UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

BUSQUEDA TABU EN PARALELO PARA EL PROBLEMA DEL PLEGAMIENTO DE PROTEINAS

T E S I S

Que para obtener el título de:

M A T E M A T I C O

p r e s e n t a

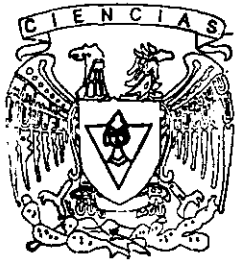
RIVEROS CASTRO, FELIPE DE JESUS

Directores de Tesis:

Dr. Luis B. Morales

Dr. Ramón Garduño Juárez

1999



FACULTAD DE CIENCIAS
UNAM

VNIVERMDAD NACIONAL
AVTNOMA DE
MEXICO

MAT. MARGARITA ELVIRA CHÁVEZ CANO
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"Búsqueda Tabú en Paralelo para el Problema del Plegamiento de Proteínas"

realizado por Riveros Castro Felipe de Jesús

con número de cuenta 8834949-1 , pasante de la carrera de Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis
Propietario Dr. Luis B. Morales Mendoza
Director de Tesis
Propietario Dr. Ramón Garduño Juárez

Propietario M. en I. María Luz Gasca Soto

Suplente Dr. Jesús López Estrada

Suplente M. en C. Jesús Galaviz Casas

Consejo Departamental de Matemáticas

Mtro. César Guevara Bravo

BÚSQUEDA TABÚ EN PARALELO PARA EL PROBLEMA DEL PLEGAMIENTO DE PROTEÍNAS

Felipe de Jesús Riveros Castro

Marzo de 1999

AGRADECIMIENTOS:

Este trabajo me da la oportunidad de agradecer a las personas que de algún modo han sido importantes en mi desarrollo profesional; en especial:

- A mis padres que me brindaron la oportunidad que ellos no tuvieron.
- A mis hermanos que siempre los llevo conmigo.
- A Martha mi esposa que siempre me apoya.
- A mis directores de tesis, Luis B. Morales y Ramón Garduño Juárez quienes me brindaron la oportunidad de trabajar con ellos.
- A los sinodales que de forma profesional cedieron parte de su tiempo y aceptaron revisar esta tesis.

Riveros Castro Felipe de Jesús
México D.F. marzo de 1999

Quisiera agradecer el apoyo económico parcial brindado por :

- PAPIIT – DGAPA – UNAM (IN – 102397)
- CRAY – UNAM fondo de investigación (SC – 002195) y;
- CONACyT – México (25245 – E)

También quiero agradecer las facilidades otorgadas por la Dirección General de Servicios de Computo Académico DGSCA UNAM.

Riveros Castro Felipe de Jesús
México D.F. marzo de 1999

Contenido

Resumen

Este trabajo propone un acercamiento a la solución del problema denominado plegamiento de proteínas, cómo predecir la estructura tridimensional de una proteína dada su secuencia de aminoácidos, además se hace una breve introducción a la programación en paralelo utilizada para el desarrollo de la función de evaluación; mediante el uso de una heurística llamada Búsqueda Tabú.

Desde un punto de vista matemático, existen varios importantes para el manejo del problema:

- La selección de una función de energía potencial apropiada;
- El parámetro de identificación mediante la correcta captura de información experimental; y
- La optimización global de la energía potencial.

Introducción:

El "plegamiento de proteínas" es uno de los problemas que más reto representa en la bioquímica actual y es una fuente muy rica de problemas interesantes en el desarrollo de modelos matemáticos.

Después de haberse descifrado satisfactoriamente el código genético que define cómo las secuencias de aminoácidos en las proteínas se encuentran codificadas en el ADN, uno de los más importantes pasos que faltan para entender la base química de la vida, es el problema del plegamiento de proteínas. La labor de entender y predecir cómo la información codificada en la secuencia de aminoácidos de proteínas en el tiempo de su formación, se traduce en una estructura tridimensional de la proteína biológicamente activa.

Según Arnold Neumaier [1] "Las proteínas son las máquinas y los ladrillos de construcción de las células vivientes. Si comparamos un cuerpo vivo con nuestro mundo, cada célula corresponde a un pueblo, y las proteínas son las casas, los puentes, los carros, grúas, caminos, aviones, etcétera. Hay gran número de diferentes proteínas, cada una llevando a cabo una labor específica."

Ya que actualmente se conoce cómo utilizar la ingeniería genética para producir proteínas con una secuencia de aminoácidos dada, el conocimiento de cómo dicha proteína

se puede plegar, permitiría predecir sus propiedades químicas y biológicas. Si fuéramos capaces de resolver el problema del plegamiento de proteínas, esto simplificaría enormemente la labor de interpretar la información recabada por el proyecto del genoma humano, entendiendo el mecanismo de enfermedades infecciosas y hereditarias, diseñando drogas con propiedades específicas terapéuticas, y cosechando polímeros biológicos con propiedades de materiales específicos.

Para buscar una solución al problema del plegamiento de proteínas y lograr simular estructuras tridimensionales, se han empleado variados métodos. Sin embargo, debido a que este problema presenta una explosión combinatoria, se requiere de técnicas robustas para su solución.

Una característica de la mayoría de los problemas de optimización combinatoria es que presentan varios mínimos globales y muchos mínimos locales, y es aquí donde muchas técnicas resultan poco eficientes. La Búsqueda Tabú es una heurística y se presenta como una atractiva alternativa, ya que ha probado ser eficaz en varios problemas de optimización combinatoria como el problema del club de bridge, coloración de gráficas, problemas de horarios, el problema del agente viajero, etcétera.

Además hemos empleado la programación en paralelo para disminuir los tiempos computacionales en la búsqueda de posibles soluciones.

Los principales objetivos de este trabajo son:

1. Aplicar la técnica de la Búsqueda Tabú para obtener conformaciones de estructuras moleculares de proteínas a partir de una cadena de aminoácidos, y reportar la experiencia computacional obtenida.

2. Estudiar la programación en paralelo y utilizarla en la implementación de la función de evaluación.

3. Desarrollar una función de evaluación en paralelo, que sea altamente eficiente para la aplicación de la técnica Búsqueda Tabú al problema de plegamiento de proteínas.

Contenido y aportación de la tesis

Se proporciona una aplicación concreta en la que se muestra la importancia de la Búsqueda Tabú en los problemas de optimización combinatoria. Como aportaciones se pueden citar las siguientes:

1. Se desarrolla un programa en paralelo para el problema de plegamiento de proteínas.

2. Se reporta la experiencia computacional y se realiza una introducción de la programación en paralelo en la máquina SGI\CRAY ORIGIN 2000 con 32 procesadores, que se localiza en la Dirección General de Servicios de Cómputo Académico (DGSCA UNAM).

En el primer capítulo se describe brevemente el problema de plegado de proteínas y el por qué utilizar Búsqueda Tabú para su solución. En el capítulo 2 se describe con más detenimiento el problema, planteándose como un problema de optimización combinatoria. En el capítulo 3 se presenta una introducción a la optimización combinatoria. En el capítulo 4 se introduce a la técnica de la Búsqueda Tabú y se describen mediante el modelado del problema sus principales elementos. En el capítulo 5 se introduce a la programación en paralelo utilizando la *interface de paso de mensajes* (MPI) en la máquina SGI\CRAY ORIGIN 2000. En el capítulo 6 se describe como se desarrollo el programa para Búsqueda Tabú. En el capítulo 7 se muestra la experimentación realizada al aplicar el programa. En el capítulo 8 se presentan las conclusiones.

Durante el desarrollo de este trabajo se utilizará una molécula llamada *met-encefálina* cuya estructura molecular se ha adoptado como patrón para comprobar la eficiencia de nuestro programa.

Capítulo 1

Descripción del problema.

En este capítulo se plantea el problema como un problema matemático y se introduce la terminología que se empleará posteriormente.

1.1 Marco teórico del problema

La búsqueda conformacional de moléculas de proteínas, en una primera aproximación, puede ser vista como el problema de encontrar una estructura molecular tridimensional que corresponda al mínimo local más bajo de una función matemática apropiada, que describa la energía potencial del sistema. Simulaciones por computadora se utilizan comúnmente para realizar tal tarea. Una preocupación importante en las simulaciones por computadora es la obtención de un conjunto de conformaciones de baja energía con significado biológico, esto es, considerando que el estado nativo termodinámico de estas moléculas puede corresponder al mínimo más bajo de energía o *mínimo global*. Una posible manera de accesar a este mínimo es a través de una búsqueda cuidadosa del paisaje de energía conformacional que eventualmente nos acercará a un *mínimo global*. En principio, podría ser suficiente encontrar un conjunto crudo de estructuras alrededor del *mínimo global* que a través de una optimización local puede llegar al buscado mínimo de energía. Dado que la mayoría de las funciones de energía potencial que describen el *espacio conformacional de péptidos* tienen un gran número de mínimos locales y posiblemente un único *mínimo global*, encontrarlo en una forma certera y rápida es de interés general.

Una molécula de proteína es un *polímero* compuesto de una secuencia específica de 20 *aminoácidos* naturales, los ladrillos de construcción o monomeros, conectados a través de *enlaces péptidicos*. En la naturaleza la mayoría de las proteínas se pliegan espontáneamente, de ambas formas, *in vivo* e *in vitro*, a sus conformaciones nativas. Bajo condiciones biológicas, algunos movimientos internos de estas moléculas ocurren en lapsos más cortos que otros. Experimentalmente, el promedio de la distancia de *enlace covalente* y *el ángulo covalente* son relativamente constantes, llevando a la suposición de que los cambios conformacionales observados a través de los *ángulos dihedros* podrían determinar completamente la forma general de la molécula de proteína. Así, si uno especifica la posición de todos los átomos en la molécula de proteína como una función de sus coordenadas internas, *distancia de enlace, ángulos de enlace* y *ángulos dihedros*, bajo la suposición de *longitudes de enlace constantes* y *ángulos de enlace constantes*, el problema reduce drásticamente su tamaño en el espacio conformacional, que por sí mismo es extremadamente complicado por muchos factores externos que también dan origen a una gran cantidad de mínimos.

Varios campos de fuerza diferentes para proteínas han sido diseñados como la suma de un conjunto de contribuciones a la energía potencial. Entre los más usados están: ECEPP, MM2, ECEPP/2, CHARMM, DISCOVER, AMBER, GROMOS87, MM3 y ECEPP/3. En este trabajo se utiliza el campo de fuerza ECEPP/3 , que supone que todas las longitudes de los *enlaces covalentes* y los *ángulos de enlace* se encuentran fijos en sus *valores de equilibrio*; que la energía conformacional correspondiente surge de una suma de energías *electrostática, de no-enlace, de puentes de hidrógeno* y contribuciones de *energía torsional*, además de varios potenciales empíricos para el cerrado del ciclo de un puente disulfuro y una energía conformacional fija para el *anillo pirolidino* en ambos residuos *prolil* e *hidroxiprolil*.

Aunque el tamaño del problema puede reducirse cuando la función de energía se escribe en términos de los *ángulos torsionales*, es sabido que cuando una función se somete a técnicas de optimización simple, generalmente se llegará a *mínimos locales*. Para sobreponer este efecto muchos autores han diseñado interesantes métodos estocásticos y determinísticos que imponen restricciones y predisponen la búsqueda hacia la región donde puede encontrarse el *mínimo global*. Entre los métodos estocásticos empleados se encuentran el Recocido Simulado (RS), Método de Umbrales (MU), Monte Carlo con Minimización (MCM), Monte Carlo con energía Libre con Minimización (MCLM), Ensamble Multicanonico (EM), y Algoritmos Genéticos (AG). Entre los métodos determinísticos encontramos Dinámicas Moleculares con Minimización (DMM), el Método de la Ecuación de Difusión (MED), la Técnica del Campo Medio (TCM) y la Programación Dinámica (PD). La mayoría de estos métodos presentan baja eficiencia, y están limitados por la alta dimensionalidad del problema. Muy recientemente un método llamado αBB[13] ha reclamado tener la garantía teórica de obtener la solución ε - global mínima de funciones analíticas diferenciables.

Este trabajo forma parte de un proyecto* que experimenta con una serie de métodos heurísticos que pretende reducir el campo de la búsqueda conformacional y el tiempo necesario para encontrar una estructura 3D cerca o en el *mínimo global de péptidos*. Reportes previos han lidiado con el uso de Recocido Simulado (RS) y Método de Umbrales (MU), para encontrar un *mínimo global* de un *péptido* bien estudiado, la *met-encefálina*, que a través de los años ha servido como modelo de validación. Con RS y MU se han encontrado excelentes estructuras de baja energía, pero estos métodos carecen de eficiencia. En este trabajo, se sugiere un acercamiento usando la Búsqueda Tabú (BT), un método estocástico de optimización global desarrollado para tratar grandes tareas de optimización combinatoria. Aparentemente, BT es una técnica de búsqueda mejor que RS ya que mejora el tiempo requerido para obtener una solución y la calidad de la misma. Se ha diseñado un procedimiento de optimización basado en BT para encontrar la estructura más baja de energía mínima de la *met-encefálina* usando la función empírica de energía ECEPP/3 como función objetivo. Una breve descripción del campo de fuerza ECEPP/3 puede ser:

$$U = U_{elec} + U_{nonb} + U_{nb} + U_{tor} + U_{loop} + U_{S\text{-}S}$$

Donde,

$$U_{elec} = \sum_{i} \sum_{i \neq j} 332.0 q_i q_j / D r_{ij}$$

$$U_{nonb} = \sum_{i} \sum_{i \neq j} F A / r_i^{12}{}_j - C / r_i^{6}{}_j$$

$$U_{hb} = \sum_{i} \sum_{i \neq j} A'_{hx} / r_h^{12}{}_x - B_{hx} / r_h^{10}{}_x$$

$$U_{tor} = \sum_{k} (U_0 / 2.0)(1 \pm \cos n_k \theta_k)$$

$$U_{loop} = \sum_{l} B_l \sum_{i=1}^{i=3} (r_{il} - r_{i0})^2$$

$$U_{S\text{-}S} = \sum_{s} A_s (r_{4s} - r_{40})^2$$

Todas las constantes son ajustadas a datos experimentales apropiados.

