TESIS DE LICENCIATURA EN CIENCIAS DE LA COMPUTACION

Desarrollo de un algoritmo Tabu Search para el problema de Secuenciamiento de Tareas (SOP)

A. L. Fabiana Brazzalotto
Alicia Susana Carabajal

DEPARTAMENTO DE COMPUTACIÓN
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
UNIVERSIDAD DE BUENOS AIRES

Directora: Lic. Irene Loiseau

1999

AGRADECIMIENTOS

A María Teresa Ortuño y Laureano Escudero que contestaron nuestros E-mail y nos facilitaron los ejemplos para la comparación de resultados. A Darío Robak que respondió nuestras consultas aclarando numerosas dudas.
Nuestro agradecimiento además a Isabel Mendez y Marcelo Frías, y en forma muy especial a nuestra directora Irene Loiseau.
A Alfredo Olivero por su apoyo invalorable, y a Lucía Carabajal por su colaboración.

Alicia y Fabiana

A mis padres.
A Félix, y a mis hijos, Victoria y Leandro.
Por haberme acompañado, comprendido y apoyado en esta etapa.
Por su infinita paciencia y cariño.

Alicia

A mis padres. A mis amigos y compañeros de estudio, Mónica y Pablo, por todo lo que compartimos durante estos años juntos.
A mi amiga Graciela por su importante apoyo.
A Alfredo por todo.

Fabiana

A todos de corazón, Muchísimas gracias

INDICE

RESUMEN

Este trabajo consiste en desarrollar un algoritmo tipo Tabu Search para un problema de secuenciamiento de tareas (SOP-Sequential Ordering Problem) que se puede modelar como el problema de encontrar un camino hamiltoniano de costo mínimo en un grafo, respetando relaciones de precedencia. Se presentan para ello las características básicas del método Tabu Search, (que es un procedimiento de heurística general para problemas de optimización global),y el detalle de las características principales del algoritmo desarrollado. Se informan los resultados obtenidos y la comparación de los mismos, en problemas de literatura con los resultados presentados por un algoritmo exacto en [3]

ABSTRACT

The present work consists in developing a Tabu Search type algorithm, for a Sequential Ordering Problem (SOP), which can be modeled as a problem of finding a minimun cost hamiltonian path in a graph, with precedence constraints. We present here the basic characteristics of Tabu Search method . It is a general heuristic procedure for global optimization problems. Moreover, we describe the principal characteristics of the developed algorithm. They are presented, and compared on literature problems with the results appearing for an exact algorithm in [3] .

1.-INTRODUCCION

El objetivo de este trabajo fue la implementación de un algoritmo tipo *Tabu Search* para el problema de Secuenciamiento de Tareas con restricciones de Precedencia *(SOP, Sequential Ordering Problem),* y mostrar los resultados obtenidos comparándolos con aquellos que provienen de un algoritmo exacto tipo Branch & Cut, usados en [3] , para problemas de la vida real.

El *SOP* de acuerdo a lo que se menciona en [3], parece haber sido tratado por primera vez por Escudero. Su objetivo fue diseñar heurísticas para ser implementadas en un sistema de planificación de producción que dieron buen resultado en la práctica con respecto a la garantía de calidad de la solución y al tiempo de procesamiento.

Existen varios problemas de la vida real que pueden ser modelados con un SOP. Por ejemplo, aplicaciones de ruteo donde se deben realizar retiros y entregas en un determinado orden. Otro caso es un ruteo on-line de una grúa apiladora en un sistema automático de almacenamiento, o en aplicaciones de encolamiento donde un cierto trabajo tiene que estar terminado antes que otros comiencen.

Sir William Hamilton sugirió la clase de grafos que lleva su nombre cuando le fue encomendado que construyera un ciclo conteniendo cada uno de los vértices de un dodecaedro. Si G tiene un ciclo Z que pasa por todos los nodos solo una vez, entonces G es llamado un grafo hamiltoniano y Z es llamado un ciclo hamiltoniano. No existe ninguna caracterización elegante de los grafos hamiltonianos, aunque sí se conocen numerosas condiciones necesarias y suficientes [9].

El problema de encontrar un camino hamiltoniano factible con mínimo costo total, se conoce como problema de ordenamiento secuencial *(SOP)* por lo

tanto el SOP puede ser visto como una generalización del problema del camino de Hamilton, donde llamaremos factible a un camino Hamiltoniano que satisface ciertas relaciones de precedencia dadas.

Según [1], el problema SOP se puede tratar de la siguiente forma: dado un grafo dirigido , un arco representa la posibilidad de llevar a cabo dos tareas en forma secuencial y el peso de la arista, representa el costo de ejecutar las tareas en ese orden. Además, las relaciones de precedencia dadas especifican si algunas tareas deben ejecutarse antes que otras.

El problema SOP es *NP-completo* .Reduciendo en el problema *SOP* los nodos *1* y *n* a un nodo único obtenemos un *ATSP (Asymmetric Travelling Salesman Problem)* con restricciones de precedencia sobre *n-1* nodos. Más aún, el *SOP* se reduce a un problema *AHPP (Asymmetric Hamiltonian Path Problem)* en caso que el digrafo de precedencia tenga un conjunto de arcos vacío. Entonces el *AHPP* es un caso especial del *SOP*. Como el *AHPP* es un problema *NP-completo* lo mismo pasa para el *SOP*.

Con este objetivo el trabajo incluyó:

- Búsqueda de una solución inicial usando una heurística golosa.
- Desarrollo de una heurística tipo Tabu Search ad-hoc

A partir de la solución inicial factible hallada, se aplica la técnica *Tabu Search* para obtener la solución óptima.

La organización del trabajo es la siguiente: en el capítulo 2 se explica el uso de heurísticas en este tipo de problemas y se dan ejemplos de heurísticas constructivas, usadas para el problema del viajante de comercio. En el capítulo 3 se presenta al SOP, como un problema de programación lineal 0/1, y en el 4 se detallan las características generales de la técnica Tabu Search como uso de la memoria, Lista Tabu, criterio de aspiración, etc. Luego, en el capítulo 5 se describe el algoritmo Tabu Search propuesto y detalles de la implementación realizada, así como las características generales del algoritmo, y, por último se presentan los resultados obtenidos comparándolos con los logrados con un algoritmo Branch & Cut. Al final del trabajo se dan las conclusiones obtenidas con la experiencia y se detalla la bibliografía utilizada. Los detalles sobre las ejecuciones con los distintos ejemplos , se encuentran en el Apéndice.

2.- MODELO MATEMÁTICO

De acuerdo a lo propuesto en [3], vamos a modelizar el problema *SOP* utilizando dos grafos, uno de ellos completo, y el otro adicional que indica las precedencias entre los nodos. Cada nodo representa una tarea en la secuencia y cada arista, el costo de ir de una tarea a otra.

Dado un grafo dirigido $D_n=\{V,A_n\}$ sobre n nodos, coeficiente de costos c_{ij} pertenecientes a N, con $c_{ij} > 0$, asociado con cada arco (i,j) que pertenece a A_n y digrafo adicional de precedencia $P=(V,R)$, que está definido en el mismo conjunto de nodos V como D_n. Un arco (i,j) que pertenece a R representa una relación de precedencia, es decir i tiene que preceder a j en todo camino hamiltoniano factible. Obviamente, el digrafo de precedencia P debe ser acíclico (es decir, no puede contener ningún ciclo dirigido). Mas aún si $(i,j), (j,k) \in R$ entonces k no se puede llevar a cabo antes que i, en otras palabras, podemos asumir que P es transitivamente cerrado.

Vamos a presentar la formulación del SOP, como problema de Programación Lineal 0/1 .

Según se indica en [3] y [1], supongamos un digrafo completo $D=(V,A_n)$, y un digrafo de precedencia $P=(V,R)$. Para cada arco (i,j) que pertenece a A_n introducimos una variable binaria x_{ij} tal que:
x_{ij} = 1 si (i,j) pertenece a A_n en un camino hamiltoniano
x_{ij} = 0 en caso contrario.

Inmediatamente se pueden hacer deducciones simples.

