_{ii}(c) &= 1 - \textstyle\sum_{l \in S, l \neq i} G_{il} A_{il}(c) \\ &= 1 - G_{ij} A_{ij}(c) - \textstyle\sum_{l \in S, l \neq i,j} G_{il} A_{il}(c) \\ &> 1 - G_{ij} - \textstyle\sum_{l \in S, l \neq i,j} G_{il} \\ &= 1 - \textstyle\sum_{l \in S, l \neq i} G_{il} = 0. \end{aligned}$$

Por lo tanto, de acuerdo al Teorema 1 existe una distribución estacionaria única, ya que la cadena de Markov asociada es irreducible y aperiódica. Se aplica ahora el Lema 2 para completar la prueba mostrando que la distribución estacionaria definida por (19) y (20) es un vector con las propiedades siguientes: (i) es estocástico y (ii) sus componentes satisfacen la ecuación de balance dada por (13). (i) Inmediato. (ii) Es como sigue: note que la definición de las probabilidades de generación de (15) implica que $G_{ij} = G_{ji}$, puesto que $\chi_{(S_j)}(i) = 1 \Leftrightarrow \chi_{(S_i)}(j) = 1$ (Definición 1.1). Por lo tanto, la ecuación de balance de (13) se reduce a

$$q_i(c) A_{ij}(c) = q_j(c) A_{ji}(c), \quad \forall i, j \in S. \tag{21}$$

Usando la Definición 10 y las Ecuaciones (19) y (20) se sigue que

$$\begin{aligned} q_i(c) A_{ij}(c) &= \frac{1}{N_0(c)} \exp\left(\frac{-f(i)}{c}\right) \exp\left(-\frac{(f(j) - f(i))^+}{c}\right) \\ &= \tfrac{1}{N_0(c)} \exp\left(\tfrac{-f(j)}{c}\right) \exp\left(-\tfrac{(f(i)-f(j))+(f(j)-f(i))^+}{c}\right) \\ &= \tfrac{1}{N_0(c)} \exp\left(\tfrac{-f(j)}{c}\right) \exp\left(-\tfrac{(f(i)-f(j)^+}{c}\right) \\ &= q_j(c) A_{ji}(c), \end{aligned}$$

que completa la prueba del teorema.

La distribución estacionaria definida por (19) y (20) es idéntica a la distribución de (2.5) y (2.6), que fué obtenida por analogía con la física. Del Corolario 2.1,

$$\lim_{c \downarrow 0} \mathbf{q}(c) = \mathbf{q}^*, \tag{22}$$

donde las componentes de $\mathbf{q}^*$ están dadas por

$$q_i^* = \frac{1}{|\mathcal{S}_{opt}|}\chi_{(\mathcal{S}_{opt})}(i). \tag{23}$$

finalmente se obtiene

$$\lim_{c\downarrow 0}\lim_{k\to\infty} P_c\{X(k) = i\} = \lim_{c\downarrow 0} q_i(c) = q_i^*, \tag{24}$$

o

$$\lim_{c\downarrow 0}\lim_{k\to\infty} P_c\{X(k) \in \mathcal{S}_{opt}\} = 1. \tag{25}$$

Este resultado refleja la propiedad básica del algoritmo de recocido simulado, es decir, garantiza que el algoritmo encuentra una solución óptima asintóticamente.

La prueba de convergencia dada anteriormente sigue siendo válida si se reemplaza la Definición 9 por una más general siempre que las probabilidades de generación sean simétricas, es decir,

$$G_{ij} = G_{ji}, \ \ para\ toda\ i, j \in \mathcal{S}. \tag{26}$$

Similarmente, [LUN86] muestra que la generación de probabilidades puede definirse como

$$G_{ij} = \frac{1}{|\mathcal{S}_i|}\chi_{(\mathcal{S}_i)}(j), \ \ para\ toda\ i, j \in \mathcal{S}. \tag{27}$$

En este caso las componentes de la distribución estacionaria están dadas por

$$q_i(c) = \frac{|\mathcal{S}_i| \exp\left(\frac{-f(i)}{c}\right)}{\sum_{j\in\mathcal{S}} |\mathcal{S}_j| \exp\left(\frac{-f(j)}{c}\right)}. \tag{28}$$

Estas componentes convergen a las q_i^* dadas por (23) cuando $c \downarrow 0$.

Finalmente, una matriz de generación que satisface (26) y (27) implica que $|\mathcal{S}_i| = \Theta$, para toda $i \in \mathcal{S}$. Esto puede demostrarse directamente.

Ejemplo 1 *(El problema del agente viajero) Sea $(\mathcal{S}, f)$ denote una instancia del problema del agente viajero con n ciudades y sea $\mathcal{N}_2$ denote la estructura de vecindades 2-cambio. Entonces recordando a (15) y (1.9) tenemos*

$$G_{ij} = \frac{1}{(n-1)(n-2)}\chi_{(\mathcal{S}_i)}(j). \tag{29}$$

Además, como un resultado del mecanismo de 2-cambio, la condición (1) del Teorema 2 se satisface (vea Ejemplo 1.3 y Ejemplo 1.7). Consecuentemente, el algoritmo de recocido simulado deberá encontrar una ruta óptima, cuando se aplica al problema del agente viajero usando una estructura de vecindades 2-cambio con probabilidad de generación dada por (29) y probabilidad de aceptación dada por (16).

Como ya se mencionó en la sección 1, la convergencia asintótica del algoritmo de recocido simulado puede también probarse para una clase más general de probabilidades de generación y aceptación. Esto queda establecido en el siguiente teorema.

Teorema 3 *Sea (S, f) denote una instancia de un problema de optimización combinatoria, sean las probabilidades de transición asociadas con el algoritmo de recocido simulado definidas por (14) y sean las probabilidades de generación $G_{ij}(c)$ y las probabilidades de aceptación $A_{ij}(c)$ satifacen las siguientes condiciones:*

(G1)

$$\forall c > 0 \ \forall i, j \in S \ \exists p \geq 1, \ \exists \ l_0, l_1, ..., l_p \in S, \ con \ l_0 = i, l_p = j,$$
$$y \ G_{l_k l_{k+1}}(c) > 0, \ k = 0, 1, ..., p-1,$$

(G2)

$$\forall c > 0 \ \forall i, j \in S : \ G_{ij}(c) = G_{ji}(c),$$

(A1)

$$\forall c > 0 \ \forall i, j \in S : \ A_{ij}(c) = 1 \qquad si \ f(i) \geq f(j),$$

$$A_{ij}(c) \in (0,1) \ si \ f(i) < f(j),$$

(A2)

$$\forall c > 0 \ \forall i, j, k \in S \ con \ f(i) \leq f(j) \leq f(k) :$$

$$A_{ik}(c) = A_{ij}(c) A_{jk}(c),$$

(A3)

$$\forall i, j \in S \ con \ f(i) < f(j) : \ \lim_{c \downarrow 0} A_{ij}(c) = 0.$$

Entonces existe una distribución estacionaria $\mathbf{q}(c)$ *cuyas componentes están dadas por*

$$q_i(c) = \frac{A_{i_{opt} i}(c)}{\sum_{j \in S} A_{i_{opt} j}(c)}, \ para \ toda \ i \in S \tag{30}$$

y para cualquier $i_{opt} \in \mathcal{S}$. *Sin embargo se tiene*

$$\lim_{c \downarrow 0} q_i(c) = \frac{1}{|\mathcal{S}_{opt}|} \chi_{(\mathcal{S}_{opt})}(i). \tag{31}$$

Prueba La prueba de este teorema es similar a las pruebas del Teorema 2 y del Corolario 2.

El conjunto de condiciones dadas en el Teorema 3 es el conjunto más general de condiciones de probabilidades de aceptación y generación conocido en la literatura, excepto la por la condición (G1). Aunque las condiciones (A1)-(A3) del Teorema 3 definen una clase más general de probabilidades de aceptación que la forma exponencial dada en (16), es difícil construir expresiones analíticas para probabilidades de aceptación que satisfagan (A1)-(A3) pero que sean esencialmente diferentes de la forma exponencial en (16).

Como ya se mencionó, la condición (G1) del Teorema 3 puede reemplazarse por una condición más general. Debe notarse que está condición, similar a la condición (1) del Teorema 2, establece que la cadena de Markov asociada con la matriz de generación G es irreducible, si este no es el caso, la convergencia asintótica al subconjunto de soluciones globalmente óptimas puede probarse [FEL50] si la condición (G1) en el Teorema 3 se reemplaza por la siguiente condición, que es necesaria y suficiente:

$(G1')$

$$\forall i \in \mathcal{S} \;\; \exists i_{opt}, \;\; p \geq 1, \;\; \exists \;\; l_0, l_1, ..., l_p \in \mathcal{S},$$

$$con \;\; l_0 = i, l_p = i_{opt}, \;\; y$$

$$G_{l_k l_{k+1}} > 0, \;\; k = 0, 1, ..., p-1. \tag{32}$$

La condición $(G1')$ establece que debe ser posible construir una sucesión finita de transiciones con probabilidad de generación diferente de cero, que permita a partir de una solución arbitraria i llegar a alguna solución óptima i_{opt}.

Para la prueba de la validez de la condición $(G1')$, debe hacerse una distinción entre soluciones transitorias y recurrentes, donde una solución se llama transitoria si la probabilidad de que la cadena de Markov siempre regrese a esta solución es igual a cero y es recurrente si la cadena de Markov puede regresar a esta solución con una probabilidad mayor que cero [FEL50]. Además, se menciona que en este caso la distribución estacionaria de (30) no siempre existe y debe reemplazarse por una *matriz estacionaria* $Q(c)$ cuyos elementos q_{ij} denoten la probabilidad de encontrar una solución j después de un número infinito de transiciones, comenzando desde una solución i.