---

* Proyecto PAPIIT IN-102397 "Métodos heurísticos aplicados y la solución del problema del Plegado de Proteínas."

Dadas estas definiciones, el problema de minimizar la energía potencial puede verse resumido de la siguiente manera:

min. $U(\phi_i, \psi_i, \omega_i, \chi_i^j, \varphi_k^N, \varphi_k^C)$, sujeta a las siguientes restricciones experimentales

$$-180° \leq \phi_i, \psi_i \leq +180° \qquad i = 1,\ldots,N_{res} \qquad (A)$$

$$* \quad -10° \leq (\omega_i - 180°) \leq +10° \qquad i = 1,\ldots,N_{res} \qquad (B)$$

$$-180° \leq \chi_i^j \leq +180° \qquad i = 1,\ldots,N_{res},\ j = 1,\ldots,J^i \qquad (C)$$

$$-180° \leq \varphi_k, \varphi_k \leq +180° \qquad k = 1,\ldots,K^N,\ k = 1,\ldots,K^C \qquad (D)$$

En este proceso fueron incluidos los ángulos anticorrelacionados* en los ángulos ϕ_i, ψ_{i+1}

1.2 Composición de las proteínas

En los organismos vivos[1], las proteínas están implicadas en una mayor cantidad y variedad de eventos bioquímicos que cualquier otro tipo de moléculas. De ahí la importancia de su estudio.

Los *aminoácidos*, son las unidades básicas de las proteínas. Los *aminoácidos* se unen por medio de los *enlaces péptidicos* para formar *cadenas péptidicas*. A cada una de estas unidades se le conoce como *residuo*. Un *residuo* consiste en un grupo *amino*, un grupo *carboxilo*, un átomo de hidrógeno y un grupo distintivo *"R"*[2] enlazados al átomo de carbono que se llama *carbono-α*. Una proteína o *cadena péptidica* está formada por una *cadena principal* y una *cadena lateral*. El grupo distintivo *R* de cada *aminoácido* forma la *cadena lateral*, Fig. 1.2.

Existen veinte tipos de *cadenas laterales* de *aminoácidos*, estas varían en tamaño, forma, carga, capacidad de puentes de hidrógeno y reactividad química. Estos tipos son el alfabeto fundamental de las proteínas. A los ángulos internos de la cadena se les conoce como *ángulos χ*, Fig. 1.1.

---

* El término de ángulos anticorrelacionados se refiere a ángulos opuestos, se explica en el capítulo 6 inciso 1.

[1] Poder catalítico: aumentan la velocidad de reacción al menos 1000000 veces, transformaciones químicas en los seres vivos.

[2] Un grupo "R" se refiere a una cadena lateral o conformación de los átomos que no forman parte de la cadena principal.

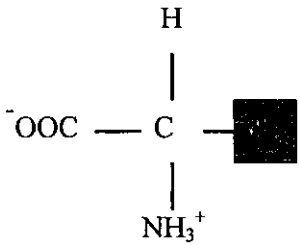
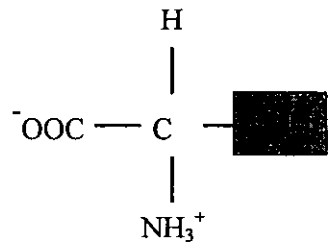
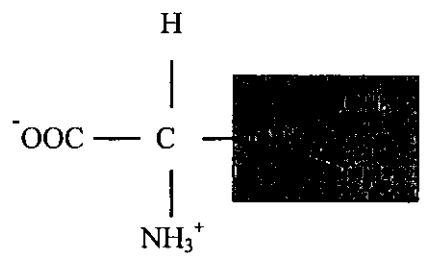
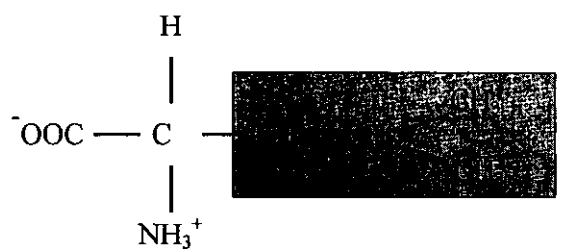
Estructura	Nombre	Abreviatura
	Glicina	Gly (G)
	Alanina	Ala (A)
	Valina	Val (V)
	Leucina	Leu (L)

Fig. 1.1 Algunos ejemplos de estructuras de aminoácidos

Se llamara *conformación* a la organización de los átomos que componen de manera tridimensional la estructura de la proteína. Con esto debemos hacer notar que la secuencia de *monómeros*[3] que forman a una proteína no cambia, sino solo su posición determinada en el espacio por la rotación de los ángulos de sus enlaces.

2.3 Conformación de proteínas

El término *Conformación*, en general, se emplea para determinar los arreglos espaciales de una molécula por rotaciones sencillas en uniones rotables. En el caso de las proteínas, la conformación describe la organización espacial del conjunto de *péptidos* que la forman. No debemos confundir el término *conformación* con el término *configuración*. Configuración se refiere a un arreglo específico, característico y estable de átomos o grupos de átomos. La conversión de una configuración a otra es solo posible rompiendo directamente el enlace; sin embargo, cada configuración puede existir en un número infinito de conformaciones. Por ejemplo, si vemos a una proteína como una secuencia de *péptidos* específica, su configuración se referirá al orden que tiene esa secuencia, mientras que su conformación indicará la posición en el espacio de cada uno de los *péptidos* que forman esa secuencia.

Se dice que una *cadena péptidica* tiene una dirección porque sus elementos constructores tienen extremos diferentes. Por convención se toma al extremo amino como el comienzo de la cadena, Fig.1.2.

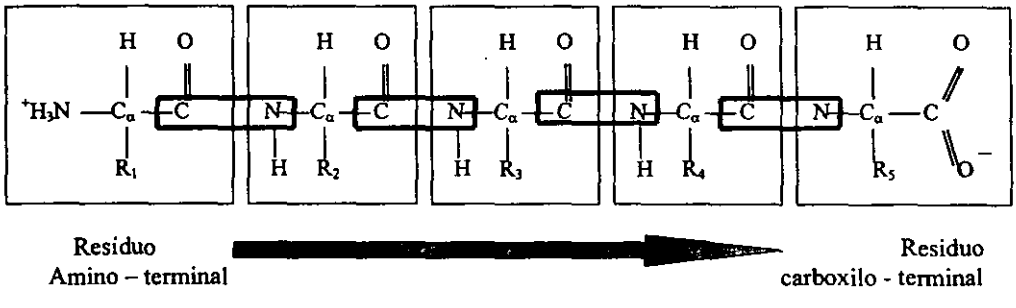


Fig. 1.2 Polipéptido formado por 5 aminoácidos. La cadena comienza en el extremo amino.

[3] El término monómero se utilizará para referirse también a un sólo residuo o aminoácido

En las proteínas el *grupo alfa carboxilo* se une al *grupo alfa amino* por medio de un *enlace peptídico*, la unión de dos *aminoácidos* se lleva a cabo a través de la pérdida de una molécula de agua.

Una característica de las proteínas es que poseen estructuras tridimensionales bien definidas. Una cadena estirada u organizada al azar, no tiene actividad biológica.

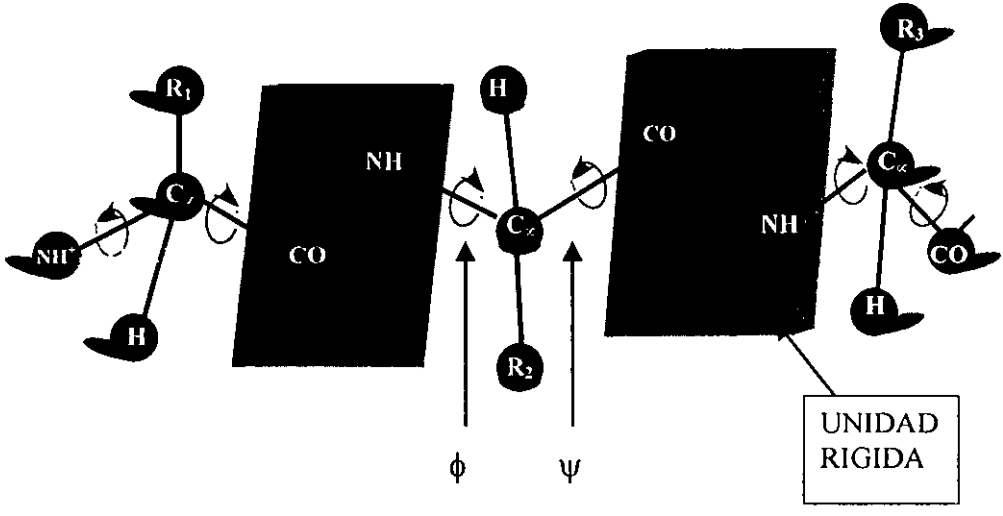


Fig. 1. 3: Unidad planar rígida

El *grupo péptidico* es una *unidad planar rígida*, que quiere decir que el hidrógeno del grupo - NH - está casi siempre en posición "*trans*" respecto al oxigeno del *grupo carboxilo* (CO); existe muy poca libertad de rotación alrededor del enlace entre el *carbono carboxilico* y el *nitrógeno* del *grupo peptídico*, Fig. 1.3.

En cambio, existe un enorme grado de libertad rotacional alrededor de los enlaces de *carbono* α y el *carbono carboxilico* (CO), y entre el *carbono* α y el *átomo de nitrógeno amida* (enlace sencillo puro). A estas rotaciones se les llama *ángulos* ψ y ϕ, Fig. 1.4.

La conformación de la cadena principal o columna vertebral del *polipéptido* está definida completamente cuando los ángulos ψ y ϕ se conocen para cada uno de los *residuos* de los *aminoácidos*.

La secuencia de *aminoácidos* es muy importante, ya que determina la conformación de la proteína específica. Al mismo tiempo, la función de una proteína nace de la conformación que esta tenga.

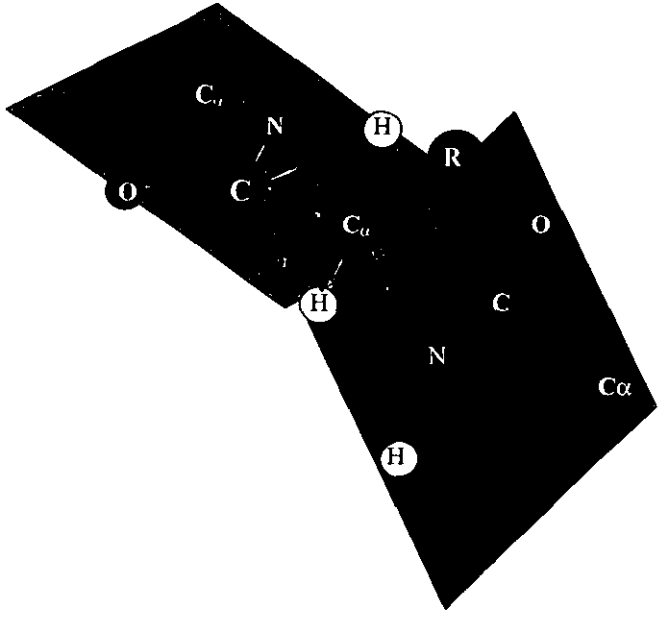


Fig. 1.4: Rotaciones de enlaces simples Cα - N

Capítulo 2

Optimización combinatoria

En éste capítulo se hará una breve introducción a la optimización combinatoria y los problemas que presenta..

2.1 Problemas de optimización combinatoria

Muchos investigadores han estudiado el problema de buscar soluciones óptimas a problemas que pueden ser estructurados como una función de algunas *variables de control*, y tal vez con la presencia de algunas *restricciones*. Tales problemas pueden ser formulados de la siguiente manera:

Minimizar $f(x)$ sujeta a:

$g_i(x) \geq b_i; \quad i = 1, \ldots, m;$

$h_j(x) \geq c_j; \quad j = 1, \ldots, n.$

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

Existen muchas clases de tales problemas, obtenidos aplicando restricciones sobre el tipo de funciones bajo consideración y en los valores que las variables de control puedan tomar. Quizás los problemas más conocidos son aquellos obtenidos restringiendo a que $f(x)$, $g_i(x)$ y $h_j(x)$ sean funciones lineales de las variables de control, que están permitidas a tomar *variables continuas*, lo que nos conduce a problemas de *programación lineal*.

Nosotros consideraremos aquí otra clase de problemas; aquellos de *naturaleza combinatoria*. Este término es usualmente reservado para problemas en los cuales las variables de control son *discretas*. El problema de buscar una solución optima a tales problemas es conocido como *optimización combinatoria*.

Los problemas de *optimización combinatoria* tienen nexos cercanos a la programación lineal y muchos de los intentos para resolverlos utilizan estos métodos, generalmente introduciendo *variables enteras* que toman valores de 0 o 1, para producir una formulación de *programación entera*. Tales formulaciones en ocasiones involucran un gran número de variables y restricciones, y no pueden hacer frente a problemas muy grandes.

Un ejemplo de un problema de optimización combinatoria es el siguiente:

Problema de Asignación. Un conjunto de n personas está disponible para realizar n tareas. Si la i-ésima persona realiza la j-ésima tarea, esta representa un costo de Cij unidades. Entonces el problema consiste en encontrar una asignación $\pi_1, \ldots, \pi_n$ la cual minimice:

$$\sum_{i=1}^{n} C_{i\pi j},$$

aquí la solución está representada por la permutación
$\pi_1, \ldots, \pi_n$ de los números $1, \ldots, n$.

Debe quedar claro que, en el ejemplo anterior, utilizamos el término problema en el sentido genérico. Dada una situación en la vida real, tendremos un problema particular, en el cual los símbolos utilizados para describirlo deben tomar valores numéricos específicos. Es común usar el término ejemplar para poder distinguir una situación particular de una general.