Dado $D_n = (V, A_n)$ y $P=(V,R)$ entonces:
1) x_{ji} = 0 para todo (i,j) que pertenece a R
2) $X_{ik}=0 \; \forall \; (i,k) \in R$ tal que $(i,j) \in R$ y $(j,k) \in R$

3) *Sea $W \subset V$, entonces $A(W) = \{(i,j) \in A_n / i,j \in W\}$.*

4) *Si $j \in V$ entonces $\delta^+(j) = \{(j,k) \in A_n\}$ y $\delta^-(j) = \{(i,j) \in A_n\}$,*

5) *Si $W \subseteq V \setminus \{j\}$ entonces $(j:W) = \{(j,k) \in A_n / k \in W\}$ y $(W:j) = \{(i,j) \in A_n / i \in W\}$*

Formulación como Problema de Programación Lineal 0/1

$x^* = Min\ c'x$

s.a

(1) $x(\delta^-(i)) = 1\ \forall\ i \in V \setminus \{1\}$

(2) $x(\delta^+(i)) = 1\ \forall\ i \in V \setminus \{n\}$

(3) Dado un conjunto $A \subseteq A_n$, $x(A)$ denota $\sum_{(i,j)} x_{ij}$.
$x(A(W)) \leq |W| - 1\ \forall\ W \subset V,\ 2 \leq |W| \leq n-1$

(4) $x(j:W) + x(A(W)) + x(W:i) <= |W|\ \forall\ (i,j) \in R$ y $\forall\ W \subseteq V \setminus \{i,j\},\ W \neq 0$

(5) $x_{ij} \in \{0,1\}\ \forall\ (i,j) \in A_n$

(1) a (3) y (5) es la formulación del problema de determinar un camino hamiltoniano mínimo en $D=(V,A_n)$.

(4) es la desigualdad que garantiza que se cumplan las restricciones de precedencia.

(1) Indica que en el camino hamiltoniano, llega un solo arco al nodo i para todo i en V

(2) Indica que en el camino hamiltoniano, sale un solo arco del nodo i para todo i en V

(1) y (2) Aseguran que por cada nodo se pasa una y solo una vez

(3) Es la restricción llamada 'eliminación de subtours', ya que establece que para todo subconjunto de k nodos unidos por aristas, solo formarán parte del camino Hamiltoniano, a lo sumo k-1 aristas, para así evitar la formación de ciclos.

(4) Establece que si i debe preceder a j, no hay directa ni transitivamente un

arco que llegue desde j hacia i

(5) Establece que x ij es una variable binaria

N. Ascheuer en [3] usa esta formulación para el SOP para luego aplicar un algoritmo Branch & Cut. Nuestros resultados serán comparados con los obtenidos por ellos.

3.- HEURÍSTICAS

3.1.- Por qué usar heurísticas?

En todos los ejemplos de problemas de optimización combinatoria conocidos (como diseño de redes de telecomunicación eficientes, ruteo de vehículos de entrega, etc), es teóricamente posible enumerar todas las combinaciones de soluciones y evaluar cada una con respecto a la función objetivo. Las que proveen el valor más favorable, se llaman "óptimas". Sin embargo, desde una visión práctica, seguir cada estrategia de enumeración completa se torna muy laborioso, ya que el número de combinaciones crece en forma exponencial con el tamaño del problema.

Se ha hecho mucho trabajo en los últimos 40 años para desarrollar métodos de búsqueda de óptimos que no requieren explícitamente examinar cada alternativa. Estas investigaciones tuvieron éxito en el campo de optimización combinatoria, y los métodos de búsqueda desarrollados son capaces de resolver problemas de la vida real, como versiones del problema del viajante de comercio, etc. A pesar de eso, gran parte de los problemas encontrados en la industria son computacionalmente intratables por este método, o son muy grandes para ser manejados por algoritmos exactos. En estos casos, los métodos heurísticos se emplean normalmente para encontrar soluciones buenas, pero no necesariamente óptimas. Se han desarrollado varias técnicas generales para construir heurísticas, que demostraron ser efectivas para obtener buenas soluciones en problemas difíciles de optimización combinatoria. Las más conocidas son *Simulated Annealing*, *Tabu Search*, *Algoritmos Genéticos* y *GRASP* (Greedy Randomized Adaptive Search Procedures).

En el problema que nos ocupa, "minimizar el costo de ejecución de una secuencia de trabajos donde existen relaciones de precedencia", para buscar una solución exacta una de las posibilidades es analizar todas las alternativas

posibles sobre el orden en que se deberían ejecutar las tareas, algo que es prácticamente imposible para secuencias de trabajos numerosas. En el caso del SOP, que como ya se mencionó es NP-completo no se conocen algoritmos buenos para resolverlos, es decir, algoritmos que resuelvan el problema en tiempo polinomial, por lo tanto se usarán heurísticas para lograr buenas soluciones en un tiempo razonable.

3.2.- Heurísticas constructivas

Son las técnicas heurísticas capaces de construir una solución inicial a partir de los datos de entrada.

Tradicionalmente se han usado heurísticas constructivas para el problema del viajante de comercio. Las más conocidas de estas heurístivas se pueden categorizar como heurísticas miopes o golosas.

Este tipo de heurísticas trabaja viendo al problema representado por un grafo, cuyas aristas tienen asociado un costo que puede representar ir de una ciudad a otra (viajante de comercio) o ejecutar un trabajo antes que otro (secuenciamiento de trabajos). Para ello arma una lista ordenada de los nodos con sus costos asociados, de menor a mayor.

Para citar algunos ejemplos de heurísticas comúnmente usadas para el problema del Viajante de Comercio, vemos las siguientes [4]:

a) Vecino más próximo

Esta versión de algoritmo miope se debe a Belmore y Nemhauser (1988). Consiste simplemente en partir del vértice inicial. Comenzamos desde x_1 perteneciente a N, y elegimos en cada iteración un vértice, eliminándolo de un conjunto de posibles.

Algoritmo *VP(G=(N,M))* [camino en *G*] (Según [4])

Input {*G=(N,M)* con *N={1,2,.....,n}* y c_{ij} para cada $i,j \in N$}

$H = \{1\}$

$x_0 = 0$

$N = N \setminus \{1\}$

$i = 1$

Mientras $N \neq 0$ hacer

Inicio

Sea {c_{ik} el costo de a_{ik}, i,k perteneciente a N y a_{ik} perteneciente a M}

Elegir *j* perteneciente a *N* tal que c_{ij} = mínimo {c_{ik}}

$H = H \cup \{j\}$

$x_0 = x_0 + c_{ij}$

$N = N \setminus \{j\}$

$i = j$

Fin

Output {$H = H \cup \{1\}$, $x_0 = x_o + c_{ij}$}

Fin

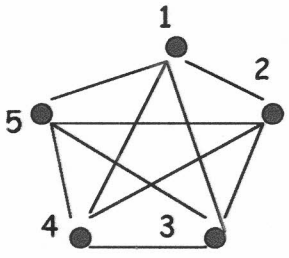
Este algoritmo es de orden O (n^2). Un ciclo Hamiltoniano obtenido con este algoritmo sufre una fuerte influencia de la elección del vértice inicial. Para eliminar esta influencia basta repetir el procedimiento *n* veces tomando cada vez un vértice diferente. El algoritmo resultante en este caso será del orden de O (n^3).