4.3 CADENAS DE MARKOV NO HOMOGÉNEAS

Las pruebas de convergencia asintótica, dadas en la sección anterior, indican que el algoritmo de recocido simulado requiere un número infinito de transiciones para aproximarse, tanto como se quiera, a la distribución estacionaria. Una cota más precisa del número de transiciones se da en la Sección 4.4. En este sentido, la implantación del algoritmo requiere de sucesiones infinitamente largas de cadenas de Markov homogéneas con valores decendientes del parámetro de control c. Esto es impráctico; sin embargo, se pueden hacer ligeras mejorías, es decir, se puede describir el algoritmo de recocido simulado como una sucesión de cadenas de Markov homogéneas de longitud finita, generadas con valores descendientes del parámetro de control. Este proceso puede describirse combinando las cadenas de Markov homogéneas en una sóla cadena de Markov no homegénea. De esta manera la sucesión infinita de cadenas de Markov homogéneas, se reduce a una cadena de Markov no homegénea de longitud infinita. A continuación se presentan algunas definiciones y resultados de cadenas de Markov no homogéneas

Definición 11 *Sea* $\mathbf{P}(k)$ *la matriz de transición asociada con una cadena de Markov no homogénea. Entonces la matriz* $U(m,n)$ *está definida como*

$$U(m,n) = \prod_{k=m}^{n} \mathbf{P}(k), \;\; 0 < m \leq n. \tag{33}$$

Es decir, las componentes de $U(m,n)$ *son iguales a*

$$U_{ij}(m,n) = P\{X(n) = j | X(m-1) = i\}. \tag{34}$$

Definición 12 *Sea* $\mathbf{P}$ *una matriz estocástica de* $n \times n$, *entonces el* coeficiente de ergodicidad $\tau_1(\mathbf{P})$ *está definido como*

$$\tau_1(\mathbf{P}) = \frac{1}{2} \max_{i,j} \sum_{k=1}^{n} |P_{ik} - P_{jk}| = 1 - \min_{i,j} \sum_{k=1}^{n} min(P_{ik}, P_{jk}). \tag{35}$$

Definición 13 *Una cadena de Markov finita y no homogénea es* débilmente ergódica *si*

$$\forall i,j,l \in \mathcal{S}, \forall m > 0 : \lim_{k \to \infty} (U_{il}(m,k) - U_{jl}(m,k)) = 0. \tag{36}$$

Definición 14 *Una cadena de Markov finita y no homogénea es* fuertemente ergódica *si existe un vector aleatorio* $\mathbf{q}^*$, *tal que*

$$\forall i,j \in \mathcal{S}, \forall m > 0 : \lim_{k \to \infty} U_{ij}(m,k) = q_j^*. \tag{37}$$

Por lo tanto, para un valor dado m, el concepto de ergodicidad débil implica que $X(k)$ es independiente de $X(m)$, cuando $k \to \infty$, mientras que ergodicidad fuerte implica *convergencia en distribución*, es decir,

$$\lim_{k\to\infty} \mathbf{a}^T(m-1) \prod_{n=m}^{k} \mathbf{P}(n) = (\mathbf{q}^*)^T, \tag{38}$$

o bien

$$\begin{aligned} &\lim_{k\to\infty} P\{X(k) = j\} \\ &\quad = \lim_{k\to\infty}(\textstyle\sum_{i\in S} P\{X(k) = j | X(m-1) = i\} P\{X(m-1) = i\}) \\ &\quad = \lim_{k\to\infty}(\textstyle\sum_{i\in S} U_{ij}(m,k) P\{X(m-1) = i\}) \\ &\quad = q_j^*(\textstyle\sum_{i\in S} P\{X(m-1) = i\}) = q_j^*. \end{aligned} \tag{39}$$

Note que, para una cadena de Markov homogénea, no hay distinción entre ergodicidad fuerte y débil.

Los teoremas siguientes dan condiciones para la ergodicidad débil y fuerte de las cadenas de Markov no homogéneas.

Teorema 4 (Seneta, 1981) *Una cadena de Markov no homogénea es débilmente ergódica si y sólo si existe una sucesión de números positivos estrictamente creciente* $\{k_i\}$, $i = 0, 1, 2, \ldots$, *tales que*

$$\sum_{i=0}^{\infty} (1 - \tau_1(U(k_i, k_{i+1})) = \infty. \tag{40}$$

Teorema 5 (Isaacson y Madsen, 1976) *Una cadena de Markov no homogénea y finita es fuertemente ergódica bajo las siguientes condiciones: (i) la cadena de Markov es débilmente ergódica, (ii) para toda k existe un vector estocástico $\mathbf{q}(k)$ tal que $\mathbf{q}(k)$ es un vector propio izquierdo de $\mathbf{P}(k)$ con valor propio igual a 1. (iii) el vector propio $\mathbf{q}(k)$ satisface*

$$\sum_{k=1}^{\infty} \|\mathbf{q}(k) - \mathbf{q}(k+1)\| < \infty. \tag{41}$$

Sin embargo, si $\mathbf{q}^* = \lim_{k\to\infty} \mathbf{q}(k)$, entonces $\mathbf{q}^*$ es el vector en la Definición 14.

4.4 CONVERGENCIA ASINTÓTICA II

En esta sección se define una cadena de Markov no homogénea asociada al algoritmo de recocido simulado y se prueba convergencia asintótica del proceso

Definición 15 *Sea c_l' denote el valor del parámetro de control en la l-ésima cadena de Markov homogénea, L denote la longitud de la cadena de Markov homogénea y c_k denote el valor del parámetro de control en el k-ésimo ensayo. Defina la sucesión $\{c_k\}, k = 1, ...$ como sigue:*

$$c_k = c_l', \ lL < k \leq (l+1)L. \tag{42}$$

esto es, el parámetro de control se toma constante por pedazos. Además la sucesión $\{c_l'\}, l = 0, 1, ...$ es tal que satisface las condiciones siguientes:

$$c_{l+1}' \leq c_l', l = 0, 1, ... \tag{43}$$

y

$$\lim_{l \to \infty} c_l' = 0. \tag{44}$$

Claramente, las longitudes de las cadenas de Markov homogéneas pueden escojerse, en general, de longitud variable; sin embargo, una longitud fija no impone mayores restricciones, únicamente simplifica la notación.

Ahora se puede probar que bajos ciertas condiciones en la tasa de convergencia de la sucesión $\{c_l'\}$, la cadena de Markov no homogénea asociada con el algoritmo de recocido simulado converge en distribución a la distribución $\mathbf{q}^*$, cuyas componentes están dadas por (23). O de forma equivalente se cumple que

$$\lim_{k \to \infty} P\{X(k) = i\} = q_i^* = \frac{1}{|S_{opt}|} \chi_{(S_{opt})}(i). \tag{45}$$

Este es el objetivo del siguiente teorema.

Teorema 6 *Sea (S, f) denote una instancia de un problema de optimización combinatoria y sea $\mathbf{P}(k)$ denote la matriz de transición asociada con el algoritmo de recocido simulado definida por (14), (15) y (16). Además, se cumplan las siguientes condiciones:*

(1)

$$\forall i, j \in S \ \exists p \geq 1, \ \exists l_0, l_1, ...l_p \in S,$$
$$l_0 = i, l_p = j, \ y$$

$$G_{l_k l_{k+1}} > 0, \quad k = 0, 1, ...p-1. \tag{46}$$

(2) La sucesión $\{c_l'\}$ satisface la desigualdad siguiente:

$$c_l' \geq \frac{(L+1)\Delta}{\log(l+2)}, \quad l = 0, 1, ..., \tag{47}$$

donde

$$\Delta = \max_{i,j \in \mathcal{S}} \{f(j) - f(i) | j \in \mathcal{S}_i\} \tag{48}$$

y L se escoge como el máximo del mínimo número de transiciones requeridas para alcanzar una solución i_{opt} desde j, para toda $j \in \mathcal{S}$; esta L siempre existe como resultado de la condición (1).
Entonces la cadena de Markov converge en distribución al vector $\mathbf{q}^*$, con componentes

$$q_i^* = \frac{1}{|\mathcal{S}_{opt}|} \chi_{(\mathcal{S}_{opt})}(i), \quad \forall i \in \mathcal{S}, \tag{49}$$

o en otras palabras

$$\lim_{k \to \infty} P\{X(k) \in \mathcal{S}_{opt}\} = 1. \tag{50}$$

Prueba Para probar este teorema se usará el Teorema 5 y se mostrará que las condiciones (1) y (2) son suficientes para satisfacer las condiciones (i), (ii) y (iii) del Teorema 5.