2.2 Optimos locales y globales

Una cualidad de la mayoría de los problemas de optimización combinatoria es la de presentar muchos mínimos globales y muchisimos mínimos locales, esto es un problema ya que podemos quedar atrapados en alguno de los muchisimos mínimos locales y lo que se desea es encontrar un mínimo global. Podemos hacer más concreta esta idea si introducimos el concepto de *vecindad.*

Estrictamente hablando, una vecindad $V(x,\sigma)$ de una solución x es un conjunto de soluciones que pueden ser alcanzadas a partir de x aplicando una simple operación σ. Tal operación podría ser quitar o agregar un objeto a la solución o hacer el intercambio de dos objetos en la solución. Particularmente, en la técnica de la *Búsqueda Tabú* a estas operaciones se les conoce con el nombre de *movimientos.* Si una solución y es mejor que cualquier otra solución de su *vecindad* $V(y,\sigma)$ entonces y es un mínimo local con respecto a su *vecindad.*

En algunos casos es posible encontrar un *movimiento* σ, tal que un óptimo local sea también un óptimo global. Hay muchos tratamientos existentes para la optimización combinatoria, pero éstos tienden a concentrarse en métodos que son exactos en vez de heurísticos. Esto es, ellos están principalmente relacionados con aquella técnicas que garantizan encontrar una solución óptima al problema establecido. Estos métodos usualmente están en la teoría de la *programación lineal*, o usan métodos de *enumeración implícita* tales como el *branch and bound* [13].

2.3 Heurísticas

La técnica de la *Búsqueda Tabú* se conoce como una *heurística*[39]. Este término deriva del griego *heuriskein* que significa encontrar o descubrir. Sin embargo, las heurísticas no garantizan encontrar una solución óptima. El término heurístico es usado en contraste a los métodos que sí garantizan encontrar una solución óptima global y ha llegado a ser común en el contexto de la optimización combinatoria.

Definición. Una heurística es aquella que ayuda a guiar al proceso de búsqueda mejorando en cada intento de aproximación su eficiencia. Sin embargo, no garantiza el hallazgo de una solución óptima, pero sí nos proporciona buenas soluciones, cercanas a la óptima, con un costo computacional razonable.

Una gran cantidad de artículos tratan de como las técnicas heurísticas se han utilizado para resolver problemas de optimización combinatoria. Pareciendo doble la causa de este gran interés. Por un lado, el desarrollo del concepto de complejidad computacional ha proporcionado las bases para explorar las heurísticas. Por otro lado, han surgido nuevas técnicas, muy eficientes, para resolver problemas de optimización combinatoria en tiempos razonables de cómputo.

El método clásico para resolver ejemplares de un problema de optimización combinatoria es simplemente listar todas las soluciones factibles de un ejemplo dado, evaluar su función objetivo y escoger la mejor solución. Podríamos pensar que la solución de un problema de optimización combinatoria únicamente se limita a buscar de manera exhaustiva el valor que minimice o maximice la función objetivo dentro de un conjunto finito de posibilidades, tal vez utilizando una computadora muy rápida, el problema carecería de interés matemático, sin detenernos a pensar por un momento en el tamaño de este conjunto de soluciones.

Sin embargo, es obvio que este método de enumeración es muy ineficiente conforme crece el tamaño de la entrada del problema debido a la explosión combinatoria del espacio de soluciones, es decir, dado un conjunto de elementos se pueden obtener diferentes arreglos ordenados de estos, permitiendo una gran cantidad de posibilidades para cualquier entrada de tamaño grande, por ejemplo de orden de n!.

Para ilustrar este punto, considérese el *Problema del Agente Viajero (Travelling Salesman Problem),* el cual ha fascinado a los investigadores en optimización combinatoria, probablemente porque es muy fácil de describir, pero muy difícil de resolver. Sin embargo, se han utilizado ya técnicas heurísticas junto con *técnicas de paralelización* para resolver ejemplares de gran tamaño para este problema con éxito[20].

Problema del agente viajero. Un vendedor tiene que buscar una ruta que visite cada una de N ciudades, una y sólo una vez, que minimice el total de la distancia recorrida, iniciando en cualquiera de ellas y volviendo a la misma ciudad. Cómo el punto inicial es arbitrario, hay (N - 1)! posibles soluciones o (N - 1)! / 2 si la distancia entre cada par de ciudades es la misma sin considerar la dirección del viaje.

Supóngase que tenemos una computadora que puede listar todas las posibles soluciones de un ejemplar; entonces utilizando la fórmula anterior, tendríamos como resultado la Tabla 3.1[39] que nos muestra el crecimiento del tiempo de cómputo con respecto al tamaño de la entrada del problema, en el cual podemos observar que la *enumeración completa* es ineficiente para obtener una solución óptima, ya que por ejemplo un problema de 25 ciudades tomaría aproximadamente 6 siglos en ser resuelto.

20	1	hora
21	20	horas
22	17.5	días
23	1.05	años
24	24.26	años
25	5.82	siglos

Tabla 3.1: Crecimiento del tiempo de cómputo [39]

Varios algoritmos exactos han sido inventados para encontrar soluciones óptimas de problemas más eficientemente que la *enumeración completa*. El algoritmo más conocido es el *Simplex* para problemas de *programación lineal*. Estos algoritmos fueron capaces de resolver pequeños ejemplares, pero no lo fueron para encontrar soluciones óptimas, en una cantidad razonable de tiempo computacional, cuando los ejemplares son grandes. Como el poder computacional se ha incrementado en los últimos años, ha sido posible resolver grandes problemas, y los investigadores se han interesado en cómo el tiempo de solución varía con el tamaño de la entrada del problema.

Sin embargo, para problemas tales como el del agente viajero, el esfuerzo computacional sigue siendo exponencial. A finales de los años 60's los investigadores se hacían la siguiente pregunta: ¿Habrá un algoritmo de optimización en tiempo polinomial para un problema como el del agente viajero?. Nadie ha sido capaz de responder a esta pregunta, sin embargo en 1972, Karp[42] mostró que sí la respuesta es afirmativa para el problema del agente viajero, luego entonces hay también un algoritmo en tiempo polinomial para otros problemas equivalentes. Como ningún algoritmo ha sido encontrado para estos problemas, esto indica que la respuesta a la pregunta original es probablemente no. Sin embargo, la respuesta real es desconocida.

Hay problemas que tienen *algoritmos polinomiales* conocidos y se dice que están en la *clase P*. Pero ¿qué hay del problema del agente viajero y otros problemas equivalentes? ¿son de complejidad exponencial? Muchos de estos problemas están en la *clase NP*, que es una abreviación de *non-deterministic polinomial*[32]. De hecho el problema del agente viajero es de los problemas más difíciles en *NP*, sí un algoritmo polinomial fuera encontrado para este problema, esto significaría que existe un algoritmo polinomial para todos los problemas en *NP*; pero como ningún algoritmo polinomial exacto ha sido encontrado para cualquier problema en *NP*, hay fuerte evidencia de que $P \neq NP$, lo cual es un argumento para buscar formas alternativas de resolver problemas computacionalmente difíciles. Existe la posibilidad de que alguien llegue a probar que $P = NP$; pero mientras

nadie encuentre tal prueba, el uso de las técnicas heurísticas tiene una justificación considerable.

Existen otros argumentos en favor del uso de las técnicas heurísticas: lo que realmente sé esta optimizando es un modelo de un problema del mundo real, por eso no hay garantía de que la mejor solución del modelo sea también la mejor solución para el problema del mundo real. Las técnicas heurísticas son usualmente más flexibles y son capaces de hacer lo mismo con funciones objetivo y/o restricciones más complicadas que los algoritmos exactos. Este es el caso de técnicas tales como la *Búsqueda Tabú (Tabú Search);* donde las funciones objetivo no necesitan cumplir hipótesis de linealidad. Esto nos permite modelar problemas del mundo real aún más precisamente que con el uso de algoritmos exactos.

Muchos problemas de optimización combinatoria son problemas específicos, de manera que un algoritmo de una técnica heurística que funciona para un problema, puede no ser útil para resolver un problema diferente. Sin embargo, hay un gran interés en técnicas que se han desarrollado en la última década y que pueden ser aplicables con mayor generalidad.

Algunas de estas técnicas han sido desarrolladas bajo la *búsqueda local de vecinos (local neighborhood search).* El proceso inicia con una solución para un ejemplar y busca una mejor solución dentro de una *vecindad* definida. Habiendo encontrado una mejor solución, el proceso reinicia la búsqueda local con esta nueva solución y esto continúa iterativamente hasta que no pueda mejorar la solución actual encontrada. Esta solución final, probablemente sea un óptimo global, aunque con respecto a su vecindad es un óptimo local.

Una variación a lo anterior que ha ganado reciente atención es el permitir movimientos ascendentes, con lo cual la solución con la que la búsqueda local reinicia puede ser peor que la anterior, considerando que debe haber algunas restricciones para aceptar tales movimientos, en otro caso el procedimiento se sumaría a la búsqueda del espacio completo de soluciones. Por lo tanto se da la oportunidad de que el proceso pueda escapar o salir de óptimos locales y busque una mejor solución.

La heurística Búsqueda Tabú ha recibido considerable aceptación en los últimos años y ha habido muchas aplicaciones a varios problemas tales como: inventarios multiproducto[33], problema del agente viajero[20], problema de asignación cuadrática[40], problema de coloración de gráficas [34], problema de horarios [35], construcción de horarios para torneos [38], problema del club de bridge [37]; Con los cuales se ha alcanzado considerable éxito. Esta técnica es un ejemplo de los movimientos ascendentes para los óptimos locales.

Capítulo 3

Búsqueda Tabú

Se describirán las características principales y los elementos que integran la Búsqueda Tabú.

3.1 Introducción

La Búsqueda Tabú (BT) tiene sus antecedentes en métodos diseñados para cruzar límites de optimalidad local, normalmente tratados como barreras y manejados sistemáticamente para imponer y quitar restricciones que permitan la exploración de regiones de otra manera prohibidas. La forma moderna de la BT fue introducida por Glover [15].

El diccionario Webster define tabú o taboo como: "...cargado con un poder sobrenatural y prohibido, para profanar el uso o contacto", también menciona: "prohibido por la moral". La BT ciertamente no involucra referencias a cuestiones morales o sobrenaturales, pero en cambio está interesada en imponer restricciones que puedan guiar al proceso de búsqueda para negociar en otro caso regiones difíciles. Estas restricciones operan de varias formas; un ejemplo es por exclusión directa de ciertos movimientos clasificados como prohibidos.

La filosofía de la BT se basa en manejar y explotar una colección de principios inteligentes para la solución de problemas. Un elemento fundamental de la BT es el uso de la memoria flexible, la cual engloba los procesos de creación y explotación de estructuras computacionales para tomar ventaja de la historia de los movimientos realizados, esto es, la combinación de las actividades de adquisición y mejoramiento de la información.

La Búsqueda Tabú se basa en tres características principales:

1. El uso de estructuras computacionales de memoria basadas en los atributos de un movimiento, nos permiten aplicar criterios de evaluación y obtener información histórica, para mejorar el proceso de búsqueda; lo cual no sucede para estructuras con pérdida de memoria, es decir en los que no se recuerda, como algunos métodos volátiles.

2. Un mecanismo asociado de control, mediante el empleo de estructuras computacionales de memoria, basado entre las condiciones que restringen y liberan al proceso de búsqueda (restricciones tabú y criterio de aspiración).

3. La incorporación de funciones de memoria de distintos lapsos o ciclos de vida, desde memoria corta hasta memoria larga, para implantar estrategias que refuercen la combinación de movimientos y las características de solución que históricamente se han encontrado buenas, mientras que las estrategias de diversificación manejan la búsqueda dentro de nuevas regiones.

La estructura de la memoria de la BT opera por referencia a 4 dimensiones principales[38]:

1. Los movimientos más recientes (*recency*)
   Memoria que guarda los movimientos efectuados recientemente y que no serán elegidos nuevamente hasta salir de esta memoria, Lista Tabú.

2. Frecuencia (*frequency*)
   Número total de ocurrencia de todos los eventos.

3. Calidad (*quality*)
   Valor del movimiento efectuado.

4. Influencia (*influence*)
   Mide el grado de cambio inducido en una solución.

3.2 Marco teórico de la Búsqueda Tabú

3.2.1 Descripción

Para describir el funcionamiento de la BT, representamos el problema de optimización combinatoria de la siguiente manera

$$\text{Minimizar } c(x)$$
$$\text{sujeta a } x \in X$$

La función objetivo $c(x)$ puede ser lineal o no lineal, y la condición $x \in X$ supone que las restricciones especificadas de los componentes de x sean valores discretos.

3.2.2 Búsqueda local de vecinos

La BT puede ser caracterizada como una forma de búsqueda local de vecinos como se describió brevemente en el primer capítulo. En la búsqueda local de vecinos cada solución $x \in X$ tiene asociado un conjunto de vecinos $V(x) \subset X$, llamado la vecindad de x. Cada solución $x' \in V(x)$, puede alcanzarse directamente desde la solución x, aplicando una operación llamada movimiento, se dice que x cambia a x' cuando tal operación es realizada. Normalmente en la BT, las vecindades son simétricas, es decir, x' es un vecino de x si y sólo si x es un vecino de x'.

Método de la búsqueda local de vecinos

1. Inicialización

   - Seleccionar una solución inicial $x^{act} \in X$
   - Registrar la mejor solución $x^{mej} = x^{act}$ y mejor_costo = $c(x^{mej})$

2. Selección y Terminación

- Escoger una solución $x^{sig} \in V(x^{act})$.
- Si el criterio de cambio empleado no puede ser satisfecho por ningún miembro de $V(x^{act})$, esto es ninguna solución califica a ser x^{sig}.
- O si otro criterio de terminación es aplicado (tal como el número de iteraciones).
- Entonces el método se detiene.

3. Actualización

- Hacer $x^{act} = x^{sig}$, y si $c(x^{act}) >$ mejor_costo, realizar el segundo punto del primer paso. Regresar al segundo paso.

El método de la búsqueda local de vecinos puede ser alterado fácilmente para que nos dé algunos procedimientos clásicos. Los métodos descendentes, sólo permiten movimientos a soluciones vecinas que mejoren el valor actual de $c(x^{act})$, y terminan cuando la solución actual no puede ser mejorada.

Método Descendente

1. Inicialización.

   - Empezar con la inicialización de la búsqueda local de vecinos.

2. Selección y Terminación

   - Escoger $x^{sig} \in V(x^{act})$ para satisfacer $c(x^{sig}) < (x^{act})$ y termina si x^{sig} no puede ser encontrado.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos.