Con un ejemplo:

Dada la matriz de costos *D* =

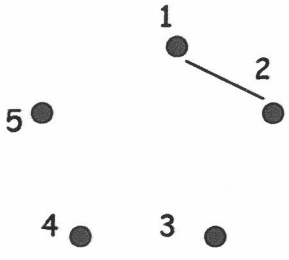
-	1	2	7	5
1	-	3	4	3
2	4	-	5	2
7	4	5	-	3
5	3	2	3	-

Paso 1:



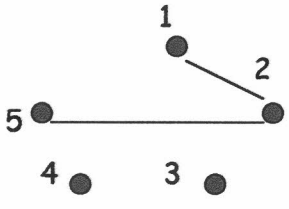
$x_0 = 0$, $H=\{1\}$

Paso 2



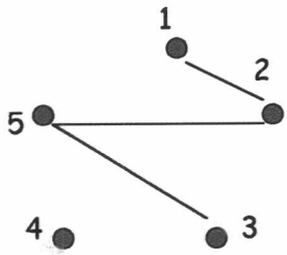
$x_0 = 1$, $H=\{1,2\}$

Paso 3



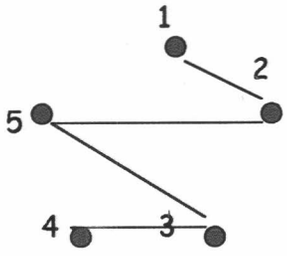
$x_0 = 4$, $H=\{1,2,5\}$

Paso 4



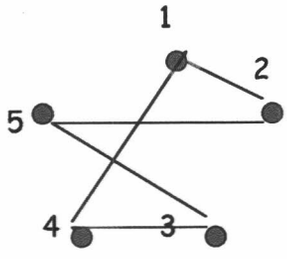
$x_0 = 6$, $H=\{1,2,5,3\}$

Paso 5



$x_o = 11$, $H=\{1,2,5,3,4\}$

Paso 6



$x_o = 18$, $H=\{1,2,5,3,4,1\}$

b) Inserción más próxima

Esta heurística consiste en:

a) Partir de un ciclo $\{i_1,i_2,i_3\}$, de dimensión 3
b) Encontrar un vértice k que no pertenezca al ciclo, tal que el costo de la arista que los une sea menor que para cualquier otro vértice del ciclo.
c) Encontrar una arista, digamos (i,i+1), del ciclo tal que $c_{ik} + c_{ki+1} - c_{ii+1}$ sea mínimo.
d) Insertar el vértice k entre i e i+1. Si el ciclo es Hamiltoniano, PARAR. En caso contrario, repetir el procedimiento desde b)

c) Inserción más distante

Es similar al anterior. Solo difiere en la elección del vértice que deberá ser el más distante que cualquier otro.

d) Inserción más barata

Es similar a b), pero la elección del vértice *k* debe ser no perteneciente al ciclo actual y además se debe elegir una arista *(i,i+1)* del ciclo tal que $c_{ik}+c_{ki+1}- c_{ii+1}$ sea mínimo.

e) Inserción por mayor ángulo

Se construye un ciclo convexo uniendo todos los nodos exteriores de un grafo, y se elige un vértice *k* no perteneciente al ciclo y la arista *(i,j)* del ciclo, tal que el ángulo formado por las aristas *(i,k)* y *(k,j)* sea máximo. Se inserta el vértice *k* en el ciclo entre *i* y *j*.

f) Método de las economías

Este método fue propuesto originalmente por Clarke y Wright (1964)(Ver [4]), para el ruteo de vehículos, y posteriormente fue utilizado para el problema del viajante de comercio (Golden, Bodin, Doyle y Stewart, 1980) (Ver[4]).

El procedimiento comienza por la selección arbitraria de un vértice inicial del grafo completo y construye ciclos de tamaño 2 desde este vértice a cada uno de los n-1 restantes. En cada iteración, dos ciclos son combinados eliminando dos aristas (desde y hasta el vértice base) y agregando una arista de conexión. Los ciclos o "subcaminos" son elegidos para el proceso de combinación de manera de maximizar la distancia economizada en relación a la lista ordenada de forma monótona decreciente. Se repite el procedimiento hasta obtener un ciclo Hamiltoniano. Este algoritmo tiene una complejidad $O(n^2 * \log_2 n)$. Si con el fin de eliminar la influencia en la elección del vértice base, repetimos el procedimiento para cada uno de los *n-1* vertices restantes como punto de referencia, el algoritmo pasaría a ser de orden $O(n^3 * \log_2 n)$.

En nuestro caso, para hallar la solución inicial, se adaptó la heurística del "vecino más próximo", presentada en [4], enriquecida por la elección del mejor vértice y validación de las precedencias. Esta heurística, busca dentro de todos los nodos posibles para moverse, a aquel que tiene menor costo. Va armando un camino hamiltoniano que cumpla con todas las restricciones de precedencias establecidas. De esta forma se garantiza que la solución hallada por este método es factible.

En este trabajo nos limitamos a la implementación de la heurística del "vecino más próximo" quedando para futuras investigaciones probar con otras heurísticas golosas para poder evaluar los resultados y compararlos con los obtenidos en el presente trabajo.

4.- QUÉ ES EL TABU SEARCH

Tabu Search es un procedimiento de heurística general para optimización global. Como otras técnicas heurísticas, *Tabu Search* está basado en ideas simples que pueden ser aplicadas eficientemente para conseguir soluciones casi óptimas para muchos tipos de problemas difíciles. En particular, un algoritmo de tipo *Tabu Search* permite una amplia exploración del espacio de soluciones evitando mediante la confección de "Listas Tabu", que veremos más adelante, quedar "atascado" en óptimos locales.

Durante las últimas décadas muchos esfuerzos se han dedicado a la solución de grandes problemas de optimización combinatorios. Muchos de los algoritmos son diseñados para tipos de problemas específicos, y no se adaptan con mínimas modificaciones a otros tipos de problemas de optimización.

Tabu Search [12] alcanzó gran éxito en problemas de optimización práctica, y ha logrado muy buenos resultados en problemas clásicos como el problema del viajante de comercio o el coloreo de grafos. Las aplicaciones están creciendo día a día en áreas como administración de recursos, diseño de procesos, logística, telecomunicaciones y optimización general combinatoria. La forma actual de *Tabu Search* la propuso en 1986 Fred Glover[12]. Está basado en métodos diseñados para cruzar los límites de factibilidad u optimalidad local, que son tratados normalmente como barreras. Ideas similares al *Tabu Search* fueron desarrolladas en forma independiente por Pierre Hansen, que lo llamó *"método steepest ascent/ mildest descent"[12]*

Junto con *Simulated Annealing* y *Genetic Methods*, *Tabu Search* fue definido por el Comité de la Próxima Década de Investigación Operativa (CONDOR 1988) como "extremadamente prometedor" para el futuro tratamiento de operaciones prácticas. La diversidad de aplicaciones exitosas implementadas de *Tabu Search* indican que el futuro es ahora.

4.1.- En qué consiste la técnica

Tabu Search opera sobre una función *f(X)* a ser optimizada sobre un conjunto *X*, donde *f(X)* puede ser lineal o no lineal y el conjunto *X* es el conjunto de restricciones sobre el vector *x*. Estas restricciones pueden incluir desigualdades lineales o no lineales. *Tabu Search* comienza de la misma manera que una búsqueda local o de vecindario, procediendo iterativamente desde un punto (solución) a otro hasta que se satisface un criterio de terminación. Cada *x* perteneciente a *X* tiene un vecindario asociado *N(x)* incluído en *X* y cada solución *x´* perteneciente a *N(x)* se alcanza desde *x* por una operación llamada *movimiento*. Normalmente en *Tabu Search*, los vecindarios se asumen como simétricos, *x´* es vecino de *x* sii *x* es vecino de *x´*.

Puede haber *movimientos de inserción* o *movimientos de permutación*. En el caso de grafos, el *movimiento de inserción* , por ejemplo, simplemente inserta un nodo en el grafo entre dos existentes unidos por una arista, y el *movimiento de permutación* intercambia las posiciones de dos nodos en el grafo. Hay algunas implementaciones que utilizan una estrategia combinada con ambos tipos de *movimientos*. Algunas experiencias demuestran que es ventajoso usar más de una técnica para moverse de una solución factible a otra dentro del método.

Uso de la memoria

Con el uso de la memoria, una mala elección hecha estratégicamente en un punto del proceso, puede convertirse en una buena decisión en un punto más adelante del mismo proceso de búsqueda.

Esta característica se llama "exploración interesada", y contiene los principios básicos de una búsqueda inteligente (explota rasgos característicos de las soluciones históricamente buenas y además explora nuevas regiones que

parecen prometedoras).

Otra de las características de *Tabu Search* es la memoria flexible, que permite una gran flexibilidad en la búsqueda. Este tipo de memoria sirve para usar la información recolectada (entre otras cosas, calidad e influencia de cada elección), durante la búsqueda para hacer elecciones "inteligentes" de las soluciones, porque identifica elementos comunes a soluciones "buenas", también se usa para "penalizar" movimientos que podrían provocar que la búsqueda se quedara en óptimos locales. Para ello, usa los elementos de la "Lista Tabu".

Este tipo de memoria también contrasta con los diseños de memoria rígida típica de estrategias *"branch-and-bound"*

La memoria flexible de *Tabu Search* permite, además la implementación de procedimientos, que son capaces de buscar en una gran parte del espacio de solución con mínimo requerimiento de memoria.