(i) Las desigualdades siguientes se pueden probar directamente:

$$(A) \quad \forall k > 0, \forall i \in \mathcal{S}, j \in \mathcal{S}_i - \{i\} : P_{ij}(k) \geq \frac{1}{\Theta} \exp\left(\frac{-\Delta}{c_k}\right)$$

y

$$(B) \quad \exists k_0 > 0 : \forall k > k_0, \forall i \in \mathcal{S} - \hat{\mathcal{S}}' : P_{ii}(k) \geq \frac{1}{\Theta} \exp\left(\frac{-\Delta}{c_k}\right),$$

donde Δ está dado por (48) y $\hat{\mathcal{S}}'$ denota el conjunto de soluciones localmente maximales. De hecho, (A) se sigue directamente de las definiciones de $P_{ij}(k)$ y Δ, dadas en (14), (15) y (16) y (48) respectivamente. (B) se obtiene como sigue. Sea $\delta = \min^+_{i,j \in \mathcal{S}} \{f(j) - f(i) | j \in \mathcal{S}_i\}$, donde $\min^+$ denota que el mínimo se toma solamente sobre los términos positivos. Entonces siempre es posible encontrar una $k_0 > 0$ tal que

$$1 - \exp\left(\frac{-\delta}{c_k}\right) > \exp\left(\frac{-\Delta}{c_k}\right), \quad \forall k > k_0,$$

En consecuencia, se sigue directamente que

$$\forall k > k_0, \forall i \in \mathcal{S} - \hat{\mathcal{S}}' : P_{ii}(k) = 1 - \sum_{j \in \mathcal{S}_i - \{i\}} P_{ij}(k)$$

$$= 1 - \textstyle\sum_{j \in \mathcal{S}_i - \{i\}} \frac{1}{\Theta} A_{ij}(c_k) \geq 1 - \frac{1}{\Theta}\left\{\Theta - 1 - \exp\left(\frac{-\delta}{c_k}\right)\right\}$$

$$= \frac{1}{\Theta}\left\{1 - \exp\left(\frac{-\delta}{c_k}\right)\right\} > \frac{1}{\Theta}\exp\left(\frac{-\Delta}{c_k}\right).$$

Ahora, sea $p \in \mathcal{S}$ tal que $U_{pi_{opt}}(k, k+L) = \min_{i \in \mathcal{S}} U_{ii_{opt}}(k, k+L)$. Entonces como resultado de la forma de escoger L, se tiene que para toda $k \geq k_0$

$$\begin{aligned}
&\min_{i \in \mathcal{S}} U_{ii_{opt}}(k, k+L) \\
&\quad = P\{X(k+L) = i_{opt} | X(k-1) = p\} \\
&\quad = \textstyle\sum_{j_1 \in \mathcal{S}} \sum_{j_2 \in \mathcal{S}} \cdots \sum_{j_L \in \mathcal{S}} P_{pj_1}(k) P_{j_1 j_2}(k+1) \cdots P_{j_L i_{opt}}(k+L) \\
&\quad \geq P_{pl_1}(k) P_{l_1 l_2}(k+1) \cdots P_{l_L i_{opt}}(k+L) \\
&\quad \geq \left[\frac{1}{\Theta}\exp\left(\frac{-\Delta}{c_k}\right)\right]^{L+1}.
\end{aligned} \tag{51}$$

Así, se obtiene:

$$\begin{aligned}
&1 - \tau_1(U(k, k+L)) \\
&\quad = \min_{i,j \in \mathcal{S}} \textstyle\sum_{p \in \mathcal{S}} \min\{U_{ip}(k, k+L), U_{jp}(k, k+L)\} \\
&\quad \geq \min_{i \in \mathcal{S}} U_{ii_{opt}}(k, k+L) \\
&\quad \geq \Theta^{-L-1} \exp\left(\frac{-(L+1)\Delta}{c_k}\right), \quad \forall k \geq k_0.
\end{aligned}$$

La ergodicidad débil se sigue directamente del Teorema 4 tomando $k_l = lL$, de tal

manera que se obtiene

$$\begin{aligned}\sum_{l=0}^{\infty}(1-\tau_1(U(k_l,k_{l+1}))) &\geq \sum_{l=k_0}^{\infty}(1-\tau_1(U(lL,lL+L)))\\ &\geq \sum_{l=k_0}^{\infty}\Theta^{-L-1}\exp\left(\frac{-(L+1)\Delta}{c_l'}\right)\\ &\geq \sum_{l=k_0}^{\infty}\Theta^{-L-1}(l+2)^{-1}=\infty.\end{aligned}$$

(ii) La condición (1) garantiza la existencia del vector propio izquierdo $\mathbf{q}(k)$ de $\mathbf{P}(k)$, dado por $\mathbf{q}(k)=\mathbf{q}(c_k)$, es decir, la distribución estacionaria de la cadena de Markov homogénea con matriz de transición $\mathbf{P}=\mathbf{P}(k)$; vea Teorema 2. Para la matriz de transición asociada con el algoritmo de recocido simulado definida por (14), (15) y (16), las componentes del vector propio están dadas por (19) y (20).

(iii) Del Corolario 2.4 se tiene que

$$\forall i\in\mathcal{S}_{opt}: q_i(c_{k+1})-q_i(c_k)>0.$$

y

$$\exists k_1:\forall k>k_1,\forall i\in\mathcal{S}-\mathcal{S}_{opt}: q_i(c_{k+1})-q_i(c_k)<0.$$

Consecuentemente, se obtiene

$$\begin{aligned}\forall k\geq k_1:\|\mathbf{q}(k)-\mathbf{q}(k+1)\| &\equiv \sum_{i\in\mathcal{S}}|q_i(c_k)-q_i(c_{k+1})|\\ &=\sum_{i\in\mathcal{S}_{opt}}(q_i(c_{k+1})-q_i(c_k))-\sum_{i\in\mathcal{S}-\mathcal{S}_{opt}}(q_i(c_{k+1})-q_i(c_k)).\end{aligned}$$

Sustituyendo $\sum_{i\in\mathcal{S}-\mathcal{S}_{opt}}q_i(c)=1-\sum_{i\in\mathcal{S}_{opt}}q_i(c)$ entonces se obtiene

$$\forall k>k_1:\|\mathbf{q}(k)-\mathbf{q}(k+1)\|=2\left(\sum_{i\in\mathcal{S}_{opt}}q_i(c_{k+1})-\sum_{i\in\mathcal{S}_{opt}}q_i(c_k)\right).$$

y por lo tanto

$$\sum_{k=k_1+1}^{\infty}\|\mathbf{q}(k)-\mathbf{q}(k+1)\|=2\left(\sum_{i\in\mathcal{S}_{opt}}q_i(c_\infty)-\sum_{i\in\mathcal{S}_{opt}}q_i(c_{k_1+1})\right)\leq 2.$$

Así, (41) puede escribirse como

$$\sum_{k=1}^{\infty} \|\mathbf{q}(k) - \mathbf{q}(k+1)\|$$

$$= \sum_{k=1}^{k_1} \|\mathbf{q}(k) - \mathbf{q}(k+1)\| + \sum_{k=k_1+1}^{\infty} \|\mathbf{q}(k) - \mathbf{q}(k+1)\| \leq 2k_1 + 2.$$

Por lo tanto, la suma $\sum_{k=1}^{\infty} \|\mathbf{q}(k) - \mathbf{q}(k+1)\|$ es finita y se completa la prueba de la parte (iii).

Finalmente, sabemos del Corolario 2.1 que $\lim_{k\to\infty} \mathbf{q}(k) = \mathbf{q}^*$, donde las componentes del vector $\mathbf{q}^*$ están dadas por (49) puesto que $\lim_{k\to\infty} c_k = \lim_{l\to\infty} c_l' = 0$. Con esto se completa la prueba del teorema.

En conclusión, se ha mostrado que el algoritmo de recocido simulado, formulado como una sucesión de cadenas de Markov homogénas de longitud finita, converge en distribución al conjunto de soluciones óptimas, siempre y cuando la temperatura se baje poco a poco; vea condición (2) del Teorema 6.

Note que las condiciones de convergencia asintótica dadas anteriormente son *suficientes* pero no *necesarias*. Condiciones necesarias y suficientes han sido dadas por Hajek [HAJ88], para discutir este resultado se necesitan las siguientes definiciones.

Definición 16 *Sean $i, j \in S$, entonces j es* alcanzable en altura h *desde i si $\exists p \geq 1, \exists l_0, ..., l_p \in S$ con $l_0 = i$ y $l_p = j$, tal que $G_{l_k,l_{k+1}} > 0$ y $f(l_k) \leq h$, para toda $k = 0, ..., p-1$.*

Definición 17 *Sea $\hat{i}$ un mínimo local, entonces la* profundidad $d(\hat{i})$ *de $\hat{i}$ es la más pequeña h tal que hay una solución $j \in S$ con $f(j) < f(\hat{i})$ que es alcanzable en altura $f(\hat{i}) + h$ desde $\hat{i}$. Por definición $d(i_{opt}) = \infty$.*

Ahora se discutirá el resultado obtenido por Hajek.

Teorema 7 (Hajek,1988) *Sea $\{c_k\}$ una sucesión de valores del parámetro de control semejante a la sucesión dada en (47), es decir,*

$$c_k = \frac{\Gamma}{\log(k+2)}, \quad k = 0, 1, ..., \tag{52}$$

para alguna constante Γ. Entonces la convergencia asintótica del algoritmo de recocido simulado, usando las probabilidades de transición de (14), (15) y (16), está garantizada si y sólo si

(i) la condición (1) del Teorema 6 se cumple,

(ii) i es alcanzable desde j en altura h si y sólo si j es alcanzable desde i en altura h, para arbitrarios $i, j \in S$ y h

(iii) la constante Γ satisface $\Gamma \geq D$, donde

$$D = \max_{\hat{i} \in S - S_{opt}} d(\hat{i}), \tag{53}$$

es decir, D es la profundidad del mínimo local más profundo, no del mínimo global.

Del hecho que las condiciones de Hajek son necesarias y suficientes se concluye que $D \leq \Delta$ donde Δ está dada en (48). Cotas para los valores de D han sido obtenidas en para varios problemas de optimización combinatoria.

Finalmente se menciona que bajo ciertas condiciones, la convergencia asintótica de la cadena de Markov no homogénea asociada con el algoritmo de recocido simulado puede probarse para condiciones generales en las probabilidades de generación y aceptación; también vea Teorema 3.