El defecto evidente del método descendente es que al final la x^{act} obtenida es un óptimo local, que en muchos de los casos no será un optimo global.

3.2.3 Características de la Búsqueda Tabú

La BT, en contraste con los métodos mencionados en capítulos anteriores, emplea una filosofía algo diferente para ir más allá de quedar atrapado en un óptimo local. La exploración de ciertas formas de memoria flexible para controlar el proceso de búsqueda es el tema fundamental de la BT. El efecto de tal memoria se obtiene estableciendo que la BT mantenga una historia selectiva H de los estados encontrados durante la búsqueda y reemplazando $V(x^{act})$ por una vecindad modificada la cual puede ser denotada por $V(H,x^{act})$. La historia determina por lo tanto, cuales soluciones pueden ser alcanzadas por un movimiento a partir de la solución actual, seleccionando $x^{sig} \in V(H,x^{act})$.

En las estrategias de la memoria corta, la vecindad $V(H,x^{act})$ es típicamente un subconjunto de $V(x^{act})$ y la clasificación tabú sirve para identificar elementos de la vecindad $V(x^{act})$ excluidos de $V(H,x^{act})$. En las estrategias de la memoria intermedia y memoria larga, la vecindad $V(H,x^{act})$ puede contener soluciones que no están en $V(x^{act})$, generalmente consiste en escoger soluciones elite (óptimos locales de alta calidad) encontrados en diferentes puntos del proceso de búsqueda. Tales soluciones elite son identificadas como elementos de un sector regional en estrategias de intensificación de memoria intermedia y como elementos de sectores diferentes en estrategias de diversificación en memoria larga, además los componentes de las soluciones elite, en contraste a las soluciones mismas, están incluidos dentro de los elementos que pueden ser retenidos e integrados para proveer entradas al proceso de búsqueda.

La BT también utiliza la historia para crear una evaluación modificada de la solución accesible actual. Formalmente, esto puede ser expresado diciendo que la BT reemplaza la función objetivo c(x) por una función c(H, *x)*, la cual tiene el propósito de evaluar la calidad relativa de la solución accesible actual. La relevancia de esta función modificada ocurre al uso de un criterio de elección agresivo para localizar un mejor candidato x^{sig}, es decir, el mejor valor de $c(H, x^{sig})$, sobre el conjunto de candidatos derivados de $V(H, x^{act})$. La referencia a *c(x)* es retenida para determinar si un movimiento nos conduce a una mejor solución.

Para problemas grandes, donde $V(H, x^{act})$ puede tener muchos elementos, o para problemas donde es demasiado costoso examinar esos elementos, la BT tiene como cualidad muy importante el poder aislar un subconjunto de candidatos de la vecindad y examinarlo en vez de examinar la vecindad completa. Esto puede ser hecho en etapas, permitiendo el subconjunto candidato ser ampliado si las alternativas de satisfacción de los niveles de aspiración no son encontradas. Nos referimos a este subconjunto explícitamente por la notación $Candidato\_N(x^{act})$. Entonces el procedimiento de la BT puede ser expresado de la siguiente manera:

Método básico de la Búsqueda Tabú

1. Inicialización.

   - Empezar con la inicialización de la Búsqueda local de vecinos, y con el registro de la historia H vacía.

2. Selección y Terminación.

- Determinar el subconjunto Candidato_N(x^{act}) como un subconjunto de V(H,x^{act}).
- Seleccionar $x^{sig} \in$ Candidato_N(x^{act}) para minimizar c(H, *x)* sobre este subconjunto.
- El candidato x^{sig} es llamado el mejor elemento del subconjunto Candidato_N(x^{act}).
- Terminar por el número de iteraciones si no se encuentra una solución óptima.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos, y adicionalmente actualizar el registro de la historia H.

3.2.4 Memoria de la Búsqueda Tabú

Un atributo de un movimiento de x^{act} a x^{sig}, o más precisamente de un movimiento de prueba x^{act} a una solución tentativa x^{pru}, puede abarcar cualquier aspecto que cambie como un resultado del movimiento. Por ejemplo, un movimiento que cambia el valor de dos variables simultáneamente.

El registro de los atributos de un movimiento es en ocasiones utilizado en la BT para imponer restricciones, llamadas restricciones tabú, que evitan elegir movimientos que inviertan los cambios presentados por estos atributos. Por ejemplo, un movimiento es Tabú sí: x_j cambia de 1 a 0, donde x_j cambió de 0 a 1.

Las restricciones tabú también son utilizadas para evitar movimientos que conduzcan a repeticiones o ciclos en un camino de búsqueda. Estas restricciones tienen el papel de evitar el regreso a un movimiento cuyos atributos produzcan una solución previamente encontrada. Por lo tanto las restricciones tabú varían de acuerdo a como son definidas.

Una restricción tabú es activada únicamente en el caso en que sus atributos ocurran dentro de cierto número de iteraciones anteriores a la iteración actual, creando la lista tabú de los movimientos más recientes, o que ocurran con cierta frecuencia a lo largo de las iteraciones, creando la matriz de frecuencia. Más precisamente, una restricción tabú es ejecutada únicamente cuando los atributos de la definición rebasen ciertos umbrales de frecuencia o sean iguales a los movimientos más recientes. Para entender esto, se define un atributo como tabú-activo cuando su atributo inverso asociado ha ocurrido en cierto intervalo de los movimientos mas recientes. Un atributo que no es activo es llamado tabú-inactivo.

La condición de ser tabú-activo o tabú-inactivo es llamado el estado tabú de un atributo. En algunas ocasiones un atributo es llamado tabú o no-tabú para indicar que es tabu-activo o tabu-inactivo.

Debemos señalar, sin embargo, que evitar el ciclado no es el último objetivo de BT. En algunos casos, un buen camino de búsqueda resultará en revisar los atributos de un movimiento que nos conduzcan a una mejor solución que cualquiera encontrada anteriormente. El objetivo es continuar estimulando el descubrimiento de buenos movimientos que conduzcan a nuevas soluciones de alta calidad.

3.2.5 Criterios de aspiración

Los criterios de aspiración se introducen en la técnica de BT para poder determinar cuándo una restricción tabú puede ser rechazada, eliminando la clasificación de tabú aplicada a cierto movimiento. El uso apropiado de este criterio de aspiración puede resultar muy importante para permitir así un buen nivel de desempeño de la BT.

Las primeras aplicaciones de la BT emplearon únicamente un tipo simple del criterio de aspiración, el cual consistía en rechazar la clasificación tabú de un movimiento de prueba cuando al evaluar la función objetivo se obtiene una mejor solución que la mejor solución encontrada hasta el momento. Este criterio de aspiración sigue siendo utilizado en muchas aplicaciones; sin embargo, otros criterios pueden mejorar también la efectividad de la Búsqueda Tabú.

Uno de los criterios de aspiración surge introduciendo el concepto de influencia, el cual mide el grado de cambio inducido en la estructura de la solución, la influencia es en ocasiones asociada con la idea de la distancia del movimiento, es decir, un movimiento de mayor distancia es concebido como el de mayor influencia.

Los movimientos de gran influencia son importantes especialmente durante los intervalos para romper con la optimalidad local, debido a que una serie de movimientos que hacen solo pequeños cambios estructurales es improbable que descubran un mejoramiento importante.

Las aspiraciones son de dos tipos: movimientos de aspiración y atributos de aspiración. Un movimiento de aspiración, cuando se satisface, rechaza el estado tabú del movimiento. Un atributo de aspiración, cuando se satisface, rechaza el estado tabú del atributo. En el último caso, el movimiento pudo o no cambiar su clasificación tabú, dependiendo si la restricción tabú puede ser activada por más de un atributo.

La esencia del método depende de cómo el registro de la historia H es definido y usado, y cómo la vecindad Candidato_N(x^{act}) y la evaluación de la función c(H, *x)* son determinadas. El método de la BT utilizado introduciendo el criterio de aspiración y la memoria larga es el siguiente:

Método mejorado de la Búsqueda Tabú

1. Inicialización.

   - Empezar con la obtención de vecinos, con el registro de la historia H vacía y a la matriz de frecuencia se le asignan ceros en todas las entradas como valor inicial.

2. Selección y Terminación.

- Determinar el subconjunto Candidato_N(x^{act}) como un subconjunto de V(x^{act}).
- Inicializar mínimo y $mínimo_p$ con un valor alto.
- Generar todos los vecinos $x^{sig} \in$ Candidato_N(x^{act}), cuyo costo c(x^{sig}) ≤ mínimo.
- Aplicar el criterio de aspiración.
- Si c(x^{sig}) < mejor_costo
- aceptar de inmediato a x^{sig}
- asignar $mínimo_p$ = mínimo = x^{sig}.
- Si no
- penalizar a x^{sig} con la matriz de frecuencia, pena = x^{sig} + frecuencia(i, j).
- Verificar que el movimiento no sea tabú

- Si pena ≤ $mínimo_p$.
- aceptar el movimiento
- $mínimo_p$ = pena y mínimo = x^{sig}.
- Terminar si encuentra una solución óptima o por un número límite de iteraciones.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos, y adicionalmente actualizar el registro de la historia H, así como también la matriz de frecuencia.

Capítulo 4

Paralelismo

Se presentan las características principales de la computación paralela, como son los diferentes tipos de memoria, y se mencionarán algunas herramientas de programación y bibliotecas.

4.1 Introducción

Una demanda por sistemas más rápidos y eficientes, tanto en el área de computación científica como en el área comercial, ha ido en constante aumento al correr de los años. Los límites físicos existentes para un aumento de la velocidad en un único procesador y su alto costo nos llevan al surgimiento de nuevas arquitecturas para sistemas de cómputo de alto desempeño. Estas máquinas pueden ser clasificadas en tres tipos:

Multiprocesadores vectoriales: poseen un pequeño número de procesadores, donde cada uno es un procesador vectorial de alto desempeño.

Sistemas MPP (Massively Parallel Processors): poseen de centenas a millares de procesadores interconectados, constituyendo una línea principal de computadoras paralelas. Pueden ser de memoria distribuida o compartida.

Estaciones de trabajo conectadas por redes: máquinas interligadas por red de media y alta velocidad, que pueden trabajar como una máquina virtual de alto desempeño.

4.2 ¿Qué es computación paralela?

Computación paralela es el uso de más de una Unidad de Procesamiento Central (CPU) para la solución un problema, Fig. 4.1. La computación paralela abarca el campo entero de las máquinas. Desde dos computadoras personales conectadas vía Ethernet a miles de los más poderosos procesadores en una supercomputadora paralela.

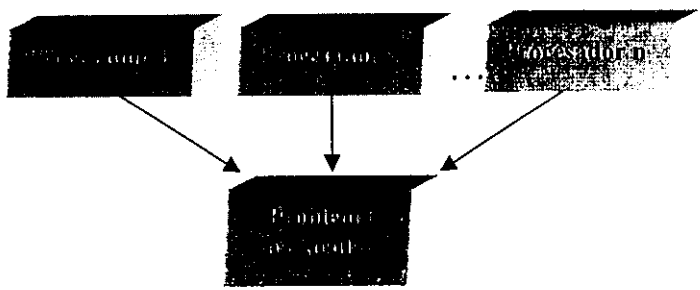


Fig. 4.1: Computadora paralela usando más de un procesador para resolver en conjunto un problema particular.

4.3 Terminología

Algunos de los términos utilizados en procesamiento paralelo son descritos a continuación:

- Tarea o proceso: es la unidad lógica mínima definible de un programa.
- Ejecución secuencial: ejecución de un programa o tarea en un único procesador, donde una instrucción se procesada independientemente.
- Paralelización de código: Transformación de un programa secuencial en uno paralelo, a través de la identificación de tareas independientes para un programa, de manera que se ejecuta de modo paralelo. Generalmente requiere modificaciones al código de un algoritmo utilizado en modo secuencial.

- Aceleración de un programa (Speedup): Razón entre los tiempos de ejecución de las versiones seçuencial y paralela de un mismo programa o algoritmo. Para un determinado algoritmo, se considera la mejor implementación secuencial como referencia para comparar con la implementación paralela

$$\text{aceleración (Speedup)} = \frac{\text{tiempo de ejecución de un programa secuencial}}{\text{tiempo de ejecución de la versión paralela}}$$

- Eficiencia de un programa: Razón entre la aceleración de un programa y el número de procesadores utilizados para la ejecución del mismo.

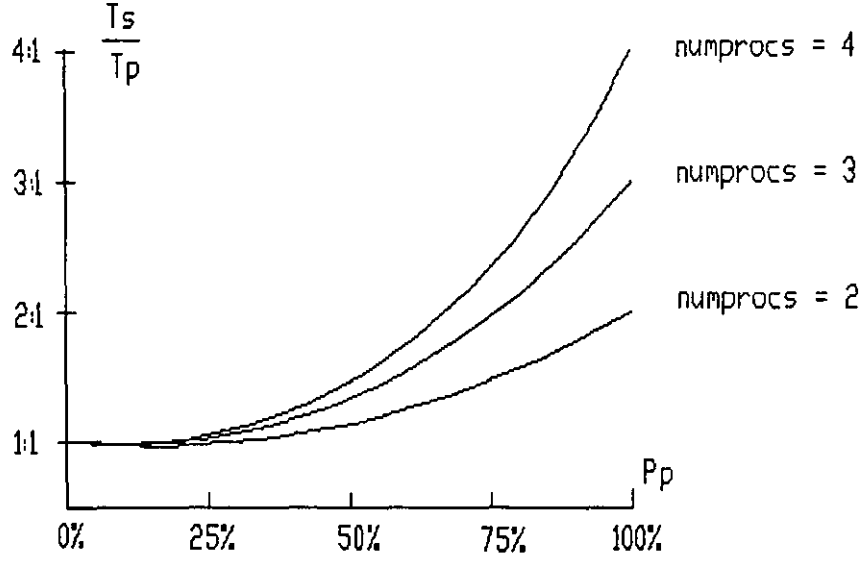
$$\text{eficiencia} = \frac{\text{aceleración}}{\text{número de procesadores}}$$

Intuitivamente este parámetro o medida es la fracción de tiempo que un procesador está siendo utilizado para ejecutar el algoritmo.