Las estructuras de memoria en *Tabu Search*, tienen cuatro características principales: cercanía, frecuencia, calidad e influencia.

- Cercanía y frecuencia se complementan entre si. El más común de los usos de esta memoria es mantener pistas de atributos de soluciones que fueron cambiadas en un pasado reciente. Para ello se arman las "Listas Tabu".

- Calidad es la capacidad de diferenciar el mérito de una solución encontrada durante la búsqueda. Así la memoria se puede usar para identificar elementos que son comunes a soluciones buenas y para moverse de una solución a otra. Esta forma de memoria se basa en incentivar acciones que guíen a soluciones buenas y penalicen a otras acciones que lleven a soluciones peores.

- Influencia considera el impacto de los cambios producidos por las distintas soluciones.

Tipos de memoria

Una importante distinción en *Tabu Search* es la diferencia entre memoria a corto plazo y memoria a largo plazo. El más común de los usos de la memoria a corto plazo es mantener pistas de atributos de soluciones que fueron cambiados en un pasado reciente. Para explotar esta memoria, los atributos seleccionados se llaman *Tabu*, y las soluciones que contienen elementos *Tabu*, se transforman en Tabu. La duración que un atributo permanece en la Lista Tabu , (medida en número de iteraciones) se llama "permanencia en la Lista Tabu". El espacio de memoria necesario depende de los atributos y del tamaño del vecindario, pero no depende del tiempo de permanencia de un atributo o solución en la Lista Tabu.

En algunas aplicaciones, las componentes de la memoria *Tabu Search* a corto plazo son suficientes para producir soluciones de muy alta calidad. Sin embargo, en general, *Tabu Search* se vuelve más fuerte si se incluye la memoria a largo plazo y sus estrategias asociadas. En estas estrategias, el vecindario modificado puede contener soluciones que no están en el vecindario original, y que generalmente consisten de soluciones elite (óptimo local de alta calidad) encontradas en varios puntos del proceso de solución. La experiencia ha demostrado que, contrariamente a lo que se podría suponer, el uso de esta memoria no requiere largas ejecuciones antes de ver sus beneficios. En general, sus mejoras se manifiestan en un tiempo razonable, y de ser así, pueden permitir que las soluciones sean terminadas algo antes de lo que sería posible de otra forma, debido a encontrar soluciones de alta calidad en un espacio de tiempo económico.

Lista Tabu

La Lista Tabu está compuesta por movimientos "prohibidos", atributos de soluciones y soluciones encontradas en las *n* últimas iteraciones. A partir de esta lista, se restringe el vecindario de soluciones hallado, *N(x)*, con *x* la solución actual, limitándolo a aquellas soluciones que no estén incluídas ni explícita ni implícitamente en la lista, obteniendo *N*(x)*. Luego, de *N*(x)*, se elige la mejor solución, teniendo en cuenta un determinado criterio, que puede ser la mejora en el valor de la función objetivo, se realiza una operación llamada *movimiento* que va desde la solución actual a la encontrada, actualizando la "Lista Tabu". La longitud de la Lista Tabu y los operadores para manejar la lista juegan un importante rol en la performance.

Tiempo de permanencia en la Lista Tabu

Una forma de identificar un efectivo *tiempo de permanencia en la Lista Tabu* y la selección de reglas de activación Tabu, para una clase de problemas determinada; es observar la aparición de ciclos cuando el tiempo de permanencia es muy corto, y el deterioro en la calidad de la solución cuando ese tiempo es muy largo. El deterioro lo causa normalmente el prohibir demasiados movimientos. El mejor valor para determinar ese tiempo de permanencia queda en un rango intermedio entre los dos extremos. En general, los valores pequeños para ese parámetro permiten a la exploración de soluciones ''acercarse'' al óptimo local, mientras que los largos son preferibles cuando es importante romper con la vecindad del óptimo local.

Hay dos formas dinámicas usadas habitualmente para el tiempo de permanencia en la Lista Tabu. Ambas usan un rango definido por parámetros *tmín* y *tmax*. El parámetro *t* se selecciona aleatoriamente dentro del rango, siguiendo una distribución uniforme. En la primera forma, el tiempo se mantiene constante por una cantidad determinada de iteraciones (que llamaremos α), al término de

las cuales, se selecciona un nuevo tiempo de permanencia usando el mismo mecanismo. La segunda forma, calcula una nueva *t* en cada iteración .

Criterio de aspiración

A pesar de que una solución del vecindario esté en la Lista Tabu, puede ser seleccionada, si cumple con el llamado "criterio de aspiración".

Este criterio se usa para determinar cuando una regla de activación- Tabu puede ser pasada por encima, teniendo en cuenta la calidad de la solución a examinar, es decir la mejora que se logra en el valor de la función objetivo y su comparación con soluciones históricamente buenas. El uso correcto de este criterio puede ser muy importante para lograr un buen nivel de performance en el método. Existen muchas clases de "criterios de aspiración", desde el más sencillo que es remover una clasificación Tabu porque el movimiento produce un mejor valor de la función objetivo, hasta criterios que prueban la influencia de la movida, es decir el grado de cambio que induce en la solución.

En *Tabu Search* se pueden hacer aspiraciones por *movimiento* y aspiraciones por atributo, depende de cómo esté constituída la Lista Tabu. Una aspiración por *movimiento*, cuando se satisface revoca la clasificación Tabu del *movimiento*. Una aspiración por atributo, cuando se satisface, revoca el *Estado Tabu* del atributo.

Ejemplos de criterios de aspiración

Aspiración por	Descripción
Default	Si todos los *movimientos* disponibles son clasificados *Tabu*, y no están admitidos por otro criterio de aspiración, entonces se elige una movida para la cual el número de iteraciones que restan sea mínimo.
Objetivo	Global: Una aspiración por *movimiento* se satisface si el *movimiento* produce una solución mejor que la mejor obtenida. Regional: Una aspiración por *movimiento* se satisface si el *movimiento* produce una solución mejor que la mejor encontrada en la región donde está la solución.
Dirección de búsqueda	Un atributo puede agregarse o borrarse de una solución, si la dirección de la búsqueda (mejorando o no) cambió.
Influencia	El *Estado Tabu* de una movida de baja influencia se revoca si se puede hacer un *movimiento* de alta influencia estableciendo el estado para la de baja influencia.