Teorema 8 (Anily y Federgruen,1987) *Sean las probabilidades de transición asociadas con el algoritmo de recocido simulado definidas por (14) y sean las probabilidades de generación $G_{ij}(c)$ y probabilidades de aceptación $A_{ij}(c)$ satisfagan las condiciones (G1)-(A3) del Teorema 3. Además, sea*

$$A(c) = \min_{i,j}\{A_{ij}(c) | i \in S, j \in S_i\}.$$

Entonces,

$$\lim_{k \to \infty} P\{X(k) \in S_{opt}\} = 1,$$

si

$$\sum_{k=0}^{\infty} (A(c_k'))^{L+1} = \infty, \tag{54}$$

donde la sucesión $\{c_k'\}$ está definida por (42) y L denota número mínimo de transiciones para alcanzar una solución óptima desde cualquier solución dada; vea también la condición (2) del Teorema 6.

Aplicando este resultado al caso especial donde las probabilidades de aceptación están definidas por (16) se tiene que la convergencia asintótica al conjunto de soluciones globalmente óptimas se obtiene si la sucesión $\{c_k'\}$ satisface la condición (2) del Teorema 6.

4.5 COMPORTAMIENTO ASINTÓTICO

En la sección anterior se ha demostrado que el algoritmo de recocido simulado converge en probabilidad al conjunto de soluciones óptimas, en otras palabras, asintóticamente el algoritmo encuentra una solución óptima con probabilidad uno. Como resultado de los límites de (25) o (50), el algoritmo asintóticamente produce una solución que converge al conjunto de soluciones óptimas y esto se alcanza solamente después de un número infinito de transiciones. En cualquier implementación práctica esto es claramente imposible de realizar y se debe conformar con una aproximación de la convergencia asintótica.

Enseguida, se discuten algunas propiedades que pueden derivarse del *comportamiento* asintótico del algoritmo de recocido simulado basado en el marco de referencia matemático presentado en las secciones anteriores. Las primeras dos propiedades que se discutirán están relacionadas con la aproximación de la distribución estacionaria. La tercera propiedad se relaciona con la aproximación de la distribución uniforme $\mathbf{q}^*$ sobre el conjunto de soluciones globalmente óptimas.

Con respecto a la aproximación de la distribución estacionaria, tenemos las siguientes dos propiedades.

Propiedad 1 *Sea* $\mathbf{P}(c)$ *denote la matriz de transición de la cadena de Markov homogénea asociada con el algoritmo de recocido simulado definida por (14) y* $\mathbf{q}(c)$ *denote la distribución estacionaria correspondiente dada por el vector propio izquierdo con valor propio 1 de* $\mathbf{P}(c)$. *Entonces, cuando* $k \to \infty$, *se tiene*

$$\|\mathbf{a}(k) - \mathbf{q}(c)\| = O(k^s |\lambda_2(c)|^k), \tag{55}$$

donde $\lambda_2(c)$ $(0 < |\lambda_2(c)| < 1)$ *denota el segundo valor propio más grande de* $\mathbf{P}(c)$ *con multiplicidad* m_2 *y* $s = m_2 - 1$.

Por lo tanto, la velocidad de convergencia de la distribución estacionaria está determinada por $\lambda_2(c)$. Desafortunadamente, el calculo de $\lambda_2(c)$ es imposible, desde el punto de vista práctico, debido al tamaño de la matriz $\mathbf{P}(c)$. Aproximación de la norma en (55) conduce a la siguiente propiedad.

Propiedad 2 *Sea* ϵ *denote un número positivo arbitrariamente pequeño, entonces*

$$\|\mathbf{a}(k) - \mathbf{q}(c)\| < \epsilon \tag{56}$$

si

$$k > K\left(1 + \frac{\ln(\frac{1}{2}\epsilon)}{\ln(1 - \gamma^K(c))}\right) \tag{57}$$

donde $\gamma(c) = \min^{+}_{i,j \in S} P_{ij}(c)$ *y* $K = |S|^2 - 3|S| + 3$.

Por lo tanto (56) y (57) indican que se puede acercar tanto como se desee a la distribución estacionaria, solamente si el número de transiciones es al menos cuadrático en el tamaño del espacio de soluciones. Sin embargo, el tamaño $|S|$ para muchos problemas es exponencial en el tamaño del problema, por ejemplo, para las instancias del problema del agente viajero $|S| = (n-1)!$, donde n denota el número de ciudades. Por lo tanto, el análisis presentado arriba indica que la aproximación a la distribución estacionaria se alcanza en un tiempo exponencial de la ejecución del algoritmo de recocido simulado.

Con respecto a la convergencia asintótica de la cadena de Markov no homogénea asociada al algoritmo de recocido simulado se tiene el siguiente resultado.

Propiedad 3 *Sea la matriz de transición del la cadena de Markov no homogénea asociada con el algoritmo de recocido simulado definida por (14), (15) y (16) y sea la sucesión* $\{c_l'\}$ *dada por (47), es decir,*

$$c_l' = \frac{(L+1)\Delta}{\log(l+2)}, \quad l = 0, 1, ..., \tag{58}$$

donde L *y* Δ *están definidos como en el Teorema 6, y sea* $\mathbf{q}^*$ *la distribución uniforme de probabilidad en el conjunto de soluciones óptimas definido por (49). Entonces cuando* $k \to \infty$, *se tiene*

$$\|\mathbf{a}(k) - \mathbf{q}^*\| < \epsilon, \tag{59}$$

para un valor de ϵ *positivo arbitrariamente pequeño, si*

$$k = O\left(\left(\frac{1}{\epsilon}\right)^{\frac{1}{\min(a,b)}}\right), \tag{60}$$

donde

$$a = \frac{1}{(L+1)\Theta^{L+1}} \;\; y \;\; b = \frac{\hat{f} - f_{opt}}{(L+1)\Delta}, \tag{61}$$

con $\hat{f} = \min_{i \in S - S_{opt}} f(i)$.

La evaluación de esta cota para instancias de problemas particulares típicamente permite un número de transiciones que es más grande que el tamaño del espacio de soluciones y por lo tanto el tiempo de ejecución es exponencial para la mayoría de los problemas. Esto se puede ilustrar con el ejemplo siguiente.

Ejemplo 2 *Sea (S, f) denote una instancia del problema del agente viajero con n ciudades y $\mathcal{N}_2$ denote la estructura de vecindades con dos cambios. Entonces de acuerdo al Ejemplo 1.7 se tiene que $L = n-2$ y $\Theta = (n-1)(n-2)$. Por lo tanto, (61) se convierte en*

$$a \approx \frac{1}{n-1}\left(\frac{1}{(n-1)(n-2)}\right)^{n-1} \quad y \quad b < \frac{1}{n-1}, \tag{62}$$

donde n denota el número de ciudades. Ahora, usando el hecho que $a \ll b$ y escogiendo $\frac{1}{\epsilon} = n$ y de (60) se tiene que

$$k = O\left(n^{n^{2n-1}}\right), \tag{63}$$

mientras que $|S| = O((n-1)!)$. Así, hacer la enumeración de todas las soluciones toma menos tiempo que obtener una aproximación de la solución arbitrariamente cercana por medio del algoritmo de recocido simulado.

En suma, se ha mostrado que el algoritmo de recocido simulado se comporta como un algoritmo de optimización únicamente si se ejecuta con un número infinito de transiciones. Aproximarse arbitrariamente cerca de la distribución estacionaria, requiere de un número de transiciones, que en la mayoría de los problemas es más grande que el espacio de soluciones, dando como resultado que el algoritmo se ejecute en un tiempo exponencial. En conclusión, el algoritmo de recocido simulado resulta impráctico para la solución de problemas combinatorios que requieran obtener el mínimo global. Sin embargo, en el capítulo siguiente se mostrará como el comportamiento asintótico del algoritmo de recocido simulado puede aproximarse en tiempo polinomial. Evidentemente, el costo de esto es que, no se puede garantizar que se obtenga la solución óptima, pero se mostrará que el algoritmo de recocido simulado, usando aproximación polinomial obtiene soluciones cercanas a la óptima para la mayoría de las instancias.

CAPÍTULO 5

APROXIMACIÓN EN TIEMPO FINITO

El algoritmo de recocido simulado ha sido presentado y sus principales módulos analizados, en particular, el aspecto relacionado con la convergencia bajo ciertas condiciones. Como se ha visto, en el Capítulo 4, la implementación requiere de un número infinito de transiciones. Sin embargo, para propósitos prácticos se requiere de una implantación que converja en un número finito de pasos. Este capítulo se centra en la implementación del algoritmo de recocido simulado en un tiempo finito, de manera que el algoritmo resultante dará una aproximación de la solución óptima del problema de optimización combinatoria tratada.

El capítulo se desarrolla como sigue: En la Sección 5.1 se discuten algunos rasgos y características de los programas de enfriamiento. Además se presenta un programa de enfriamiento que se usa frecuentemente en la literatura. En la Sección 5.2 se analiza un programa de enfriamiento que permite una ejecución del algoritmo en tiempo polinomial.

5.1 PROGRAMA DE ENFRIAMIENTO

Una implementación en tiempo finito del algoritmo de recocido simulado puede realizarse generando una cadena de Markov homogénea de longitud finita, para una sucesión finita de valores decrecientes del parámetro de control. En la implantación se deben especificar un conjunto de parámetros que gobiernen la convergencia del algoritmo. Estos parámetros se combinan en el llamado programa de enfriamento. (*cooling schedule*)

Definición 1 *Un* programa de enfriamiento *se especifica por:*

- *una sucesión finita de valores del parámetro de control, es decir.*
  - *Un* valor inicial c_0 *del parámetro de control*
  - *Una* función decreciente *que disminuya el valor del parámetro de control,*
  - *Un* valor final *del parámetro de control que especifique el* criterio de paro,

y

- *Un número finito de transiciones de cada valor del parámetro de control, es decir,*
  - *Una* longitud *finita de cada cadena de Markov homogénea.*

El punto clave del programa de enfriamiento es el concepto de cuasi equilibrio, que se define como sigue.