Idealmente, si p procesadores utilizan 100% de su tiempo solamente para operaciones de computación del algoritmo, la eficiencia sería igual a 1. Como un programa paralelo requiere comunicación entre los procesadores, parte del tiempo de los procesadores es gastado en esta actividad, llevando a valores de eficiencia menores a 1.

- Ley de Amdahl: Cantidades posibles de aceleración debido a la paralelización de porciones de código. Es decir, la cantidad que podemos acelerar el proceso en relación al tiempo secuencial, dependiendo de la cantidad de código que es factible paralelizar y el número de procesadores, Gráfica 5.1.

$$Tp = Ts\left(\frac{Pp}{Np} + Ps\right)$$



Gráfica 5.1

Ts	TIEMPO DE CORRIDA DE PROGRAMA SECUENCIAL
Tp	TIEMPO DE CORRIDA DE PROGRAMA EN PARALELO
Ps	PORCENTAJE DE TIEMPO USADO CORRIENDO EN SERIAL
Pp	PORCENTAJE DE TIEMPO USADO CORRIENDO EN PARALELO
Np	NUMERO DE PROCESADORES

La gráfica nos muestra la aplicación de la Ley de Amdahl, en donde vemos la eficiencia de los procesadores utilizados en relación al porcentaje del código que ha sido paralelizado, ejemplo:

Si la porción de código que es factible programarse en paralelo es el 100% y utilizamos 4 procesadores entonces: el tiempo de CPU sería la cuarta parte del tiempo que se utilizó en la versión en serie, hay que recordar que este sería el caso ideal.

- Sincronización : Coordinación temporal entre procesadores, usada para un paso de información entre ellos. La sincronización es un factor de decremento de la aceleración exigida por un programa paralelo, debido a que algunos procesadores pueden estar inactivos, esperando que otros procesadores terminen sus respectivas tareas. Una comunicación entre procesadores se dice asíncrona cuando exige una coordinación entre tareas.

- Granulado: Volumen de procesamiento realizado por cada tarea, en relación al volumen de comunicación existente entre las tareas. Un programa de granulado muy fino posee tareas que realizan pocas instrucciones y necesitan comunicarse mucho, en cuanto a un programa de granulado grueso posee tareas que ejecutan muchas instrucciones y se comunican poco. Cuando la comunicación entre los procesadores sea asíncrona, el granulado afecta el tiempo de ejecución de un programa, esto es porque programas con granulado fino necesitan sincronización más frecuente, introduciendo periodos de inactividad de los procesadores y aumentando el tiempo de ejecución.

- Escalabilidad: Este es un sistema definido por un algoritmo paralelo en una máquina paralela y escalable cuando la aceleración obtenida crece proporcionalmente al número de procesadores utilizados por el hardware. Los factores que contribuyen para la escalabilidad de un sistema son el hardware, el algoritmo utilizado y el código implementado.

- Balanceo de carga: Distribución de las tareas entre los procesadores, a modo de garantizar la ejecución lo más eficiente posible del programa en paralelo. Este balance puede ser visto de manera estática, antes del inicio de la ejecución, o de manera dinámica durante el tiempo de ejecución.

- SPMD (Single Program Multiple Data): Modelo de programación bastante utilizado, donde todos los procesadores ejecutan un mismo programa sobre diferente tipos de datos.

- Determinismo: Un algoritmo o programa es determinístico si su ejecución con una entrada específica siempre resulta una misma salida. Se dice que un programa es no deterministico si existe la posibilidad de obtener salidas diferentes para varias ejecuciones del programa, utilizándose siempre una misma entrada.

4.4 Modelos de computación paralela

En éste apartado se presenta un breve clasificación de los modelos de cómputo paralelo.

Clasificaciones según Flynn: SISD, MISD, SIMD, y MIMD

En 1966 Flynn[41] publica sus estudios sobre arquitecturas en paralelo, los cuales establecieron la terminología tradicional. Debido a los avances tecnológicos de las últimas dos décadas, la relevancia de esta taxonomía ha disminuido, pero la terminología ha sido muy usada para describir las computadoras en paralelo.

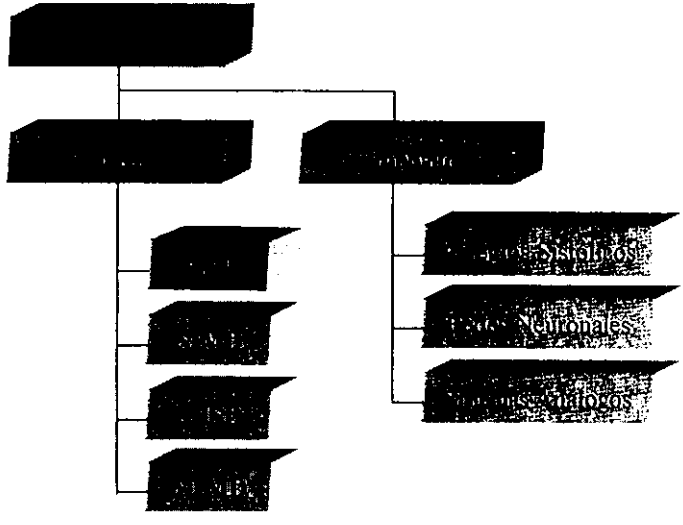


Fig. 4.2: Diagrama de la taxonomía computacional. La computadora de propósito general más común es la SISD

En 1966[41] Flynn establece las cuatro clases de arquitecturas de máquinas Figura. 4.2.

SISD ---- Single Instruction Single Data;
SIMD ---- Single Instruction Multiple Data;
MISD ---- Multiple Instruction Single Data;
MIMD ---- Multiple Instruction Multiple Data;

Haremos referencia a ellas por sus siglas en inglés. A continuación las describimos:

La arquitectura SISD es la arquitectura no paralela, por ejemplo la PC convencional.

La arquitectura SIMD consiste de un número de procesadores idénticos procesando un ciclo de pasos sincronizados y realizando el proceso sobre diferentes datos.

El modelo MISD es la arquitectura teórica propuesta por flynn pero que no tuvo uso.

El modelo MIMD cubre un vasto número de esquemas de interconexión, de procesadores y arquitecturas. El punto clave es que cada procesador opera de forma independiente de los otros, potencialmente corriendo programas totalmente diferentes. Los procesadores de MIMD se comunican utilizando una red de alta velocidad. La red permite que los procesadores compartan datos y sincronicen cálculos. Raro es el problema en paralelo que no necesita ninguna comunicación o sincronización entre procesadores. Las máquinas MIMD modernas utilizan explícitamente un paradigma de paso de mensajes para comunicar los procesadores. Un procesador en específico envía a un procesador destino, el cual recibe el dato.

4.5 Modelos de organización de memoria

Existen dos tipos básicos de organización de memoria: memoria compartida y memoria distribuida [43,44], los cuales se describen a continuación.

Memoria compartida

En las máquinas que utilizan memoria compartida, todos los procesadores pueden acceder a cualquier dirección de memoria, conforme a lo mostrado en la Figura 4.3. La sincronización entre tareas es realizada a través de un control de las operaciones de lectura y escritura ejecutadas por las tareas de memoria compartida. Una ventaja de este tipo de acceso es que el compartir de datos puede ser de manera rápida. Una desventaja es el nivel de escalabilidad del sistema, el limitado número de caminos existentes entre los procesadores a la memoria.

Una manera de eliminar la desventaja anterior consiste en proveer a cada procesador con una memoria local. La cual almacena un programa para ser ejecutado y la estructuras de datos que no son compartidas entre los procesadores. Las estructuras globales están almacenadas en memoria compartida, conforme a lo ilustrado en la Figura 4.4.

Una extensión de este tipo de arquitectura anterior consiste en la eliminar totalmente el espacio de memoria que se comparte, conforme a lo mostrado en la Figura 4.5 En ambos casos, estas referencias vistas por un procesador en la memoria de otro procesador son controladas por el hardware de interconexión de memoria con los procesadores. Se debe observar que los tiempos de acceso a memorias locales son típicamente menores que los tiempos a memoria remota. Un factor importante para la aplicación correcta de este modelo y el control sobre el acceso a memoria compartida de las tareas. Por ejemplo una tarea no puede alterar un dato mientras otra tarea esta leyendo este mismo dato.

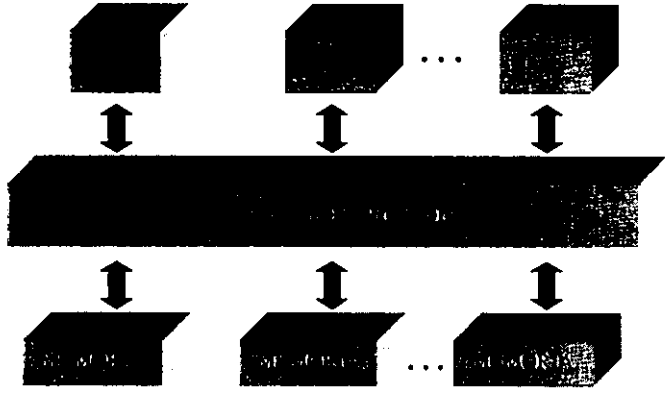


Fig. 4.3 Arquitectura de memoria compartida.

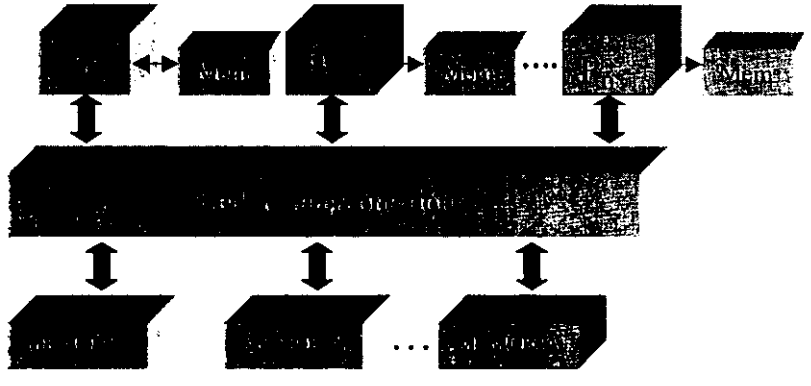


Fig. 4.4 Arquitectura de memoria compartida con memoria local en los procesadores.

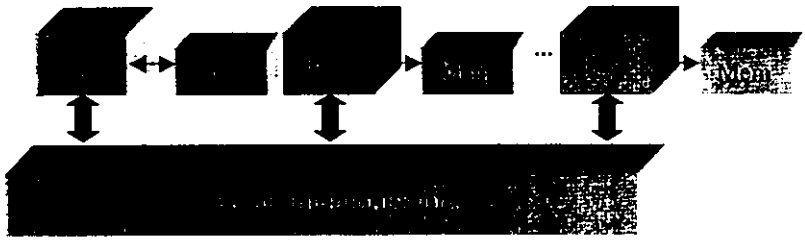


Fig. 4.5 Arquitectura de memoria compartida solamente con memoria local en los procesadores.

Memoria distribuida

En máquinas que utilizan memoria distribuida y distribución de memoria física entre los procesadores, cada memoria local sólo puede dar acceso al procesador al cual esta asociada. El intercambio entre las tareas se realiza a través de mensajes entre las redes de comunicación, conforme a lo ilustrado en la Figura 4.6.

El envío y recibimiento de mensajes entre los procesadores pueden ser vistos en forma con-espera o sin-espera. En el envío con-espera la ejecución de una tarea que genera el envío de un mensaje es interrumpida antes que el mensaje sea recibido por el receptor; en cuanto que en la forma sin-espera inicia el proceso de envío y continúa su ejecución sin aguardar a confirmar que el mensaje fue recibido por el receptor. En el recibimiento con-espera, una tarea es interrumpida cuando espera la llegada de un determinado mensaje. En la forma sin-espera se verifica si el mensaje dejado está disponible, en el caso de que no esté, se continúa con el procesamiento.

Las ventajas de utilizar programación distribuida son la escalabilidad del sistema, la posibilidad de mayor confianza en el sistema a través del duplicado de datos entre los procesadores y, consecuentemente, tolerancia a fallas y la facilidad de la incorporación de procesadores más especializados. Las desventajas son la difícil implementación de algunos algoritmos, tales como la descomposición de datos en un arreglo, y la reestructuración de implementaciones ya existentes, por ejemplo en aplicaciones secuenciales o en aquellas que utilizan memoria compartida.

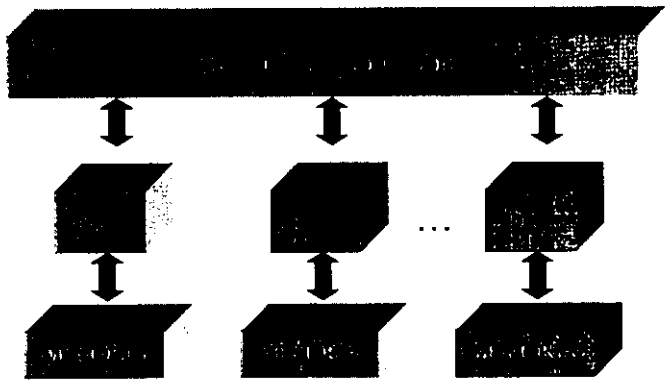


Fig. 4.6 Arquitectura de memoria distribuida.

Debe hacerse notar la semejanza entre la arquitectura de memoria compartida mostrada en la Figura 4.5 y la de memoria distribuida mostrada en la Figura 4.6. La mayor diferencia entre ellas es que, la red de interconexión permite que los procesadores ejecuten operaciones de lectura y de escritura en memorias pertenecientes a otros procesadores. En la segunda el acceso a memoria de otro procesador tiene que ser visto de manera explícita a través del paso de mensajes entre los procesadores.

4.6 Métodos de programación

Como ya comentamos anteriormente, la programación paralela es una disciplina relativamente reciente, donde los patrones todavía no están bien definidos. De una forma general los métodos de programación paralela, actualmente utilizados, pueden ser clasificados en tres modelos[36]:

- **Paralelismo de datos**: los datos son distribuidos entre los procesadores que ejecutan una misma secuencia entre ellos. Un factor importante para la implementación eficiente de este tipo de programación es un direccionamiento de la localización de datos. Para que un programa de este tipo se ejecute eficientemente en máquina de memoria distribuida, los datos deben estar alojados en la memoria de procesadores próximos a los que van a utilizar estos datos. Este modelo también es conocido como descomposición de dominio.