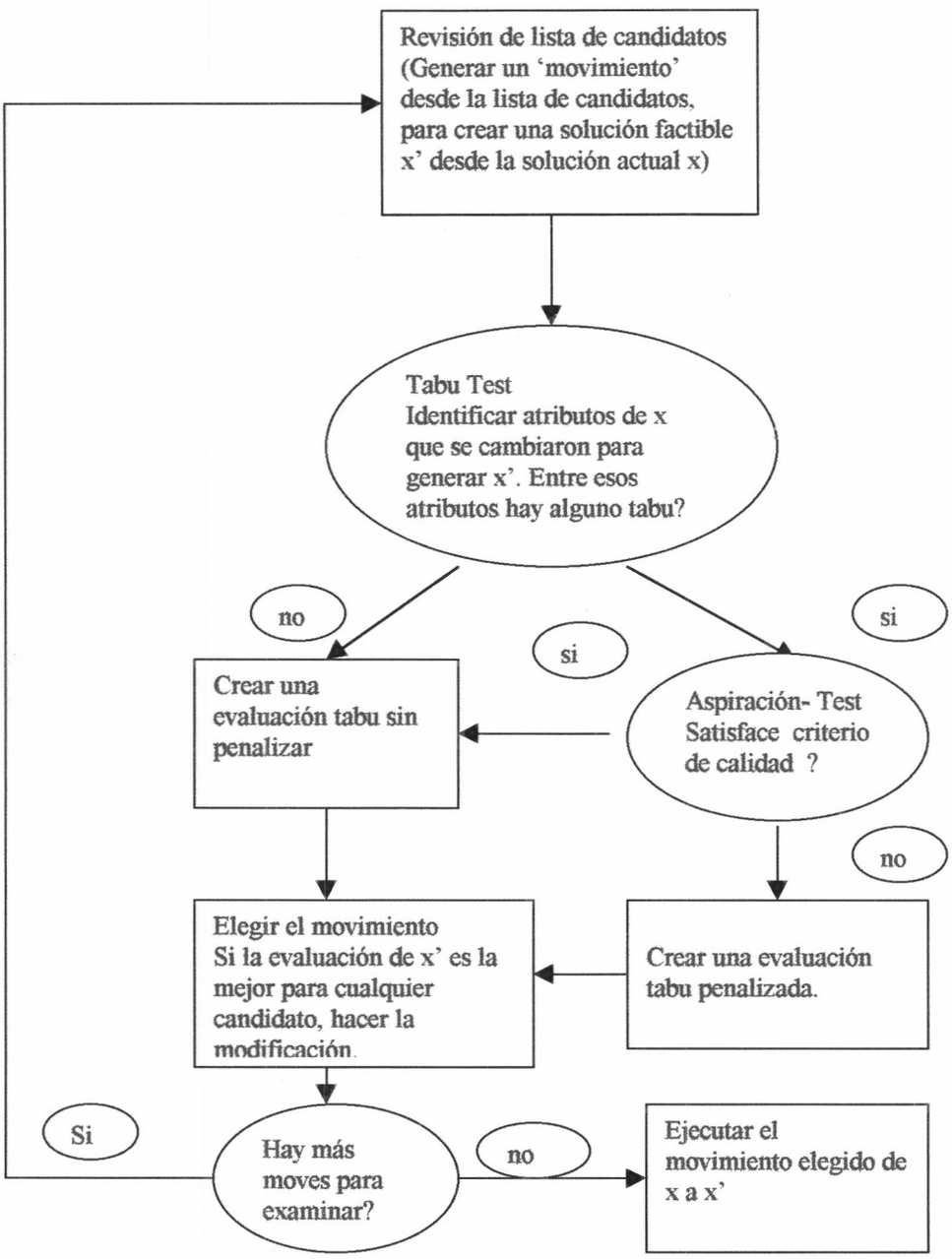
Intensificación y diversificación

Otra característica muy importante del *Tabu Search* son las estrategias de intensificación y diversificación. Estas estrategias están relacionadas con el uso de memoria a largo plazo. Las estrategias de intensificación están basadas

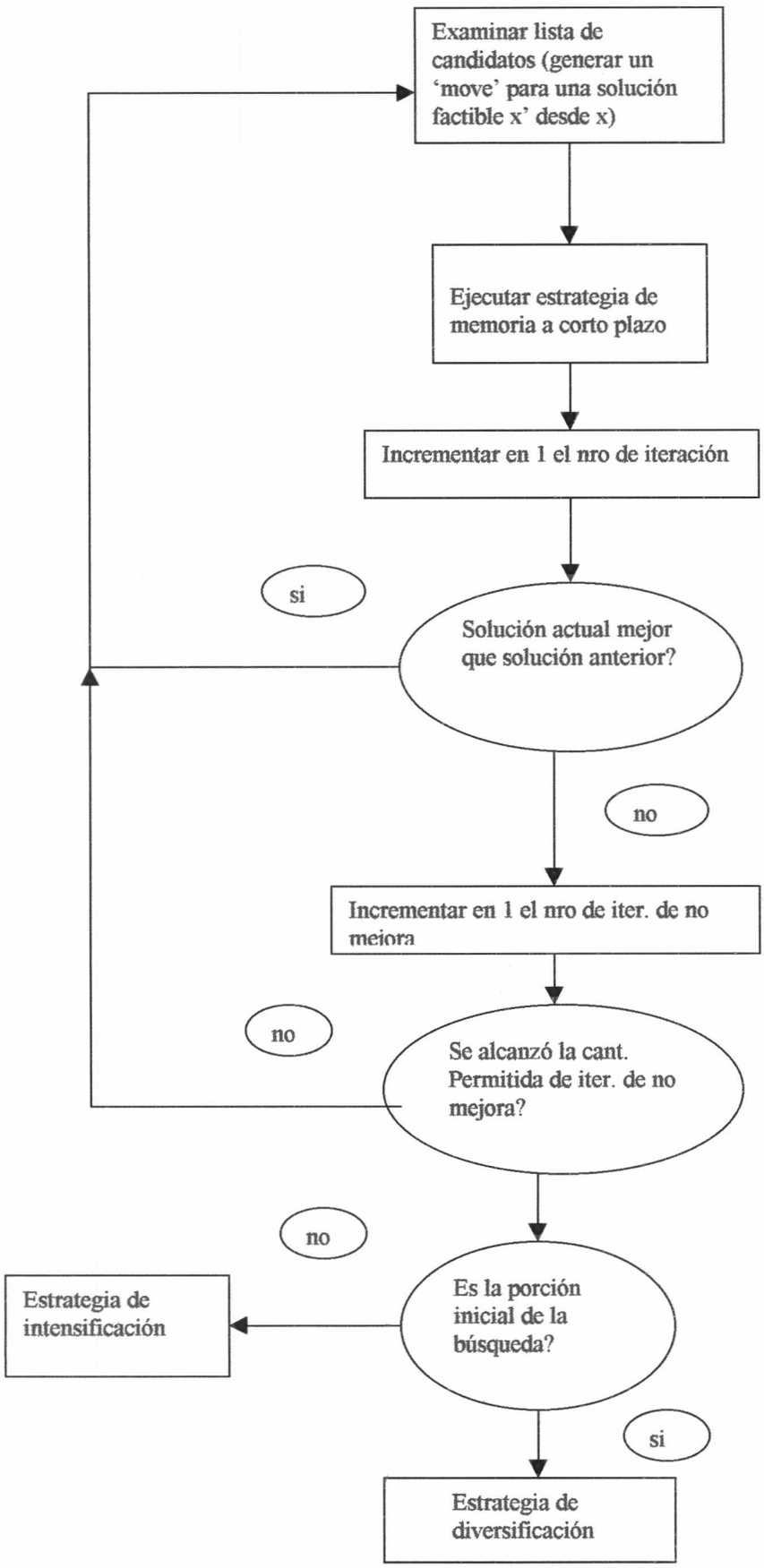
en modificar las reglas de selección para encarar combinaciones de movimientos y rasgos de solución que históricamente se encontraron buenos. La gran diferencia entre las etapas de diversificación e intensificación, es que en la etapa de intensificación, la búsqueda "da vueltas" para examinar el vecindario de soluciones elite (las mejores soluciones encontradas durante la búsqueda) , porque se piensa que ahí puede haber soluciones adicionales de buena calidad, mientras que en la etapa de diversificación encara la búsqueda del proceso para moverse y examinar regiones no visitadas.

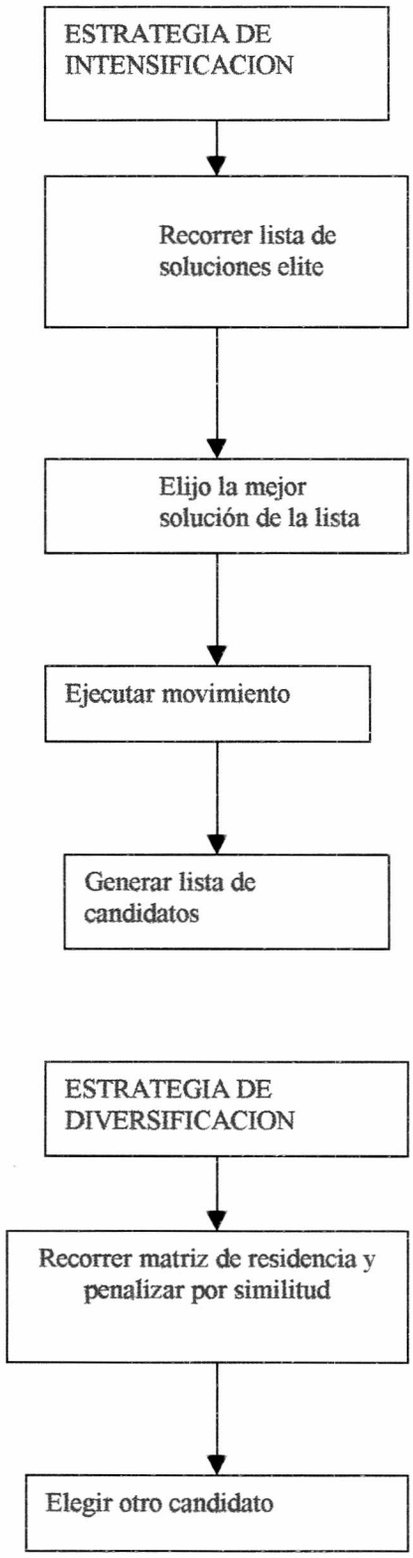
Normalmente, *Tabu Search* trabaja de la siguiente forma: en su forma más simple usa una estrategia de memoria a corto plazo, para guiar inteligentemente la búsqueda lejos de los vecindarios recientemente visitados. En su versión más completa, usa la estrategia de memoria a largo plazo para permitir intensificación y diversificación, teniendo en cuenta un conjunto de soluciones buenas conocidas, y las características de cada una.