Definición 2 *Sea L_k denote la longitud de la k-ésima cadena de Markov y c_k el correspondiente valor del parámetro de control. Entonces se alcanza el* cuasi equilibrio *si* $\mathbf{a}(L_k, c_k)$, *es decir, la distribución de probabilidad de las soluciones después de L_k ensayos de la k-ésima cadena de Markov, es "suficiente cercana" a* $\mathbf{q}(c_k)$, *la distribución estacionaria en c_k, definida por (4.19) y (4.20). Equivalentemente se cumple,*

$$\| \mathbf{a}(L_k, c_k) - \mathbf{q}(c_k) \| < \epsilon \tag{1}$$

para algún valor especificado de ϵ

En el capítulo anterior se observó que (1) se cumple para valores arbitrariamente pequeños de ϵ si el número de transiciones necesarios es cuadrático en el tamaño del espacio de soluciones. Esto conduce, para muchos problemas de optimización combinatoria a una ejecución en tiempo exponencial del algoritmo de recocido simulado. Por lo tanto, para aplicaciones prácticas se necesita una cuantificación menos rígida del cuasi equilibrio que la dada en la condición (1). En la literatura, esto da lugar a diferentes interpretaciones del concepto de cuasi equilibrio, dando como resultado una gran variedad de programas de enfriamiento. En la siguiente sección se dará una cuantificación más precisa.

La construcción de un programa de enfriamiento, usando el concepto de cuasi equilibrio, se basa en los argumentos siguientes. Si la probabilidad de aceptación está definida por (4.16), entonces para $c \to \infty$, la distribución estacionaria viene dada por la distribución uniforme discreta en el conjunto finito de soluciones S, es decir:

$$\lim_{c \to \infty} \mathbf{q}(c) = \frac{1}{|S|}\mathbf{1} \tag{2}$$

donde $\mathbf{1}$ denota un vector de $|S|$ componentes todas iguales a 1. La ecuación (2) se sigue en forma inmediata a partir de (4.19) y (4.20). Por lo tanto, si se escoge un valor de c_k suficientemente grande, (que permita aceptar cualquier propuesta) se obtiene directamente el cuasi equilibrio para estos valores del parámetro de control, ya que en este caso todas las soluciones ocurren con igual probabilidad. Después se debe de escoger la longitud de la cadena de Markov y la función decreciente tal que las condiciones de cuasi-equilibrio

se restablezcan al final de cada cadena de Markov. De esta forma, las distribuciones de equilibrio para las diferentes cadenas de Markov deberán estar "cercanas" de manera que eventualmente llegarán a $\mathbf{q}^*$, la distribución uniforme en el conjunto de soluciones óptimas, cuando $c_k \downarrow 0$; vea (2.7) o (4.20).

Es intuitivamente claro que grandes decrementos a c_k requierán grandes longitudes de cadenas de Markov para restablecer el cuasi equilibrio al siguiente valor del parámetro de control, c_{k+1}. Por lo tanto, a pequeños decrementos en el parámetro c_k la longitud de las cadenas de Markov será pequeña.

La búsqueda de un programa de enfriamiento adecuado ha sido el objetivo de muchos artículos publicados en los últimos años. En el resto de esta sección se discutirá el programa de enfriamiento propuesto por Kirkpatrick, Gelatt y Vecchi [KIR83]. Este programa de enfriamiento se ha usado en muchas aplicaciones del algoritmo de recocido simulado y se basa en un número de reglas conceptualmente simples. El programa de enfriamiento propuesto por Kirkpatrick se centra en los parámetros de la Definición 1.

Valor inicial del parámetro de control

Como ya se mencionó, el valor de c_0 debe ser suficientemente grande para permitir que todas las transiciones sean factibles de ser aceptadas. Esto se logra empezando con un valor positivo pequeño c_0 y multiplicandolo por un factor constante mayor que 1, hasta que el criterio de aceptación este cercano a 1. Pensando en la analogía física del Capítulo 2, esto corresponde al calentamiento del sólido hasta que todas sus partículas se acomoden aleatoriamente.

Decremento del parámetro de control

En forma general se opta por hacer pequeños cambios en el valor del parámetro de control y se usa frecuentemente una función decreciente dada por

$$c_{k+1} = \alpha c_k, \quad k = 1, 2, \ldots, \tag{3}$$

donde α es una constante menor que 1 pero suficientemente cercana a 1. Los valores típicos están entre 0.8 y 0.99.

Valor final del parámetro de control

La ejecución del algoritmo finaliza, si el valor de la función de costo de la solución obtenida en el último ensayo de la cadena de Markov, permanece sin cambio para un número de cadenas consecutivas.

Longitud de la cadena de Markov

La longitud de la cadena de Markov se basa en el requerimiento de que para cada valor del parámetro de control c_k el cuasi equilibrio se restablezca. El número de transiciones necesarias para alcanzar esto, se calcula de un argumento intuitivo de cuasi equilibrio que deberá restablecerse después de aceptar al menos algún número fijo de transiciones. Sin embargo, puesto que las transiciones se aceptan con probabilidad decreciente, se debe obtener $L_k \rightarrow \infty$ para $c_k \downarrow 0$. Consecuentemente, L_k está acotada por alguna constante $\overline{L}$ que evite cadenas de Markov extremadamente grandes para valores pequeños de c_k

5.2 UN ENFRIAMIENTO POLINOMIAL

En esta sección se discute un programa de enfriamiento polinomial presentado por Aarts y Van Laarhoven [AAR89]. Este programa de enfriamiento permite una ejecución del algoritmo de recocido simulado en tiempo polinomial, pero no puede darse ninguna garantía de la diferencia de costo entre la solución final y el costo óptimo. La discusión se centra en la construcción de los cuatro parámetros del programa de enfriamiento, vea Definición 1.

Valor inicial del parámetro de Control

El valor inicial c_0 del parámetro de control se obtiene nuevamente a partir del requerimiento de que este valor debe de aceptar casi todas las transiciones propuestas. Suponga que una sucesión de ensayos se genera con un cierto valor c del parámetro de control. Sea m_1, el número de transiciones propuestos de i a j con $f(j) \leq f(i)$, y m_2 el número de transiciónes con $f(j) > f(i)$. Además, sea $\overline{\Delta f}^{(+)}$ el costo promedio de la diferencia sobre los m_2 transiciones. Entonces la tasa de aceptación puede aproximarse por la expresión siguiente:

$$\chi \approx \frac{m_1 + m_2 \exp\left(\frac{-\overline{\Delta f}^{(+)}}{c}\right)}{m_1 + m_2}, \tag{4}$$

de donde se obtiene:

$$c = \frac{\overline{\Delta f}^{(+)}}{\ln\left(\frac{m_2}{m_2\chi - m_1(1-\chi)}\right)}. \tag{5}$$

El valor inicial c_0 puede calcularse de (5) de la manera siguiente. Inicialmente, c_0 es igual a cero. Enseguida, una sucesión de m_0 ensayos se genera. Después de cada ensayo, un nuevo valor de c_0 se calcula de (5), donde χ se acerca a χ_0; de aquí en adelante, haremos referencia a χ_0 como la *tasa de aceptación inicial.* los valores de m_1 y m_2 corresponden al número de transiciones con costo decreciente y creciente respectivamente, obtenidas en la sucesión de ensayos actual; eventualmente $m_0 = m_1 + m_2$. Experimentos numéricos indican una convergencia rápida al valor final de c_0 obtenido siguiendo el procedimiento anterior. El valor final c_0 de este proceso, se toma como valor inicial del parámetro de control.

Decremento del parámetro de control

En la sección anterior se argumentó que para pequeños decrementos del parámetro de control, las distribuciones estacionarias subsecuentes de las cadenas de Markov homogéneas deben estar muy cercanas una de la otra. Como una consecuencia de esto, después de decrecer c_k a c_{k+1}, un número pequeño de transiciones debe ser suficiente para restablecer el cuasi equilibrio en c_{k+1}, si inicialmente existía el cuasi equilibrio en c_k. Suponiendo que la condición de cuasi equilibrio pueda ser reemplazada por

$$\forall k \geq 0 : \parallel \mathbf{q}(c_k) - \mathbf{q}(c_{k+1}) \parallel < \epsilon \tag{6}$$

para algún valor positivo ϵ. De modo que si suponemos que el cuasi equilibrio se mantiene a través del proceso de optimización si (6) se cumple para toda k. Evidentemente, esta suposición requiere que el cuasi equilibrio se alcance en c_0. Esto último puede lograrse si se escoje el valor inicial del parámetro de control.

Por lo tanto, para dos valores sucesivos del parámetro de control, se desea que la distribución estacionaria sea "cercana". Esto se puede cuantificar pidiendo que

$$\forall i \in \mathcal{S} : \frac{1}{1+\delta} < \frac{q_i(c_k)}{q_i(c_{k+1})} < 1 + \delta, \;\; k = 0, 1, ..., \tag{7}$$

para algún número δ pequeño, que puede relacionarse con ϵ a través de (6). El teorema siguiente da una condición suficiente para que se satisfaga (7).

Teorema 1 *Sea* $\mathbf{q}(c_k)$ *la distribución estacionaria de la cadena de Markov asociada con el algoritmo de recocido simulado con componentes dadas por (4.19) y (4.20), y sea*

c_k y c_{k+1} dos valores sucesivos del parámetro de control con $c_{k+1} < c_k$. Entonces las desigualdades en (7) se satisfacen si la siguiente condición se cumple:

$$\forall i \in S : \frac{\exp\left(-\frac{\delta_i}{c_k}\right)}{\exp\left(-\frac{\delta_i}{c_{k+1}}\right)} < 1 + \delta, \quad k = 0, 1, \ldots, \tag{8}$$

donde $\delta_i = f(i) - f_{opt}$.