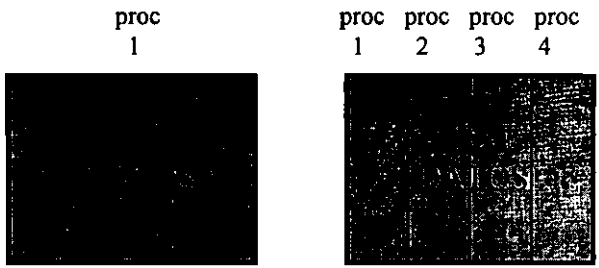


Fig. 4.7: Un conjunto de datos puede ser particionado en varios pedazos.

- **Paralelismo de tareas:** El programa es fraccionado en tareas corporativas. Estas tareas pueden ejecutar diferentes procedimientos entre sí, cuyo procesamiento puede ser visto de manera asincrona. Para mayor eficiencia de la implementación de los programas debe tomarse en cuenta el direccionamiento de la localización de los datos en un nivel de granuladidad de cada tarea. Este modelo también es conocido como descomposición funcional.

Un procesador

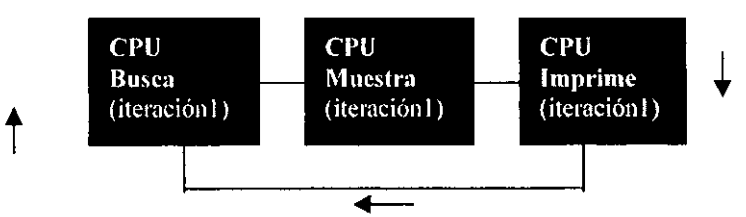


Fig. 4.8 Secuencia del proceso en serie.

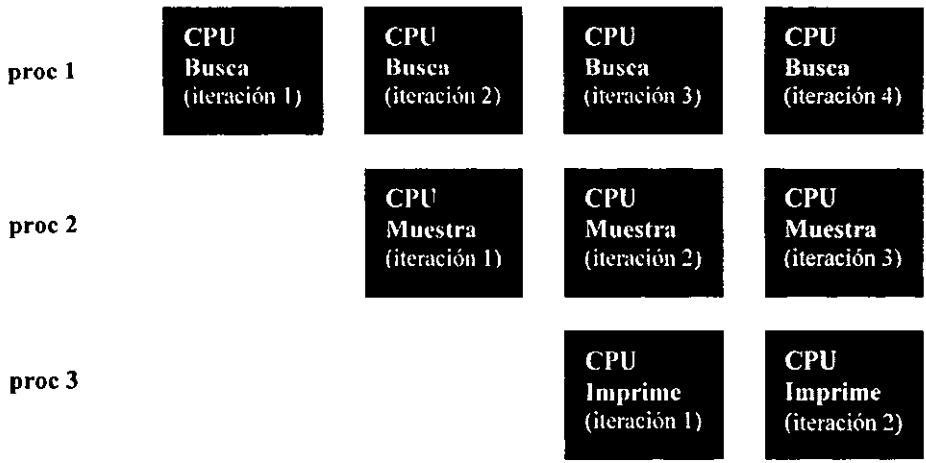


Fig. 4.9. Secuencia de un proceso en paralelo con 3 procesadores

- **Paralelismo de objetos**: en este caso, el paralelismo puede ser realizado de diferentes formas. Por ejemplo, (1) una función miembro de un objeto puede llamar una función publica de otro objeto remoto o no, y (2) un tipo abstracto de datos, como un arreglo, puede ser implementado de una manera distribuida. Son necesarios los mismos cuidados relacionados a la eficiencia discutidos con anterioridad con relación a la localidad de los datos y a la granuladidad de las tareas.

Dependiendo de la aplicación, un modelo podrá ser más adecuado que otro. En muchas aplicaciones, una combinación de los paradigmas puede ser útil para un mejor desempeño del programa.

En este trabajo utilizamos el paralelismo de datos debido a que la mayor cantidad de tiempo consumido por el programa se ubica en el cálculo de la energía de las conformaciones obtenidas, es decir particionamos el espacio de búsqueda entre el número de procesadores a utilizar y cada procesador realiza un número semejante de evaluaciones,

al final se comparan los datos y se elige el mejor*. Esto lo realizamos empleando las bibliotecas para el paso de mensajes MPI (Massage Passing Interface).

4.7 Herramientas para ambientes de memoria distribuida

Como en el caso de programación secuencial, existen varios lenguajes y herramientas para la implementación de programas en paralelo, siendo que cada una de estas representa una mejor opción para una determinada clase de problemas. Estas están divididas en dos clases: **lenguajes paralelos y bibliotecas para paso de mensajes**. La primera clase abarca los lenguajes que poseen instrucciones especiales para la creación de aplicaciones paralelas por los usuarios. La segunda clase contempla las herramientas que ofrecen funciones de una biblioteca de funciones para que el usuario realice la comunicación entre las tareas de su programa paralelo.

Lenguajes paralelos: La mayor parte de estos lenguajes poseen extensiones vistas en lenguajes secuenciales ya existentes, de modo de tornar transparentes para el usuario las primitivas de comunicación utilizadas en el sistema de computación, como una tentativa de facilitar la implementación de programas paralelos. Estos lenguajes poseen un compilador que transforma el programa del usuario y envía el resultado a un compilador de lenguaje secuencial; como ejemplos de lenguaje paralelo pueden citarse a: *HPF (High performance Fortran), Fortran-M, Network Linda, CC++, Regis, SR,* [41].

Bibliotecas para paso de mensajes: Una manera simple de agregar nueva funcionalidad a un lenguaje de programación a través de la creación de bibliotecas de software. El usuario realiza llamadas a nuevas rutinas de biblioteca para procesar las nuevas funciones, sin que el lenguaje original tenga que ser modificado. Debido a la facilidad de implementación antes mencionado, han sido desarrolladas varias herramientas basadas en bibliotecas de paso de mensajes para facilitar el desarrollo de programas paralelos; por ejemplo: *PVM (Parallel Virtual Machine), P4, Express, ISIS, MPI,* [41].

* Se da una explicación más detallada en el capítulo 5

4.8 MPI (Massage Passing Interface)

MPI no es propiamente un sistema implementado, pero sí una propuesta de estandarización para una interfaz de paso de mensajes para máquinas paralelas con memoria distribuida. La idea de estandarizar esta interface es la de hacer posible la portabilidad de los programas entre las diferentes máquinas y de facilitar su uso.

El núcleo de MPI está formado por rutinas para comunicación punto a punto entre pares de proceso. Estas rutinas pueden ser llamadas de forma con-espera o sin-espera. Existen tres modos de comunicación: *lectura (ready)*, en el cual un mensaje puede ser enviado a una operación de recepción correspondiente sin haber sido iniciada; *estándar (standard)*, en la cual un mensaje puede ser enviado independientemente de haber una operación de recepción para él; y sincronizado (synchronous), que funciona de manera semejante al modo estándar, con la diferencia de que una operación de envío no será considerada terminada hasta que la operación de recepción correspondiente sea iniciada en el proceso destino.

Otras facilidades que ofrecen las rutinas para grupos de procesos, es que coordinan la comunicación entre los procesos pertenecientes a un mismo grupo. Estas rutinas permiten una transmisión colectiva de datos y otorgan las siguientes facilidades:

- Envío de un mismo dato por un participante del grupo para todos los demás (broadcast).
- Envío de datos diferentes de un determinado participante de un grupo para todos los demás.
- Envío de datos de todos los participantes de un grupo para un determinado participante
- Envío de datos de cada participante para todos los demás.
- Envío de datos diferentes de cada participante para todos los demás.

Capítulo 5

Programación de Búsqueda Tabú

Un algoritmo de BT no puede ser aplicado a cualquier problema de optimización combinatoria; debido a que cada problema se plantea de forma diferente. A continuación explicamos como desarrollamos los programas para resolver el problema de plegamiento de proteínas.

5.1 Programas

5.1.1 Generación de ejemplares

Los puntos más importantes en la implementación de la BT, para nuestra aplicación en particular, son: El espacio de búsqueda X, la función objetivo *f*, la función de vecindad N(*x*), el método para encontrar la solución inicial, la longitud de la lista tabú *T*, el criterio de aspiración, la función memoria larga y el criterio de paro.A continuación los describimos de forma más detallada.

El espacio de búsqueda.

En nuestra aplicación, *k* denota el número de residuos aminoácidos en la molécula. Así como ϕ_i, ψ_i, y ω_i denotan los ángulos dihedros del esqueleto correspondiente al *i*-ésimo residuo aminoácido, donde $i = 1, \ldots, k$ y $\chi_1, \ldots, \chi_m$ son los ángulos dihedros en la cadena lateral. Sea Ω el conjunto de ángulos con valores enteros entre -180° y 180°, entonces un vector $x = (\theta_1, \ldots, \theta_n) = (\phi_1\psi_1\omega_1, \ldots, \phi_k\psi_k\omega_k, \chi_1, \ldots, \chi_m) \in \Omega^{3k+m}$ y x determina una conformación tridimensional de la molécula.

El espacio de búsqueda queda definido como:

$$X = \{x = (\theta_1, \ldots, \theta_n) \in \Omega^{3k+m} \mid 180 - \sigma_0 \leq \omega_1 \leq 180 + \sigma_0, -\sigma_1 \leq \phi_i + \psi_{i+1} \leq \sigma_1, i=1,\ldots,k\} \quad (5.1)$$

La función objetivo $f(x)$ es la función de energía ECEPP/3

La función de vecindad $N(x)$.

Un término utilizado en las restricciones es el de los ángulos anticorrelacionados, éstos se observan en la restricción 2.1-B (ϕ,ψ), se argumenta en los reportes de Levitt[24] y Post[25] sobre experimentos de dinámica molecular en dos diferentes proteínas. Ambos artículos mencionan que los cambios en la orientación del grupo de péptidos son debido a grandes cambios en la pareja de ángulos (ϕ_i,ψ_{i+1}), especialmente cuando las cadenas laterales están en espacios cerrados y dan vueltas con giro contrario. Levitt señala que donde la simulación de dinámicas moleculares mantenía igual al ángulo ϕ_i y colocaba su opuesto en ψ_{i+1}, valores más favorables podían ser obtenidos, desde entonces se reorienta el grupo de péptidos sin cambiar la conformación de la cadena. Además reporta que la localización natural del movimiento, donde ϕ_i y ψ_{i+1} son anticorrelacionados, conduce a largos desplazamientos únicamente en algunos residuos sin tener muchos efectos en la trayectoria de la cadena.

La inclusión de las anticorrelaciones en algunos de los ángulos dihedros impone la necesidad de redefinir la función de vecindad en términos de los movimientos que transforman el vector x en x'. Se define un movimiento simple m_s para θ_i perteneciente al conjunto de los ángulos no anticorrelacionados, si este es un vector $m_1 = (i, \theta_i, \theta_i')$ con $1 \leq i \leq n$ y $-180° \leq \theta_i, \neq \theta_i' \leq +180°$ que transforma el vector $x = (\theta_1, \ldots, \theta_i, \ldots, \theta_n)$ en $x' = (\theta_1, \ldots, \theta_i', \ldots, \theta_n)$.

Tomando en cuenta los ángulos anticorrelacionados, es necesaria la definición de un movimiento compuesto m_c formado por dos movimientos m_1 y m_2. El movimiento m_1 fue

definido anteriormente, y el movimiento m_2 es definido como $m_2 = (j, \theta_j, \theta_j')$, tal que $\theta_i = \phi_l$ y $\theta_j = \psi_{j+1}$ donde $i \leq l \leq k\text{-}1$, tal que $\phi_l' + \psi_{l+1} \leq \sigma_1$. En el último caso este movimiento compuesto transforma a $x = (\theta_1,...,\theta_i,...,\theta_j,...,\theta_n)$ en $x' = (\theta_1,..., \theta_i',..., \theta_j',...,\theta_n)$ de acuerdo a esta definición del movimiento general que transforma a x en x' es determinado por el vector (i, θ_i, θ_i'). En cada iteración generamos el conjunto V* como sigue:

$$x' \in V^* \subset N(x) \Leftrightarrow p < y,$$

Donde y es un número entre 0 y 1, y $p \sim [0,1]$, los cuales fueron experimentalmente ajustados.

La longitud de la lista tabú T.

La lista Tabú es construida por tripletas (i, θ_i, θ_i'). Si un movimiento m_s o un movimiento compuesto m_c es efectuado, ningún movimiento $m_l = (l, \theta_l, \theta_l')$ es declarado prohibido durante las t iteraciones si $\theta_l' \in [\theta_l - d, \theta_l + d]$ donde $l = i$ para m_1, y $l = j$ para m_2; El valor de d es determinado experimentalmente en un intervalo entre 0°y 20° . El número de t iteraciones fue determinado experimentalmente como bueno.

El criterio de aspiración.

En este trabajo se utiliza el criterio de aspiración estándar. Este criterio permite al estatus tabú moverse de x a x' siendo removido si el valor de $f(x')$ es estrictamente menor que el mejor valor obtenido hasta entonces, Esto significa que el estatus tabú de un movimiento de x a x' pueda ser descartado si $f(x') < f(x^0)$, donde x^0 es la mejor solución encontrada hasta entonces.

Función de memoria larga.

La finalidad de la función memoria larga es diversificar la búsqueda hacia regiones convincentes, esto es, visitando las que de otra manera no fueron exploradas anteriormente. En la implementación de la función memoria larga B es un vector de dimensión n. El vector tiene ceros al comienzo del proceso. El movimiento $m = (i, \theta_i, \theta_i')$ fue perfeccionado. El contador correspondiente en el vector B es cambiado como sigue: $b_i = b_i + 1$, y los valores del movimiento que no mejoran en la función objetivo por medio de cambios en el valor de el ángulo dihedro θ_i son incrementados por b_i.

El criterio de paro.

El último concepto por explicar es el criterio de paro, por el cual el proceso de búsqueda puede ser detenido. Este es establecido que si el número de iteraciones, sin una mejora en la mejor solución, es mayor que un límite *nimax* entonces el proceso es detenido. El sentido del *nimax* llegará a estar claro en la sección de resultados.