Un diagrama del algoritmo usando memoria a corto plazo sería el siguiente



La memoria a largo plazo y sus estrategias asociadas trabajan de la siguiente forma:


Examinar lista de candidatos (generar un 'move' para una solución factible x' desde x)
Ejecutar estrategia de memoria a corto plazo
Incrementar en 1 el nro de iteración
si
Solución actual mejor que solución anterior?
no
Incrementar en 1 el nro de iter. de no mejora
no
Se alcanzó la cant. Permitida de iter. de no mejora?
no
Es la porción inicial de la búsqueda?
Estrategia de intensificación
si
Estrategia de diversificación

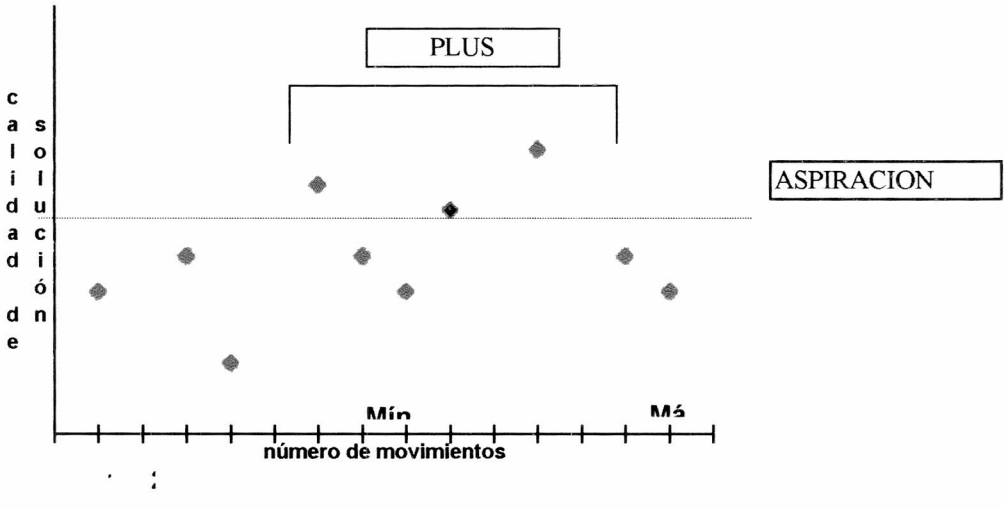

ESTRATEGIA DE INTENSIFICACION
Recorrer lista de soluciones elite
Elijo la mejor solución de la lista
Ejecutar movimiento
Generar lista de candidatos
ESTRATEGIA DE DIVERSIFICACION
Recorrer matriz de residencia y penalizar por similitud
Elegir otro candidato

Lista de candidatos

Estas estrategias se usan para restringir el número de soluciones examinadas en cada iteración. Una buena elección en la estrategia de generación de lista de candidatos mejora considerablemente la performance y da excelentes beneficios. Aún donde las estrategias de listas de candidatos no se usan explícitamente, las estructuras de memoria dan modificaciones eficientes de evaluaciones de *movimiento* de una iteración a otra, y reducen el esfuerzo de encontrar el mejor *movimiento*.

Ejemplos de estrategias para la lista de candidatos

Estrategia Aspiración Plus



El eje horizontal representa el esfuerzo computacional para examinar cada movimiento en el vecindario actual .

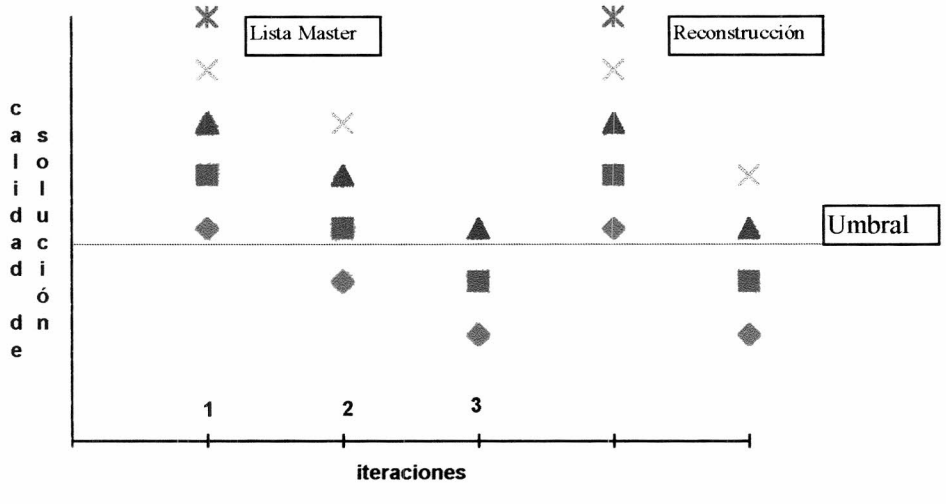
La línea de Aspiración es un umbral para la calidad del movimiento a

seleccionar, y se puede ajustar dinámicamente durante la búsqueda.

La estrategia examina los movimientos hasta encontrar uno que satisface el umbral, y en ese momento se examina un número adicional de movimientos (igual al valor Plus), y el mejor de todos se selecciona.

Estrategia de Lista de candidatos elite

Este método construye primero una lista Master examinando todos (o casi todos) los movimientos, y selecciona los k mejores, donde k es el parámetro del proceso. Entonces en cada iteración, se elige el mejor movimiento de la lista y se ejecuta. Se sigue hasta que un movimiento cae por debajo de un umbral de calidad, o hasta que se llega a una determinada cantidad de iteraciones. Entonces se construye una nueva lista y se repite el proceso.



Podemos ver que esta estrategia puede ser una variante de la de Aspiración, que permita examinar algunos movimientos fuera de la lista Master en cada iteración (donde esos sean de calidad suficiente como para reemplazar elementos de la lista Master).

Estrategia de filtros sucesivos

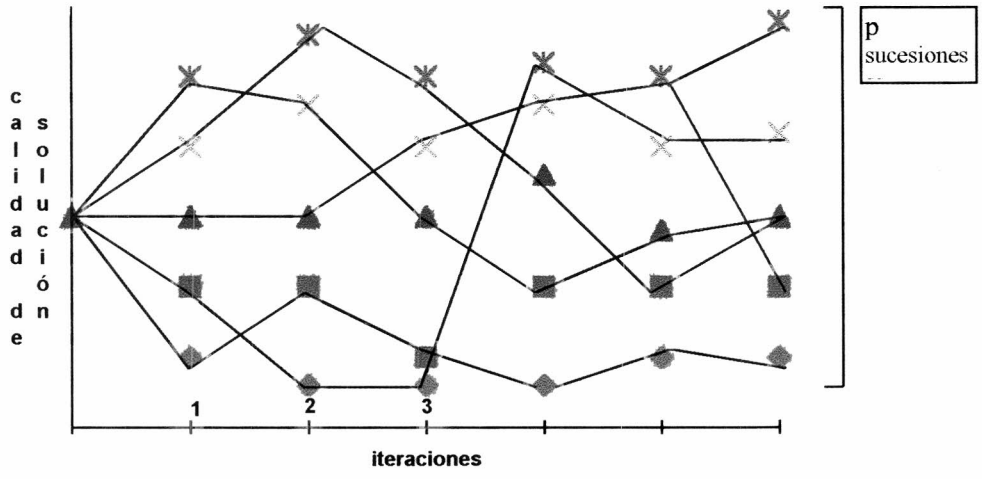
Los movimientos pueden separarse en las operaciones que lo componen, y el conjunto de movimientos revisados pueden reducirse restringiendo a aquellos que produzcan mejor calidad de beneficios para cada operación por separado. Por ejemplo, elegir un intercambio que incluye una componente de inserción y una de borrado (add y drop) puede restringir su atención sólo a los intercambios generados de un pequeño subconjunto del "mejor add" y el "mejor drop" que lo componen.