Prueba: Las componentes de la distribución estacionaria dadas por (4.19) y (4.20) pueden reescribirse como

$$\forall i \in S : q_i(c_k) = \frac{1}{M_0(c_k)} \exp\left(-\frac{\delta_i}{c_k}\right),$$

donde la constante de normalización M_0 está dada por

$$M_0(c_k) = \sum_{j \in S} \exp\left(-\frac{\delta_j}{c_k}\right).$$

Así, si $c_{k+1} < c_k$, tenemos

$$\exp\left(-\frac{\delta_i}{c_k}\right) \geq \exp\left(-\frac{\delta_i}{c_{k+1}}\right)$$

y

$$M_0(c_k) \geq M_0(c_{k+1})$$

De donde se obtiene

$$\begin{aligned} \forall i \in S : q_i(c_k) &= \tfrac{1}{M_0(c_k)} \exp\left(-\tfrac{\delta_i}{c_k}\right) \leq \tfrac{1}{M_0(c_{k+1})} \exp\left(-\tfrac{\delta_i}{c_k}\right) \\ &< (1+\delta)\tfrac{1}{M_0(c_{k+1})} \exp\left(-\tfrac{\delta_i}{c_{k+1}}\right) = (1+\delta) q_i(c_{k+1}). \end{aligned}$$

Por otra parte, se tiene

$$\begin{aligned}
\forall i \in \mathcal{S} : q_i(c_k) &= \tfrac{1}{M_0(c_k)} \exp\left(-\tfrac{\delta_i}{c_k}\right) \\
&\geq \frac{1}{\sum_{j\in\mathcal{S}} \exp\left(-\frac{\delta_j}{c_k}\right)} \exp\left(-\tfrac{\delta_i}{c_{k+1}}\right) \\
&> \frac{1}{\sum_{j\in\mathcal{S}} (1+\delta) \exp\left(-\frac{\delta_j}{c_{k+1}}\right)} \exp\left(-\tfrac{\delta_i}{c_{k+1}}\right) \\
&= \tfrac{1}{(1+\delta)} q_i(c_{k+1}),
\end{aligned}$$

que completa la prueba.

La ecuación (8) puede reescribirse para dar la siguiente condición en dos subsecuentes valores del parámetro de control:

$$\forall i \in \mathcal{S} : c_{k+1} > \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{f(i) - f_{opt}}}, \quad k = 0, 1, ..., \tag{9}$$

Ahora se introducirá una simplificación restringiendo la condición (8) al conjunto de soluciones que ocurren con mayor probabilidad durante la generación de la k-ésima cadena de Markov. La distribución de probabilidad de los costos de las soluciones pueden aproximarse por una distribución normal en la región cercana al valor promedio de la función de costo y por una distribución exponencial en la región cercana al valor óptimo de la función de costo. Además la media $\langle f \rangle_{c_k}$ y la desviación estandar σ_{c_k} de estas distribuciones puede aproximarse por el costo promedio $\bar{f}(c_k)$ y el despliegue $\sigma(c_k)$, respectivamente. Ahora definimos el conjunto $\mathcal{S}_{c_k}$ como

$$\mathcal{S}_{c_k} = \{i \in \mathcal{S} | f(i) - f_{opt} \leq \langle f \rangle_{c_k} - f_{opt} + 3\sigma_{c_k}\}. \tag{10}$$

Así, debido a las propiedades de las distribuciones normal y exponencial, una solución en la k-ésima cadena de Markov tiene una probabilidad cercana a 1 de estar en el conjunto $\mathcal{S}_{c_k}$ (la probabilidad exacta es de 0.99 para la distribución normal y de 0.95 para la exponencial). Usando la simplificación descrita anteriormente, (9) puede reemplazarse por la siguiente condición:

$$c_{k+1} > \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{\langle f \rangle_{c_k} - f_{opt} + 3\sigma_{c_k}}}, \quad k = 0, 1, ..., \tag{11}$$

que es una condición fuerte para las soluciones generadas en la k-ésima cadena de Markov.

Para muchas instancias de problemas de optimización combinatoria, el valor de f_{opt} no es conocido. Sin embargo el valor promedio y el despliegue de la función de costos típicamente exhiben un comportamiento similar como una función del parámetro de control. Así, se argumenta que $\langle f \rangle_{c_k} - f_{opt} + 3\sigma_{c_k}$ puede reemplazarse por $3\sigma_{c_k}$ y que la omisión del término $\langle f \rangle_{c_k} - f_{opt}$ puede justificarse al escoger valores pequeños de δ.

Por lo tanto, (11) puede reemplazarse por la siguiente expresión:

$$c_{k+1} = \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{3\sigma_{c_k}}}, \quad k = 0, 1, ..., \tag{12}$$

La cantidad que decrementa el valor de c en la ecuación (12) está determinada por el valor de δ, que de ahora en adelante se llamará el *parámetro distancia.* Valores pequeños de δ permitirán pequeños decrementos en c; valores grandes de δ permitirán grandes decrementos en c.

Valor final del parámetro de control

La terminación del algoritmo se base en una extrapolación del costo esperado $\langle f \rangle_{c_k}$ para $c_k \downarrow 0$. Sea

$$\Delta\langle f \rangle_c = \langle f \rangle_c - f_{opt}, \tag{13}$$

entonces la ejecución del algoritmo termina si $\Delta\langle f \rangle_c$ es pequeño comparado con $\langle f \rangle_{c_0}$, el costo esperado de c_0. Para valores suficientemente grandes de c_0 tenemos $\langle f \rangle_{c_0} \approx \langle f \rangle_\infty$; vea (2.14). Así, se puede aproximar $\Delta\langle f \rangle_c$ para $c \ll 1$ por:

$$\Delta\langle f \rangle_c \approx c\frac{\partial\langle f \rangle_c}{\partial c}. \tag{14}$$

Por lo tanto, se puede finalizar el algoritmo si para alguna k se tiene

$$\frac{c_k}{\langle f \rangle_\infty} \frac{\partial\langle f \rangle_c}{\partial c}\bigg|_{c=c_k} < \epsilon_s, \tag{15}$$

donde ϵ_s es un número positivo pequeño. De aquí en adelante nos referiremos a ϵ_s como el *parámetro de paro* y a (15) como el *criterio de paro.*

En la práctica el valor de $\bar{f}(c_k)$, puede usarse para aproximar a $\langle f \rangle_{c_k}$, aunque se pueden presentar amplias variaciones, especialmente para valores grandes de c_k. Lo anterior

puede permitir que el criterio de paro se satisfaga dando una terminación prematura del algoritmo. Este problema se salva introduciendo suavizamientos de valor de $\bar{f}(c_k)$. Estos suavizamientos pueden lograrse reemplazando cada punto $(c_k, \bar{f}(c_k))$ por un punto $(c_k, f_s(c_k))$, donde $f_s(c_k)$ es el promedio sobre el número de valores consecutivos de $\bar{f}$ alrededor de c_k.

Longitud de las Cadenas de Markov

Usando el concepto de cuasi equilibrio se argumenta que la función decreciente del parámetro de control derivado anteriormente, requiere solamente un "pequeño" número de transiciones para aproximarse rápidamente a la distribución estacionaria para el siguiente valor del parámetro de control. Se conjetura que el término "pequeño" puede ser especificado como el número de transiciones para el cual el algoritmo tiene una probabilidad suficientemente grande de visitar al menos la mayor parte de las vecindades de una solución dada. Para cuantificar este número se requiere del siguiente teorema.

Teorema 2 *Sea $\mathcal{S}$ denote un conjunto de cardinalidad S, entonces la fracción esperada de diferentes elementos de $\mathcal{S}$ que se selecciona por N muestreos aleatorios con repetición de $\mathcal{S}$, puede aproximarse, para N y S grandes, por*

$$1 - e^{-\frac{N}{S}}. \tag{16}$$

Prueba La fracción esperada de diferentes elementos de $\mathcal{S}$ que se selecciona por N muestreos aleatorios con repetición de $\mathcal{S}$, tiene la misma probabilidad de seleccionar un elemento dado de $\mathcal{S}$ en N muestras. Una expresión para esta probabilidad puede derivarse de la siguiente manera. La probabilidad de no seleccionar un elemento dado en el conjunto $\mathcal{S}$ en N muestras aleatorias está dado por $(1 - \frac{1}{S})^N$. De manera que, la probabilidad de seleccionar el elemento dado de $\mathcal{S}$ en N muestreos es igual

$$\left[1 - \left(\frac{S-1}{S}\right)^N\right].$$

Por lo tanto, para $N, S \to \infty$ se tiene

$$\lim_{N\to\infty}\left[1 - \left(\frac{S-1}{S}\right)^N\right] = \lim_{S\to\infty}\left[1 - \left(\frac{S-1}{S}\right)^{xS}\right] = 1 - e^{-x}$$

donde $x = N/S$.

Si en el programa de enfriamiento anterior tomamos la longitud de la cadenas de Markov igual al tamaño Θ de las vecindades, es decir:

$$L_k = L = \Theta, \quad k = 0, 1, ..., \tag{17}$$

Suponga ahora, que el algoritmo de recocido simulado comienza con la generación de una cadena de Markov de longitud L para una solución dada i, y suponga que la transición propuesta no se acepta durante la generación de la cadena de Markov. Entonces de acuerdo a (16), donde $S = \Theta$ y $N = L = \Theta$, la fracción de diferentes soluciones de i que son visitadas durante el proceso de generación de cadenas es igual a $1 - e^{-1} \approx \frac{2}{3}$; para tres cadenas de Markov subsecuentes, esta fracción es aproximadamente igual a 1, indicando que aproximadamente todas las vecindades han sido visitadas. Estos números se cumplen para vecindades suficientemente grandes, es decir, $\Theta > 100$; (vea [AAR89]).