Generalmente, ningún método interactivo de búsqueda local es utilizado debido a los largos tiempos computacionales que requiere. No obstante, si la búsqueda no puede ser mejorada en una computadora con un solo procesador, Es posible hacerlo usando un sistema de multiprocesadores para acelerar la búsqueda. Para incrementar el número de iteraciones por unidad de tiempo podemos realizar uno de los tres caminos siguientes:

- Acelerar los cálculos de cada iteración,
- Ejecutar varios movimientos simultáneamente, y
- Ejecutar varias búsquedas independientes.

La primera opción implica paralelización de la evaluación de la función objetivo, el movimiento de valores, o incluso los cambios de él mejor movimiento. En la segunda opción el problema principal es la descomposición o duplicación, en el sentido que si varios movimientos han sido mejorados simultáneamente, pueden surgir problemas de sincronización. Si los movimientos se toman independientes entonces el problema puede ser descompuesto. En esta paralelización uno puede imaginar una descomposición donde la evaluación del movimiento es un costoso procedimiento secuencial, pero esto es posible al ejecutar al mismo tiempo la elección del mejor vecino. Esta es la aproximación utilizada en este trabajo, dado que en la búsqueda tabú el componente que consume mayor cantidad de tiempo de CPU es la evaluación de la solución actual disponible. De este modo, la velocidad absoluta de BT puede mejorarse si esto es escrito para ejecutarse en paralelo. En una versión en serie para BT cada vecino es evaluado secuencialmente, y un nuevo vecino no es generado hasta que el anterior fue revisado. En una versión en paralelo de BT varios vecinos pueden ser evaluados al mismo tiempo, tomando ventaja de que cada vecino es independiente del resto.

El algoritmo fue implementado para particionar el conjunto de posibles movimientos en ρ subconjuntos de aproximadamente la misma medida, y evaluar cada partición en ρ diferentes procesadores. En este camino cada procesador encuentra su mejor movimiento. Si un procesador es elegido, por ejemplo el procesador cero, recibe los $\rho - 1$ mejores movimientos propuestos por los otros procesadores, y toma el mejor dentro de esos ρ movimientos, incluido el propio. Después, este procesador comunica el movimiento

elegido a los demás procesadores, lo cual permite a todos los procesadores llevar a cabo el mejor movimiento y la actualización de la lista tabú, la matriz de frecuencia y la función de aspiración.

La configuración de la red usada por el BT en paralelo, es una estructura de árbol, en la cual cada procesador no cero (nodo) es conectado al nodo cero, Figura.7.1.

Esta técnica requiere intensa comunicación dado que dos sincronizaciones entre el procesador cero y cada uno de los demás procesadores son inevitables para cada iteración. Sin embargo, dado que nuestra asignación de carga de trabajo aproximadamente es la misma para cada procesador, podemos suponer entonces que se toma aproximadamente la misma cantidad de tiempo entre sincronizaciones. Además, el tiempo de comunicación para enviar y recibir la operación es significativamente más pequeño que la cantidad de carga de trabajo realizada al mismo tiempo entre sincronizaciones.

Para los experimentos de paralelización se uso una computadora SGI/CRAY ORIGIN 2000 localizada en DGESCA UNAM con 32 procesadores y un sistema operativo IRIX 6.4, físicamente una máquina de memoria distribuida, pero lógicamente resulta ser una máquina de memoria compartida. Todos los programas fueron escritos en lenguaje C estándar y se empleó un sistema de paso múltiple de mensajes entre procesadores MPI. El código de ECEPP/3, escrito en Fortran, fué re-codificado a C usando la herramienta f2c. El código resultante fue modificado para adecuarse a la versión en paralelo de BT.

5.2 Búsqueda Tabú en paralelo (MPI)

Enseguida se definirán las variables utilizadas en el programa, el cuál consta de una función principal denominada ecepp que llama a una serie de funciones, que utilizan bases de datos como grad5, o subrutinas como nrgy4, bd11, bdgnr, gener, param, paramd, specv, *tabu*, Todas estas subrutinas pertenecen a la función de energía ECEPP/3, excepto *tabu*, la cual fue agregada como el método heurístico que va ha orientar la búsqueda del mínimo global de energía.

Ahora describiremos cómo se programo el algoritmo de la búsqueda *tabu*. Sólo haciendo mención de que se tuvieron que adaptar las demás funciones para poder paralelizar todo el programa, dada la compatibilidad de datos entre subrutinas.

Existen varios tipos de variables utilizadas en diferentes niveles del programa:

Variables globales declaradas en la función principal.

Variables locales declaradas en la función principal.

Variables externas provenientes de la función principal y declaradas como globales en las subrutinas.

Variables globales declaradas en las subrutinas.

Variables locales de cada función perteneciente a cada subrutina.

Como variables externas y declaradas como globales en *tabu* tenemos:

seed semilla para dar valor inicial a la función generadora de números aleatorios.

long_tabu longitud de la lista tabú.

max_iter máximo número de iteraciones.

dist_tabu intervalo mínimo permitido entre un mínimo anterior y uno actual.

fun_aspi función de aspiración.

Np número de procesadores utilizado en ese ejemplar.

pn identificador del procesador.

Variables globales:

minimo estructura de dimensión igual al máximo número de procesadores disponibles, posee seis campos, donde se almacena la energía, los ángulos, la posición, y las posiciones de los ángulos correlacionados.

M Número máximo de datos a enviar o recibir por un procesador.

max_Np Máximo número de procesadores disponibles.

Nv Arreglo de dimensión igual al máximo número de procesadores disponibles, donde se almacena la cantidad de vecinos evaluados por cada procesador.

Timep Arreglo de dimensión igual al máximo número de procesadores disponibles, donde se almacena el tiempo empleado por cada procesador en un ejemplar.

Procedimientos

Los principales procedimientos requeridos por tabú son:

inicia_angulos(*n, x*). Recibe como parámetros: *n*, el número de ángulos de la proteína y el arreglo *x* con los valores de cada ángulo y su posición *(x)*. Cambiamos los valores de los ángulos aleatoriamente entre -180° y +180°, respetando su posición; y los ángulos de las posiciones específicas 2,8,11,14, y 19, ángulos anticorrelacionados, se mueven aleatoriamente entre los valores $-180° \pm 10°$ y $+180° \pm 10°$.

estatus_tabu(mov_tabu, p, ángulo). Recibe como parámetros una estructura de dimensión 100, con 2 campos, que guardan los movimientos que son tabú, además de un entero que determina la posición del ángulo *p* y el ángulo que cambiamos, este procedimiento determina si el movimiento es tabú, es decir si se encuentra dentro de la lista tabú.

tabu_ (n, d__, *x*, calcf, calcg, iv, liv, lv, v, uiparm, urparm, ufparm, nom, atoms_1, calc_1, eng000_1). Recibe varios parámetros, todos ellos provenientes de las demás subrutinas pertenecientes a ecepp, se pasan como apuntadores a las direcciones de estas variables en las subrutinas donde han sido declaradas (principalmente en ecepp), como hemos mencionado anteriormente, con el fin de efectuar el menor número de llamadas entre subrutinas.

En tabu_ sé dan valores iniciales a las diferentes variables requeridas, las cuales son locales en cada procedimiento, no se mencionaran debido a que son índices o variables temporales utilizadas para realizar los cálculos.

Es en tabu_ donde está propiamente implementada la técnica de la BT, la cual comienza asignando ceros a la matriz de frecuencia y la estructura de los movimientos tabú,

enseguida se dan valores iniciales al tiempo y se obtienen los ángulos iniciales, los cuales son obtenidos aleatoriamente; posteriormente se asigna al mejor resultado inicial (best), el mayor número entero que pueda almacenar la computadora en este caso para nuestra aplicación 3.4E300 y comenzamos un ciclo, el cual se repetirá mientras que él número de iteraciones sea menor que el máximo número de iteraciones, max_iter, establecido con anterioridad, parámetro de entrada.

En este momento el valor del mínimo de energía de cada procesador es inicializado con el máximo valor entero permitido por la computadora. Se realiza un intercambio aleatorio de posiciones de los ángulos del vector x, y tomamos el primer ángulo del nuevo vector (ángulo i), para utilizarlo en el comienzo del proceso de búsqueda.

Se asigna un número determinado de ángulos de prueba, de acuerdo al tipo de ángulo del vector x, *anticorrelacionados* o *no-anticorrelacionados* y se realiza una lotería, para decidir si se calcula la energía del *vecino* determinado por el movimiento del ángulo i. Se determina el procesador que efectuará las operaciones, se verifica que el nuevo ángulo sea diferente al anterior, y se asigna en la posición correspondiente al ángulo anterior en el vector x. Si la posición corresponde a la de un ángulo *anticorrelacionado*, se efectúan los ajustes pertinentes.

Posteriormente se calcula la energía, si ésta es menor que la *mínima energía* de ese procesador y el movimiento es *no-tabú*. Se almacena la energía obtenida, la posición del ángulo que fue cambiado, el valor del ángulo anterior, el valor del nuevo ángulo, y los cambios efectuados en los ángulos *anticorrelacionados*.

A continuación, se devuelven los valores originales de los ángulos *anticorrelacionados* y se efectúa otra lotería, repitiéndose el proceso anterior. Ahora tomamos el 20% de los vecinos obtenidos para efectuar los cálculos de energía. Se devuelve el ángulo i a su posición dentro del vector x, y se toma otro ángulo, para repetir el proceso, hasta que todos los ángulos sean analizados.

Los procesadores diferentes de cero envían la información de la estructura donde se almacenaron los valores correspondientes a la mínima energía, al procesador cero, el cual recibe y compara la información en conjunto con los valores por él obtenidos, para obtener los valores de la mínima energía de ese conjunto de vecinos, posteriormente envía copias de la información a cada uno de los demás procesadores.

La energía obtenida es menor que la mejor obtenida hasta entonces, se toma ésta como mejor y la iteración como la mejor iteración hasta entonces. Se incrementa el

contador del número de iteraciones y si no es mayor que el máximo número de iteraciones se repite todo el proceso.

Cuando llegamos al final del ciclo tomamos el tiempo final y se imprimen los resultados, en específico: la longitud de la lista tabú, el número de procesadores utilizados, los ángulos finales, la mínima energía obtenida y el tiempo empleado.

Capítulo 6

En el presente capítulo se presentan los resultados obtenidos y son comparados con los obtenidos por otros autores utilizando diferentes métodos de optimización.

Resultados

El algoritmo BT en paralelo fue usado para determinar la estructura de mínimo de energía para *met-encefálina*. Esta estructura tiene un total de 24 ángulos dihedros, incluidos los ángulos ω, con movimiento acorde a la relación 5.1. El procedimiento de optimización considera valores enteros para todas las variables tomadas desde un punto aleatorio en un intervalo de [−180°, +180°] para todos los ángulos dihedros. Excepto aquellos valores para los ángulos ω los cuales se mueven en el intervalo $-10° \leq \omega\text{-}180° \leq +10°$. Con estas restricciones se redujo drásticamente la dimensión del espacio conformacional, pero conservando diferencias básicas entre las conformaciones.

El algoritmo BT en paralelo compilado con la opción –O2 fue corrido empleando varios procesadores en la máquina SGI/CRAY ORIGIN 2000. El factor de aceleración (speedup) después de la paralelización del algoritmo es descrita en la Figura 6.2 donde se observa que los resultados obtenidos se encuentran cerca del caso ideal. En promedio una corrida completa tomó cerca de 400 iteraciones de BT, correspondiente a cerca de 1386 evaluaciones de energía por iteración. Cada simulación tomó cerca de 2060 segundos de tiempo total de CPU para un único procesador, calculadas en la ORIGIN 2000. El factor de eficacia mostrado en la Figura 6.2 muestra claramente la ventaja del diseño en paralelo de BT. Por ejemplo, cuando se utilizaron 16 procesadores, se obtuvo un factor de eficacia de

12.38; esto representa el uso de cerca de 166 segundos de CPU para obtener un péptido geométrico en la vecindad del mínimo global. Un aspecto atractivo del método en paralelo es el hecho de que sin importar el número de procesadores utilizados, el mínimo cercano al mínimo global de energía, fue encontrado alrededor de la iteración 350, esto en el sentido de que con un mejor diseño del experimento se puede reducir el esfuerzo computacional. En estos experimentos se obtuvieron en cerca del 60% de resultados, estructuras con menos de -9.5 Kcal/mole. La mayor parte de estas estructuras con pequeña energía produjeron otras estructuras cercanas al mínimo global de la Tabla 6.1 después de aplicarse una optimización local con SUMSL.

Los mejores parámetros definidos en la adaptación de la sección BT fueron encontrados experimentalmente y son: **y** = 0.2, **d** = 10°, **t** = 44, y **nimax** = 400. Es factible que para mejorar la eficacia del algoritmo se tiene que experimentar con diferentes valores en los parámetros de entrada de BT.

En la Tabla 6.1 se compara los resultados obtenidos, con los mejores mínimos de energía de conformaciones de *met-encefálina* obtenidos por otros autores usando diferentes métodos de optimización. Los primeros tres resultados muestran los ángulos dihedros de la estructura con mejor mínimo de energía para *met-encefálina* con los métodos conocidos como MCM, αBB, y CSA, respectivamente. El cuarto valor muestra los ángulos dihedros para la mejor conformación utilizando BT. El quinto valor muestra los ángulos dihedros obtenidos después de realizar una optimización local con SUMSL, usando como datos de entrada los obtenidos por BT. El resultado número seis fue obtenido en 1992 con el algoritmo MU. El resultado número siete lista los ángulos dihedros obtenidos por Wang y Pachter en 1997 empleando una variante de Recosido Simulado. Todos estos valores de energías se obtuvieron a partir de un solo punto de cálculo con la más reciente versión de ECEPP/3, esta acción normaliza todas las conformaciones a los mismos parámetros obteniendo un valor de energía conformacional común. Por ejemplo Nayeem reporta un valor de −12.396 Kcal/mol, igual que Androulakis y Lee, pero reportan un valor de energía de −11.77 Kcal/mol para este mínimo global de energía de la estructura. Estos valores son claramente una consecuencia derivada de ligeras diferencias en los parámetros de ECEPP usados en las citas anteriores.