En problemas de secuenciamiento, una medida puede ser identificar atributos con información como fecha prevista, tiempo de procesamiento, penalidades por demora.

Si las permutas se usan, entonces alguna tarea es generalmente mejor candidata que otras para moverse antes o después en la secuencia. La lista de candidatas considera las permutas cuya composición incluye al menos uno de los atributos preferidos.

Lista de candidatos en Abanico Secuencial

Un tipo de lista de candidatos que es muy explotable por procesamiento en paralelo, es la lista de candidatos en abanico secuencial. La idea básica es generar algunos "p" mejores alternativas en un paso dado, y crear un abanico de soluciones, una para cada alternativa. Se revisan otra vez las mejores movidas, y sólo las "p" mejores sobre todas proveen las nuevas "p" sucesiones en el paso siguiente.



La iteración 0 construye una solución inicial. Las mejores movidas de esta solución se usan para generar p sucesiones. Entonces en cada iteración siguiente se seleccionan los mejores movimientos que guíen a buscar p soluciones diferentes. Notar que como más de un movimiento puede guiar a buscar la misma solución, más de "p" movimientos pueden necesitarse para continuar la exploración de p sucesiones distintas.

Listas de candidatos de cambios limitados

En esta estrategia , una solución mejorada se consigue restringiendo el dominio de las elecciones, por lo tanto ninguna componente de la solución cambia más de un determinado grado en cada paso. Un límite en este grado, expresado por una métrica de distancia apropiada al contexto, es seleccionarlo suficientemente grande para cercar las posibilidades consideradas estratégicamente relevantes. La métrica puede permitir grandes cambios a lo largo de una dimensión, pero limita los cambios a lo largo de la otra, así las elecciones se reducen y evalúan más rápido.

5. - ALGORITMO TABU SEARCH PROPUESTO

5.1- Características particulares

A continuación presentaremos las características particulares del algoritmo implementado, y las estructuras utilizadas para el almacenamiento de datos. También se detalla el entorno en que fue desarrollado e implementado el algoritmo.

Criterio de parada

La ejecución del programa finaliza cuando se produce el primero de los siguientes eventos:

⇒ Se alcanzó el máximo número de iteraciones solicitadas por el usuario.
⇒ Se alcanzó el máximo número de iteraciones admisibles de no mejora.

En ambos casos se genera un archivo de resultados.

Los resultados se guardan en un archivo de texto. (Ver detalle en Modulo Resultados item 5.2.15). Además en cada iteración se almacena en otro archivo de texto, los siguientes datos:

- Valor de la Función Objetivo
- Valor del reloj de la máquina

Este archivo fue creado para el estudio del comportamiento del algoritmo. Los datos almacenados en el mismo se reproducen en una planilla Excel (MSExcel 97), para facilitar la confección de un gráfico, que permite observar: los valores mínimo y máximo obtenidos, los tramos de la búsqueda en los cuales no se obtiene una mejoría en la solución y otros puntos de interés. El estudio

puede realizarse en función del tiempo o iteraciones. Los gráficos obtenidos en las ejecuciones de los distintos ejemplos, se pueden observar en la Segunda Parte del Apéndice.

Lista Tabu

La Lista Tabu está formada por una cantidad de permutaciones que llevaron a las *n* últimas soluciones obtenidas. La cantidad de permutaciones almacenadas es un parámetro ajustable por código. Además de la permutación se guarda el *tiempo de permanencia en la Lista Tabu* que le corresponde a cada una, es decir la cantidad de iteraciones que faltan para que la permutación salga de la Lista Tabu.

Tiempo de permanencia en la Lista Tabu

Se utiliza un parámetro dinámico para el tiempo de permanencia en la Lista Tabu, calculado a partir de un *tmin* y un *tmax* ingresado como parámetro. Se selecciona en forma aleatoria dentro del rango especificado, siguiendo una distribución uniforme. Y el resultado es aún mejor cuando el tiempo de permanencia se recalcula cada una cantidad determinada de iteraciones.

Criterio de aspiración

El criterio de aspiración es el que permite sacar un elemento de la Lista Tabu si cumple con ciertos requisitos. En nuestro desarrollo se utiliza el criterio de aspiración por default combinado con el de aspiración global, y trabaja de la siguiente forma:

* si todas las soluciones candidatas están en la Lista Tabu, entonces revoca es estado tabu de las permutaciones que les reste el menor tiempo de

permanencia en la lista tabu (criterio de aspiración por default).

* si la solución encontrada es Tabu pero mejora el valor de la Función Objetivo lograda en la mejor solución encontrada hasta el momento, revoca el estado tabu del elemento de la lista, y esta solución pasa a ser la seleccionada como candidata con el valor de la Función Objetivo que le corresponde (criterio de aspiración global).

Lista de candidatos

La lista de candidatos se obtiene armando un ranking de todas las soluciones vecinas, seleccionando las *n* mejores, es decir con mejor valor de Función Objetivo.

Soluciones mantenidas

Además de la lista de soluciones elite, usada en la estrategia de Intensificación , se mantienen las siguientes soluciones:

- ***Solución actual***: solución corriente en cada iteración, sobre la que se realizará un intercambio al efectuar la mejor permuta de la lista de candidatos, o bien se aplicará el criterio de aspiración. Esta solución seleccionada será la *solución actual* de la siguiente iteración
- ***Solución Inicial***: solución construida con la heurística inicial.
- ***Mejor solución histórica:*** mejor solución encontrada a lo largo de toda la búsqueda.

Parámetros de la búsqueda

Las distintas pruebas sobre cada uno de los problemas se realizaron modificando los valores de los siguientes parámetros:

- **Cantidad máxima de iteraciones (max_iter):**
  Este es un parámetro de finalización. Dado que el algoritmo tiene la característica de no quedarse en óptimos locales, aumentar este parámetro permitió que se encontrara una mejor solución en algunos casos.

- **Cantidad de iteraciones admisibles de no mejora (max_nomejora):**
  Este es un parámetro de finalización.
  El mismo evitó completar la cantidad máxima de iteraciones en los casos en que no se obtenía un mejor valor de la Función Objetivo durante max_nomejora iteraciones.
  Se experimentó asignándole múltiples valores, lo que influyó en otros parámetros de no mejoría, descriptos más adelante, definidos en función de él.

- **Porcentaje inicial de la búsqueda en el que se activa la estrategia de Diversificación (iteriniciales):**
  Este parámetro es un porcentaje de la cantidad total de iteraciones.
  Durante estas iteraciones, de cumplirse ciertas condiciones, se activa la diversificación. Se observó que activar permanentemente (100 %) o no activar (0 %) la diversificación, no mejoraba los resultados.
  Variaciones de este parámetro permitieron comprobar como algunos nodos que ocupaban permanentemente ciertas posiciones en la solución, se ubicaban en otras anteriormente no frecuentadas.
  Se realizaron múltiples variaciones de este parámetro, para estudiar el comportamiento del algoritmo. La mayoría de los mejores resultados obtenidos se lograron activando la diversificación durante el 75 % inicial de la búsqueda.

- **Porcentaje final de la búsqueda en el que se activa la estrategia de Intensificación (iterfinales):**
  Este parámetro es un porcentaje de la cantidad máxima de iteraciones.
  Durante estas iteraciones de cumplirse ciertas condiciones se activa la intensificación.
  Se observó que activar permanentemente (100 %) o no activar (0 %) la intensificación, no mejoraba los resultados.
  Se dieron diversos valores a este parámetro, en algunos casos aumentando paralelamente el número de soluciones elite mantenidas.
  Las estrategias de diversificación e intensificación son excluyentes, por lo tanto los porcentajes de las mismas nunca suman más que 100 %. Coherentemente con lo descripto en el parámetro anterior, los mejores resultados se obtuvieron con el porcentaje del 25 %.

- **Cantidad máxima de iteraciones en que puede dividirse la etapa de Intensificación (dif_iter_reini):**
  Este parámetro depende del porcentaje anterior, es decir, es la cantidad de partes en que pueden dividirse las iteraciones finales.
  De cumplirse ciertas condiciones, este parámetro limita la cantidad máxima de veces en que se realiza intensificación en la etapa respectiva.