En suma, se ha derivado un programa de enfriamiento con tres parámetros, es decir, la tasa de aceptación inicial χ_0, el parámetro de distancia δ y el criterio de paro ϵ_s. Debe notarse que los parámetros no dependen de la instancia que se este ejecutando.

Ahora se probará que el programa de enfriamiento presentado anteriormente permite un número total de pasos del parámeto de control acotado por $O(\ln |S|)$. Esto queda establecido por el siguiente teorema.

Teorema 3 *Sea la función decremento para el parámetro de control dada por*

$$c_{k+1} = \frac{c_k}{1 + \alpha_k c_k}, \quad k = 0, 1, ... \tag{18}$$

donde

$$\alpha_k = \frac{\ln(1+\delta)}{3\sigma_{c_k}}, \quad k = 0, 1, ..., \tag{19}$$

y sea K denote el primer entero para el cual el criterio de paro se satisface, es decir

$$\frac{c_K}{\langle f \rangle_\infty} \left. \frac{\partial \langle f \rangle_c}{\partial c} \right|_{c=c_K} < \epsilon_s, \tag{20}$$

Entonces tenemos

$$K = O(\ln |S|), \tag{21}$$

bajo ciertas condiciones en las derivadas (con respecto a c) del valor esperado $\langle f \rangle_c$ y la entropía S_c.

Prueba La prueba del teorema consiste de dos partes: *(i)* el número total de pasos K se expresa como una función de c_K, el valor final de parámetro de control y *(ii)* se deriva una cota inferior para c_K. Con la combinación de estos dos resultados entonces se obtiene la prueba del teorema.

i. Se puede mostrar directamente por inducción que

$$c_k \leq \frac{c_0}{1 + k\alpha c_0}, k = 0, 1, ..., \tag{22}$$

donde $\alpha = min_k \alpha_k$. Es claro que (22) se cumple para $k = 0$. Suponga que (22) se cumple para k. Entonces usando el hecho $\alpha_k \geq \alpha$ y la hipótesis de inducción se obtiene

$$c_{k+1} = \frac{1}{c_k^{-1} + \alpha_k} \leq \frac{1}{\frac{1+k\alpha c_0}{c_0} + \alpha} = \frac{c_0}{1 + (k+1)\alpha c_0}.$$

Por lo tanto (22) se cumple para $k + 1$. De manera que, se obtiene

$$K \leq \frac{c_0 - c_K}{\alpha c_0 c_K} < \frac{1}{\alpha c_K} = \frac{3max_k \sigma_{c_k}}{\ln(1+\delta)c_K} \approx \frac{3\sigma_\infty}{\ln(1+\delta)c_K},$$

donde σ_∞ esta dado en (2.16).

ii. Si el cuasi-equilibrio se establece para cada valor de c_k se puede suponer que las condiciones de equilibrio discutidas en la Sección 2.3 se cumplen y que se pueden usar las siguientes relaciones; vea (2.12) y (2.13):

$$\frac{\partial}{\partial c}\langle f \rangle_c = c\frac{\partial}{\partial c} S_c. \tag{23}$$

De (2.13) se sabe que S_c se incrementa con c. Además, de (2.18) y (2.19)

$$S_\infty = \ln |\mathcal{S}|, \tag{24}$$

y

$$S_0 = \ln |\mathcal{S}_{opt}| \tag{25}$$

Ahora, usando (23)-(25), K puede expresarse en términos de $\ln|S|$ como sigue. De (20) se concluye que

$$\forall \epsilon' \in (0, \epsilon_S] : \epsilon' = \frac{c_K}{\langle f \rangle_\infty} \frac{\partial \langle f \rangle_c}{\partial c}\bigg|_{c=c_K}.$$

Por consiguiente, si usamos (23), se obtiene

$$\epsilon' = \frac{c_K^2}{\langle f \rangle_\infty} \frac{\partial S_c}{\partial c}\bigg|_{c=c_K}.$$

Puesto que $c_K \ll 1$, se puede aproximar $\frac{\partial}{\partial c} S_c\big|_{c=c_K}$ por $\frac{S_K - S_0}{c_K}$, de donde se obtiene la siguiente relación:

$$\epsilon' \approx \frac{c_K^2}{\langle f \rangle_\infty} \frac{S_K - \ln|S_{opt}|}{c_K} < \frac{c_K \ln|S|}{\langle f \rangle_\infty}$$

Por consiguiente,

$$c_K > \frac{\langle f \rangle_\infty \epsilon'}{\ln|S|}$$

y

$$K < \frac{\ln|S|}{\alpha \langle f \rangle_\infty \epsilon'} \approx \frac{3\sigma_\infty \ln|S|}{\ln(1+\delta)\langle f \rangle_\infty \epsilon'} \tag{26}$$

Así, el número total de pasos en el parámetro de control está acotado por $\eta \ln|S|$, donde $\eta = \frac{3\sigma_\infty}{\ln(1+\delta)\langle f \rangle_\infty \epsilon'}$.

Finalmente se concluye que, si las aproximaciones usadas en la prueba son suficientemente precisas, el algoritmo de recocido simulado, usando la calendarización de los tres parámetros presentados anteriormente, requiere un tiempo $\mathcal{T}$ que satisface la siguiente relación:

$$\mathcal{T} = O(\tau L \ln|S|), \tag{27}$$

donde L denota la longitud de las cadenas de Markov individuales, $\ln|S|$ denota la cota superior del número de cadenas de Markov y τ denota el tiempo requerido para hacer una transición. Para casi todos los problemas de optimización combinatoria, L y τ pueden escojerse polinomialmente en el tamaño del problema. Consecuentemente, si $\ln|S|$ es polinomial en el tamaño del problema, que también se cumple para muchos problemas de optimización combinatoria, el algoritmo de recocido simulado corre en tiempo polinomial.

CONCLUSIONES

En este trabajo se partió del estado del arte de la optimización combinatoria y se enfocó la atención al análisis de la técnica de "Recocido Simulado". Esta técnica es aplicable a todo problema de optimización combinatoria en donde se pueda definir una estructura de vecindades y un programa de enfriamiento. El arte en la definición de estas componentes hacen útil la aplicación de esta técnica a instancias de problemas usualmente poco consideradas por su tamaño.

Se propusieron para una serie de problemas clásicos, una estructura de vecindades y un programa de enfriamiento, observándose que estas condiciones son fáciles de satisfacer para una gran mayoría de problemas combinatorios.

Para cada problema clásico propuesto, se probó la bondad de la técnica de recocido simulado, comparándola con otros tres algoritmos, un exacto, un heurístico y uno de búsqueda local. Para cada uno de los valores fijos de los parámetros asociados a cada problema, se corrieron 100 instancias generadas aleatoriamente. También se presenta experiencia computacional por algoritmo y por problema. Los resultados indican que en promedio el algoritmo difiere a lo más en el 2% de la solución óptima, aunque si el algoritmo se corre varias veces para la misma instancia, en todos los casos se obtuvo la solución óptima.

Se probó convergencia en probabilidad y en distribución del algoritmo al conjunto de soluciones óptimas del problema combinatorio dado.

Un aspecto importante que hay que observar es el hecho que la técnica resulta útil cuando se aplica a problemas combinatorios de mediana o gran escala que pertenecen a la clase NP-completa. Si el problema es de pequeña escala o se conoce un algoritmo eficiente que lo resuelva no se recomienda el uso de recocido simulado, ya que puede obtener una solución lejana a la óptima y el tiempo de ejecución generalmente es mayor, además que el esfuerzo para implantar el algoritmo en computadora es considerable.

REFERENCIAS

[AAR87] Aarts, E. y Korst, J., *Boltzmann Machines and their Applications*, Lecture Notes in Computer Science 258, Springer-Verlag, 1987, pag. 34-50.

[AAR89] Aarts, E. y Korst, J., *Simulated Annealing and Boltzmann Machines*, Wiley, 1989.

[AHN88] Ahn, S.C., Cooper, C., Cornuéjols, G. y Frieze, A., *Probabilistic Analysis of a Relaxation for the k-Median Problem*, Mathematics of Operations Research 13, pag. 1-31, 1988.

[AHO74] Aho, A.V., Hopcroft, J.E. y Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[ANI87a] Anily, S. y Federgruen A., *Ergodicity in Parametric Nonstationary Markov Chains: an Application to Simulated Annealing Methods*, Operations Research 35, 1987, pag. 867-874.

[ANI87b] Anily, S. y Federgruen A., *Simulated Annealing Methods with General Acceptance Probabilities*, Journal of Applied Probability 24, 1987, pag. 657-667.

[BAZ80] Bazaraa, M.S. y Sheralin, H.D. *Benders' Partitioning Scheme Applied to a New Formulation of the Quadratic Assignment Problem*, Naval Research Logistics Quarterly 25, 1980, pag. 29-41.

[BEN62] Benders, J.F., *Partitioning Procedures for Solving Mixed Variables Programming Problems*, Numerische Mathematik 4, 1962, pag. 238-252.

[BIN78] Binder, K. *Monte Carlo Methods in Statistical Physics*, Springer-Verlag, 1978.

[BON86] Bonomi, E. y Lutton, J.L. *The Asymptotic Behavior of Quadratic Sum Assignment Problems: A Statistical Mechanics Approach*, European Journal of Operational Research 26, 1986, pag. 295-300.

[BRO73] Bron, C. y Kerbosch, J., *Finding All Cliques of an Undirected Graph*, Comm.

of ACM, Vol. 16 No 9, Algoritmo 457, 1973, pag. 575.

[BUR78] Burkard, R.E. y Stratmann, K. *Numerical Investigations on Quadratic Assignment Problems*, Naval Research Logistics Quarterly 25, 1978, pag. 129-148.