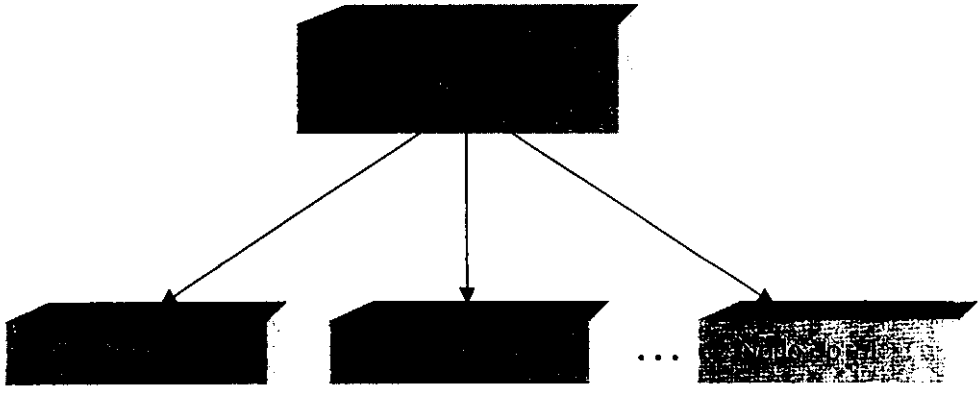


Fig. 6.1 Configuración de la red para el algoritmo en paralelo.

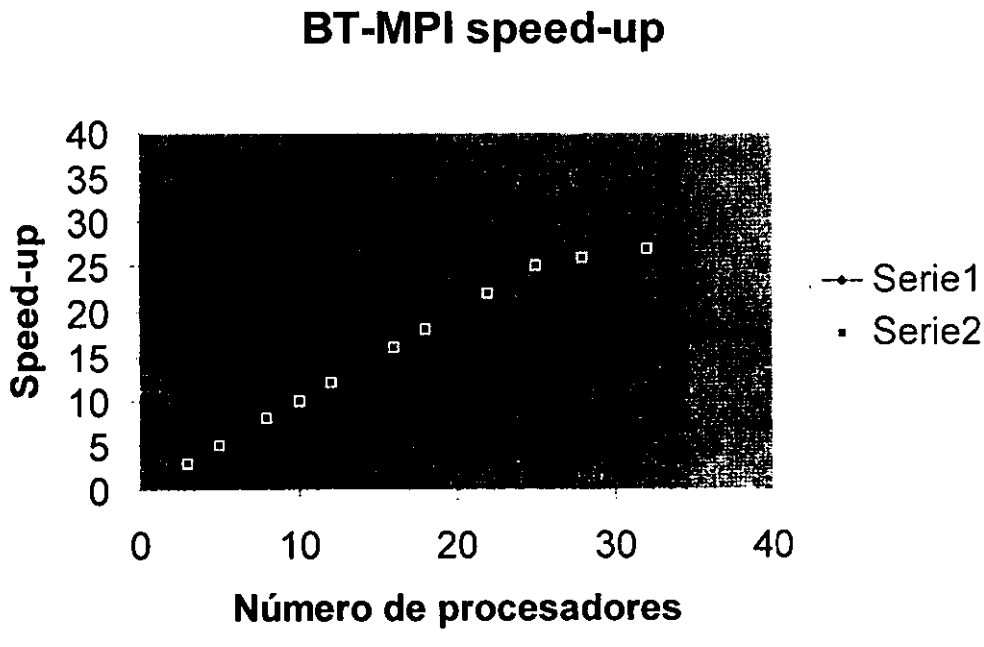


Fig. 6.2: Speedup

SERIE 1 Curvatura ideal para 25 procesadores
SERIE 2 Curvatura del trabajo realizado con 32 procesadores

Tabla 6.2
Angulos dihedros de 6 de las conformaciones con mejores mínimos de energía para Met-encefálina

		A0	A1†	A2†	BT	BT*	TA	ASA
Tyr	ϕ	-86.1	-83.5	-83.5	-80.0	-83.3	-94.0	-87.0
	Ψ	156.2	155.8	155.8	148.0	155.7	155.0	153.0
	ω	-176.9	-177.1	-177.2	-176.0	-177.1	-175.0	180.0
	χ_1	-172.6	-173.2	-173.2	-172.0	-173.2	-172.0	180.0
	χ_2	-101.3	-100.5	79.4	-98.0	-100.7	-104.0	-110.0
	χ_3	14.1	13.6	-166.4	23.0	13.7	27.0	145.0
Gly	ϕ	-154.5	-154.3	-154.3	-151.0	-154.2	-154.0	-161.0
	Ψ	83.7	86.0	86.0	86.0	85.8	74.0	72.0
	ω	168.6	168.5	168.5	173.0	168.5	170.0	180.0
Gly	ϕ	83.7	82.9	82.9	82.0	82.9	83.0	64.0
	Ψ	-73.9	-75.1	-72.5	-79.0	-75.0	-70.0	-94.0
	ω	-170.1	-169.9	-170.0	-171.0	-169.9	-170.0	-180.0
Phe	ϕ	-137.0	-136.9	-136.9	-137.0	-136.8	-136.0	-83.0
	Ψ	19.3	19.1	19.1	23.0	19.1	18.0	-26.0
	ω	-174.1	-174.1	-174.1	-176.0	-174.1	-174.0	180.0
	χ_1	58.8	58.8	58.9	58.0	58.8	59.0	72.0
	χ_2	-85.4	-85.5	94.6	95.0	94.5	-86.0	84.0
Met	ϕ	-163.6	-163.5	-163.5	-163.0	-163.4	-163.0	-79.0
	Ψ	160.5	160.9	161.2	172.0	160.8	166.0	133.0
	ω	-179.7	-179.8	-179.8	180.0	-179.8	178.0	180.0
	χ_1	52.8	52.9	52.9	51.0	52.8	52.0	-171.0
	χ_2	175.3	175.3	175.3	176.0	175.3	174.0	176.0
	χ_3	-179.8	-179.8	-179.8	177.0	-179.8	-175.0	180.0
	χ_4	61.4	61.4	-58.6	-60.0	-58.6	61.0	-60.0
Energía Kcal/mole		-12.389	-12.389	-12.389	-11.246	-12.389	-10.672	-10.643

A0.- A.Nayeem, J.Vila, y H.A Scheraga, J.Compt Chem., 12, 594(1991).
A1.- I.P.Androulakis, C. D. Maranas, y C. A. Floudas, J.Global Opt., 11, 1 (1997).
A2.- J. Lee, H. A. Scheraga, y S. Rackovsky, J. Compt. Chem., 18, 1222 (1997).
BT.- El trabajo presentado.
BT*.- Geometría optimizada desde BT con SUMSL[2].
TA.- L. B. Morales, R. Garduño Juárez, y D. Romero, J. Biomol. Struct. Dynamics, 9, 951 (1992).
ASA.- Z. Wang, y R. Pachter, J. Compt. Chem., 18, 323 (1997).

(†).- Los valores mostrados de energía son calculados con la versión más reciente de ECEPP/3. Originalmente se reporta una energía de -11.707 Kcal/mole debido a ligeras diferencias en los parámetros de ECEPP usados en la nota computacional.

Capítulo 7

Conclusiones

A pesar de que fue usada una aproximación entera para todos los ángulos dihedros, los valores correspondientes para el mejor mínimo de *met-encefálina* con búsqueda tabú son suficientemente cercanos a los reportados por otros autores. Aún más, se puede mostrar que la precisión de los valores de los ángulos dihedros son realmente buenos, desde que podemos tomar este mínimo "global" de BT y someterlo a una optimización local utilizando el algoritmo SUMSL; Los valores obtenidos son acordes a los de Nayeem y Androulakis que consecuentemente definen a sus conformaciones como las correspondientes al probable mínimo global para *met-encefálina*. El hecho de que diferentes procedimientos de optimización han obtenido las mismas energías conformacionales, da suficiente evidencia para decir que se obtuvo el mínimo global de energía para *met-encefálina*.

Además se pudo observar las bondades de la programación en paralelo, ya que los tiempos de CPU obtenidos mejoran a los publicados hasta este momento, como podemos observar en la Tabla 7.1.

Tabla 7.1 Resultados Computacionales Para Met-encefálina [13]

Método	N_{var}	CPU	Computadora
Minimización Monte Carlo	19	2-3 hrs.	IBM 3090
	24	10 hrs.	IBM 3090
Manejador Electrostatico Monte Carlo	19	2-3 hrs.	IBM 3090
	24	10 hrs.	IBM 3090
Ecuación de difusión	19	20 min.	IBM 3090
Self-Consistent Multitorsional Field	10	100 min.	IBM 3090
Recosido Simulado Multicanonical	19	6.0 hrs.	IBM RS600
Recosido Simulado	24	2.5 hrs.	Apollo DN1000
Umbrales Recosido Simulado	24	1.5 hrs.	Apollo DN1000
Recosido Simulado con Minimizador Montecarlo	24	2 hrs.	CRAY X-MP
Recosido Simulado con Minimizador Montecarlo	24	1.2 hrs.	CRAY-2S 4 procesadores
Minimizador Montecarlo vs Recosido Simulado	24	1.5-4 hrs.	IBM 30090
αBB	24	1.3 hrs.	HP-730
Búsqueda Tabú	24	34.3 min.	ORIGIN2000 Con 1 procesador
	24	1.19min.	ORIGIN2000 Con 32 procesadores

Como podemos observar los tiempos de BT de 1.19 minutos con 32 procesadores, mejoran los obtenidos hasta este momento, como son los de αBB de 1.3 horas para 24 ángulos.

BIBLIOGRAFIA

1. Arnold Neumaier, Molecular modeling of proteins and matematical predicción of protein structure, Pre-Printer Institut für Mathematik, Universität Wien, Austria.
2. E. Aarts y J. Korst, Simulated Annealing and Boltzmann Machines, John Wiley y Sons, Gran Bretaña, 1989.
3. G. Dueck y T. Scheuer, Journal Comput. Phys., 90, 161 (1990).
4. Z. Li y H. A. Scheraga, Journal Mol. Structure (Theochem), 179, 333 (1988).
5. M. Vásquez, E. Meirovitch y H. Meirovitch, Journal Comput. Chem., 98, 9380 (1994).
6. J. Lee, H. A. Scheraga, y S. Rackovsky, Journal Comput. Chem., 18, 1222 (1997).
7. U. H. E. Hansmann y Y. Okamoto, Journal Comput. Chem., 14, 1333 (1994).
8. J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis whith Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, MA, 1992.
9. K. D. Gibson y H. A. Scheraga, Journal Comput. Chem., 11, 468 (1990).
10. J. Kostrowwicki, L. Piela, B. J. Cherayil y H. A. Scheraga, Journal Phys. Chem., 95, 4113 (1991).
11. K. A. Olszewski, L. Piela, y H. A. Scheraga, Journal Phys. Chem., 96, 4672 (1992).
12. S. Vajda y C. DeLisi, Biololymers, 129, 1755 (1990).
13. I. P. Androulakis, C. D. Maranas y C. A. Floudas, Journal Global Opt., 11, 1 (1997).
14. L. B. Morales, R. Garduño-Juárez y D. Romero, Journal Biomol. Struct. Dynamics, 8, 721 (1991).
15. L. B. Morales, R. Garduño-Juárez y D. Romero, Journal Biomol. Struct. Dynamics, 9, 951 (1992).
16. F. Glover, ORSA Journal Comput., 1, 190 (1989).
17. G. Nemethy, K. D. Gibson, K. A. Palmer, C. N. Yoon, G. Payerlini, A. Zagari, S. Rumsey y H. A. Scheraga, Journal Phys. Chem., 96, 6472 (1992).
18. G. Nemethy, M. S. Pottle y H. A. Scheraga, Journal Phys. Chem. 87, 1883 (1983).
19. B. García y M. Toulouse, Comput. Oper. Res., 21 1025 (1994).
20. C. N. Fiechter, Discrete Appl. Math. , 51, 243 (1994).
21. E. Taillard, ORSA Journal Comput. 6, 108 (1994).
22. L. B. Morales, Math. Magazine, 70, 287 (1997).
23. V. Kvasnicka y J. Pospíchal, Journal Chem. Inf. Comput. Sci., 34, 1109 (1994).
24. M. Levitt, Journal Mol. Biol., 168, 621 (1983).
25. C. B. Post, C. M. Dobson y M. Karplus, Proteins, Struct. Func. Gen., 5, 337 (1989).
26. E. Taillard, Parallel Comput., 7, 443 (1991).
27. I. Foster, "Designing and Building Parallel Program", Addison-Wesley, Reading, Ma, 1995.
28. D. W. Walker, Parallel Comput., 20, 667 (1994).
29. A. Nayeem, J. Vila y H. A. Scheraga, Journal Comput. Chem., 12, 594 (1991).
30. D. M. Gay, ACM Trans. Math. Software, 9, 503 (1983).
31. Z. Wang, y R. Pachter, Journal Comput. Chem., 18, 323 (1997).
32. Garey M.R. y Johnson D.S., (1979) Computers and intractability. Freeman and Co. NY.

33. De los Cobos Silva, S., Tesis Doctoral, Facultad de Ingeniería, (1994),UNAM.
34. Hertz A. y de Werra D., (1987). Using tabu search techniques for grap coloring. Computing, 39.
35. Hertz A., (1991). Tabu search for large scale timetabling problems. European Journal of Operational Research, 54.
36. Smith P.H., (1992),(http://www.ccsf.caltech.edu/PSTP.html).
37. Morales L.B., Mathematics Magazine 70, 287 (1995)
38. Reevs C.R., .(1993). Modern heuristics techniques for combinatorial problems, Jhon Wiley & Sons, NY
39. Skorin - Kapov J., ORSA, Journal on Computing, 2:1 (1990)
40. Morales L.B., Investigación Operativa (1995)
41. S.Martins, C.C. Ribeiro y N. de la R. Rodriguez., Investigación Operativa, vol. 5 (1996)
42. R.M. Karp (1972). Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher (Eds.) Complexity of Computer Computations, Plenum Press, NY.