[BUR80] Burkard, R.E. y Derigs, U. *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs*, Springer-Verlag, 1980.

[BUR83] Burkard, R.E. y Bönniger, T. *A Heuristic for Quadratic Boolean Programs with Applications to Quadratic Assignment Problem*, European Journal of Operational Research 13, 1983, pag. 374-386.

[BUR84] Burkard, R.E. y Rendl, F. *A Thermodynamically Motivated Simulation Procedure for Combinatorial Optimization Problems*, European Journal of Operational Research 17, 1984, pag. 169-174.

[BUR90] Burkard,R.E. *Locations with Spatial Interactions: The Quadratic Assignment Problem*, Discrete Location Theory, eds. Mirchandani y Francis, Wiley, 1990.

[ČER85] Černy, V, *Termodynamical Approach to the Traveling Salesman Problem: an Efficient simulation algorithm*, Journal of Optimization Theory and Applications 45, 1985, pag. 41-51.

[CHA86] Chaudhry, S., McCormick, S.T. y Douglas, M., *Locating Independent Facilities with Maximum Weight: Greedy Heuristics*, Omega Vol. 14 No. 5, 1986, pag. 383-389.

[CHA87] Chams, M., Hertz, A. y de Werra, D. *Some Experiments with Simulated Annealing for Coloring Graphs*, European Journal of Operational Research 32, 1987, pag. 260-266.

[CHI87] Chiu, S.S., *The Minisum Location on an Undirected Network with Continuous Link Demands*, Computers and Operations Research 14, 1987, pag. 369-383.

[CHR75] Christofides, N., *Graph Theory: An Algorithmic Approach*, Academic Press, 1975.

[CHV77] Chvátal, V., *Determining the Stability Number of a Graph*, SIAM J. Comput. Vol. 6 No 4, 1977, pag. 643-662.

[COK71] Cokelez, S. y Burns J.R., *Distribution Systems Warehouse Location and Capacity*, Omega 17, 1989, pag. 45-51.

[COO71] Cook, S.A., *The Complexity of Theorem-proving Procedures*, Proceedings of 3rd Annual ACM Symposium on Theory of Computing, 1971, pag. 151-158.

[ERL78] Erlenkotter, D., *A Dual-Based Procedure for Uncapacitated Facility Location*, Operations Research, Vol. 26, No 6, 1978, pag. 992-1009.

[FEL50] Feller, W., *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, 1950.

[GAR79] Garey, M.R. y Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

[GER79] Gerhards, L. y Lindenberg, W., *Clique Detection for Nondirected Graphs: Two New Algorithms*, Computing 21, 1979, pag. 295-322.

[GLO89] Glover, F., *Tabu Search-Part I*, ORSA Journal on Computing 1, 1989, pag. 190-206.

[GLO90] Glover, F., *Tabu Search-Part II*, ORSA Journal on Computing 2, 1990, pag. 4-32.

[GOL86] Golden, B.L. y Skiscim, C.C., *Using Simulated Annealing to Solve Routing and Location Problems*, Naval Research Logistics Quarterly 33, 1986, pag. 261-279.

[GON84] Gondran, M. y Minoux, M., *Graphs and Algorithms*, Wiley, 1984.

[GRÖ84] Grötschell, M., Lovász, L. y Schrijver, A., *Polynomial Algorithms for Perfect Graphs*, Annals of Discrete Mathematics 21, 1984, pag. 325-356.

[HAJ88] Hajek, B., *Cooling Schedules for Optimal Annealing*, Mathematics of Operations Research 13, 1988, pag 311-329.

[HU 70] Hu, T.C., *Integer Programming and Network Flows*, Addison-Wesley, 1970.

[HU 82] Hu, T.C., *Combinatorial Algorithms*, Addison-Wesley, 1982.

[JAC83] Jacobsen, S.K., *Heuristics for the Capacitated Plant Location Model*, European Journal of Operational Research 12, 1983, pag. 253-261.

[JAG86] Jagannathan, R., Krishnakumar, P., Mirchandani, B. y Wong, R.T., *A Nested-Dual Approach to the Generalized Hierarchical Location Model*, Operations Research 37, 1986, pag. 86-112.

[JOH89] Johnson, D.S., Aragon, C.R., McGeoch, L.A. y Schevon, C., *Optimization by Simulated Annealing: an Experimental Evaluation; Part I, Graph Partitioning*, Operations Research 37, 1989, pag. 865-892.

[KAO89] Kao, C, *A Random-number Generator for Microcomputers*, Journal Operational Research Society, Vol 40 No 7, 1989, pag. 687-691.

[KAR75] Karp, R.M., *On the Complexity of Combinatorial Problems*, Networks, 5, 1975, pag. 45-68.

[KAR75b] Karlin, S. y Taylor, H., *A First Course in Stochastic Processes*, 2x Edición, Academic Press, 1975.

[KIK86] Kikusts P., *Another Algorithm Determining the Independence Number of a Graph*, Electron, Inf. verarb. Kybern, EIK 22, 4, (Journal of Information Processing and Cybernecties) 1986, pag. 157-166.

[KIR83] Kirkpatrick, S., Gelatt C.D. y Vecchi, M.P., *Optimization by Simulated Annealing*, Science 220, 1983, pag. 671-680.

[KHU72] Khumawala, B.M., *An Efficient Branch and Bound Algorithm for the Warehouse Location Problem*, Management Science 18, 1972, pag. 718-731.

[KOO57] Koopmans.T.C. y Beckmann, M.J., *Assignment Problems and the Location of Economic Activities*, Econometrica 25, 1957, pag. 53-76.

[LAA88] Laarhoven P.J.M. van y Aarts E.H.L. *Simulated Annealing: Theory and Applications*, Reidel, 1988.

[LAD77] Ladner, R.E., *The Computational Complexity of Probability in Systems of Modal Propositional Logic*, SIAM J. Computing, Vol 6, 1977, pag. 467-480.

[LAW76] Lawler, E.L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rine-

hart and Winston, 1976.

[LAW85] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. y Shmoys D.B. Editores, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

[LOV88] Love, R.F., Morris, J.G., Wesolowsky, G.O., *Facilities Location Models & Methods*, North-Holland, 1988.

[LUN86] Lundy, M. y Mees, A. *Convergence of an Annealing Algorithm*, Mathematical Programming 34, 1986, pag. 111-124.

[MAG81] Magnanti, T.L., y Wong, R.T. *Accelerating Benders' Descomposition: Algorithmic Enhancement and Model Selection Criteria*, Operations Research 34, 1981, pag. 464-484.

[MAG86] Magnanti, T.L., Mireault, P. y Wong, R.T. *Tailoring Benders' Descomposition for Network Design*, Mathematical Programming Study 26, 1986, pag. 112-154.

[MAR79] Martello, S., *An Algorithm for Finding a Minimal Equivalent Graph of a Strongly Connected Digraph*, Computing No 21, (1979) pag. 183-194.

[MAR90] Martello, S., y Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, 1990.

[MET53] Metropolis, N., Rosenbluth, M., Rosenbluth, A., Teller, A. y Teller, E. *Equation of state calculations by fast computing machines*, Journal of Chemical Physics 21, pag. 1087-1092.

[MIR86] Mirchandani, P.B. y Reilly, J.M., *Spatial Nodes in Discrete Location Problems*, Annals of Operations Research 6, 1986, pag. 203-222.

[MIR90] Mirchandani, P.B. y Francis, R.L. editores, *Discrete Location Theory*, Wiley, 1990.

[MIT90] Mitchell, P.B. y Crawford, J.R., *An Interactive Model for the Layout of Buildings*, Applied Mathematical Modelling 9, 1985 pag. 207-214.

[NEM89] Nemhauser, G.L., y Wolsey, L.A. *Integer and Combinatorial Optimization*,

Wiley, 1989.

[NUG68] Nugent, C.E., Vollmann, T.E. y Ruml, J., *An Experimental Comparison of Techniques for Assignment of Facilities to Locations*, Operations Research 16, 1968, pag. 150-173.

[OSM89] Osman, I. y Potts, CN. *Simulated Annealing for Permutation Flow-Shop Scheduling*, Omega 17, 1989, pag. 551-557.

[PAI87] Paige, R. y Tarjan, E., *Three Partition Refinement Algorithms*, SIAM Journal on Computing 16, 1982, pag. 266-281.

[PAP82] Papadimitriou, C.H. y Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982.

[SAS88] Sasaki, G.H. y Hajek B., *The Time Complexity of Maximum Matching by Simulated Annealing*, Journal of the ACM 35, 1988, pag. 387-403.

[SCH86] Schrijver, A., *Theory of Linear and Integer Programming*, Wiley, 1986.

[SEN81] Seneta, E., *Non-negative Matrices and Markov Chains*, Segunda edición, Springer-Verlag, 1981.

[SHA75] Shannon, R.E. y Ignizio, J.P. *A heuristic programming algorithms for warehouse location*, AIIE Transactions vol. II, No 4, pag. 334.

[SIM89] Simkin, L.P., *SLAM: Store Location Assessment Model-Theory and Practice*, Omega, Vol. 17 No 1, 1989, pag. 53-58.

[SLO82] Slominski, W.L., *Probabilistic Analysis of Combinatorial Algorithms: A Bibliography Selected Annotations*, Computing No 28, 1982, pag. 257-267.

[SYS83] Syslo, M., Deo, N. y Kowalik, J., *Discrete Optimization Algorithms with PASCAL Programs*, Prentice-Hall, 1983.

[TAY84] Taylor, H. y Karlin, S., *An Introduction to Stochastic Modeling*, Academic Press, 1984.

[TOD83] Toda, M., Kubo, R. y Saitô, N., *Statistical Physics*, Springer-Verlag, 1983.