- **Cantidad máxima admisible de iteraciones de no mejora para activar estrategia de Intensificación (maxnomejora_reini):**
  Este parámetro es un porcentaje de la cantidad máxima admisible de iteraciones de no mejora.
  Si durante maxnomejora_reini iteraciones no se obtiene un mejor valor de la Función Objetivo, de cumplirse además otras condiciones, se activa la reinicialización de la búsqueda, continuando la misma desde la mejor solución elite en ese momento.
  Se probaron en algunos problemas diversos valores de este porcentaje, aumentando al mismo tiempo la cantidad de iteraciones finales de intensificación y la cantidad máxima de iteraciones en que puede dividirse dicha etapa, combinando con un aumento en la cantidad de soluciones elite

mantenidas.

Los mejores resultados se obtuvieron con este parámetro fijado en 25 %.

- **Cantidad máxima admisible de iteraciones de no mejora para activar estrategia de Diversificación (diver_nomejora):**

  Este parámetro es un porcentaje de la cantidad máxima admisible de iteraciones de no mejora.

  De estar en las iteraciones iniciales de diversificación, si durante diver_nomejora iteraciones no se obtiene un mejor valor de la Función Objetivo, se activa dicha estrategia.

  Se realizaron variaciones de este parámetro conjuntamente con un aumento en el porcentaje de diversificación de la búsqueda, y en la cantidad de elementos de la lista de candidatos, con el objetivo de orientar la búsqueda hacia otras regiones. Esto provocó un aumento de los valores de la función de penalidad de cada movimiento y consecuentemente un ordenamiento distinto de la lista de candidatos.

  Los mejores resultados se obtuvieron con este parámetro fijado en 25 %.

- **Factor de Diversificación calculado en forma aleatoria dentro de los límites mínimo y máximo (dados por un pivote y una tolerancia respectivamente). El pivote y la tolerancia son variables (Pivote: sinicialfo Tolerancia: límites_diver)**

  El factor de diversificación es un parámetro ajustable que toma valores aleatorios, cuando se activa la diversificación, siguiendo una distribución uniforme, dentro del rango dado por el pivote más/menos la tolerancia, o toma valor cero al desactivarse.

  Cuando toma valores distintos de cero provoca resultados positivos de la función de penalidad de cada movimiento, que se suma al valor de la Función Objetivo de los mismos.

  Se probó asignando diversos valores al pivote, pero el mejor ajuste de este factor se logró tomando el valor de la Función Objetivo de la solución inicial como pivote, y una tolerancia igual al 25 % de este valor.

- **Tamaño de la Lista Tabu (tfil):**
  Este parámetro fija la cantidad de elementos que pueden permanecer en la Lista Tabu.
  La mayoría de los resultados se obtuvieron con un tamaño entre 15 y 20.

- **Límites máximo y mínimo del tiempo de permanencia en la Lista Tabu (tt):**
  Este parámetro (tt) es la máxima cantidad de iteraciones que un elemento permanece en la Lista Tabu.
  Este valor se selecciona de manera aleatorio dentro del rango [tmin, tmax], siguiendo una distribución uniforme.
  Las pruebas con este parámetro se realizaron combinando conjuntamente con distintos tamaños de la Lista Tabu, para permitir una menor o mayor permanencia de los elementos de dicha lista.
  La mayoría de los resultados se obtuvieron con una permanencia mínima (tmin) de 5 y máxima (tmax) de 10 iteraciones, de los elementos en la Lista Tabu.

- **Cantidad máxima de iteraciones con un mismo tiempo de permanencia en la Lista Tabu (cant_tt):**
  Este parámetro constituye la cantidad de iteraciones que deben transcurrir para obtener un nuevo parámetro tt(tiempo de permanencia en la Lista Tabu) aleatorio.
  La mayoría de las pruebas se realizaron modificando el tiempo de permanencia en la Lista Tabu cada 5 iteraciones.

- **Tamaño de la Lista de Candidatos o vecindario restringido (mfil):**
  Este parámetro fija la cantidad de mejores permutaciones del vecindario que serán seleccionadas en cada iteración.
  En los casos en que el resultado no era bueno se aumentó el tamaño del vecindario conjuntamente con un aumento del tiempo de permanencia en la Lista Tabu , con el objetivo de explorar otras regiones del espacio de búsqueda.

La mayoría de los resultados se obtuvieron con un tamaño entre 15 y 20.

- **Tamaño Lista de Soluciones Elite (ofil):**
  Este parámetro fija la cantidad de soluciones elite mantenidas durante la búsqueda.
  En la mayoría de las observaciones se mantuvieron en la lista las cinco mejores soluciones encontradas.
  Se incrementó el tamaño de la lista, en las pruebas que permitieron mayor número de reinicializaciones, lo que posibilitó continuar explorando regiones aparentemente buenas.

- **Tamaño Lista de Mejores Soluciones Iniciales (max_cant_solini):**
  Este parámetro fija la máxima cantidad de posibles soluciones, obtenidas por la heurística inicial, por donde comenzar la búsqueda.
  En algunos problemas se encontraron mejores soluciones, tomando como solución inicial, alguna de la lista, no necesariamente la mejor respecto del valor de la Función Objetivo. En estos casos se experimentó con todas las posibles soluciones iniciales, aumentando para esto el valor del parámetro.

Movimiento

El *movimiento* para ir de una solución a otra que se usa en este algoritmo es una permutación. También existe un *movimiento* llamado de inserción pero para nuestro caso no es útil, ya que en cada iteración trabajamos con todos los nodos del grafo que componen la solución.

Este *movimiento* permite, a partir de la solución actual generar todas las soluciones vecinas (es decir, todas aquellas que se obtienen permutando el orden de dos trabajos de la secuencia).

Para la elección de la permutación que generará la solución candidata, se realiza un ranking entre las vecinas, evaluando para ello el valor de la Función Objetivo, de este modo se tiene en cuenta la calidad de la solución Finalmente, una vez seleccionada la permutación, se ejecuta el *movimiento* generando una nueva solución.

Calculo de Función Objetivo

Este cálculo se realiza una única vez para todos los nodos. Luego con cada permutación realizada, solo se evalúa qué elementos cambian su posición en la secuencia y se recalcula el valor únicamente de esa porción del vector.

Memoria a corto plazo

La memoria a corto plazo, sirve para tener en cuenta en cada iteración la historia reciente de los últimos movimientos ejecutados, y de los valores asociados a dichos movimientos.

Esta estrategia se aplica mediante el uso de la Lista Tabu. De esta forma, se evita la ocurrencia de ciclos de tamaño menor o igual a la longitud de la Lista Tabu.

Control de ciclos

El control de ciclos se realiza en el momento del ingreso de la información que corresponde al grafo de precedencias. Se controla que no existan en el digrafo P las aristas (i,j) y (j,i), en forma directa o transitiva.

Memoria a largo plazo

Se basa en guardar puntos importantes de la historia de la búsqueda. Las estrategias asociadas al uso de este tipo de memoria son Diversificación e Intensificación.

Diversificación

La estrategia se basa en la cantidad de veces que cada uno de los trabajos está en cada posición de la secuencia. Para ello, se utiliza una matriz de residencia, donde cada fila indica el número de trabajo y cada columna la posición en la secuencia.

Matriz M de residencia

POSICION	1	2	3	n
TRABAJO				
1	3			
2		1	5	2
3	4			
		1		
N	4		3	2

Se penalizan soluciones similares, para provocar el cambio en el espacio de búsqueda. La penalidad se calcula como:

penalidad $_{ij}$ = M_{ij} / (mayor frecuencia de la matriz)

Penalidad del movimiento = penalidad $_{ij}$ + penalidad $_{ji}$

El valor del movimiento candidato se cambia por un nuevo valor penalizado, que es:

Valor penalizado= valor del movimiento + d * penalidad del movimiento

d es un parámetro ajustable de diversificación.

La diversificación se activa cuando se cumple una determinada cantidad de iteraciones de no mejora (parámetro diver_nomejora descripto en los parámetros de la búsqueda), y si la iteración corresponde al porcentaje inicial de la búsqueda donde se activa la estrategia (iteriniciales descripto en los parámetros de la búsqueda). Se penaliza hasta que se encuentra una solución que minimiza la Función Objetivo.

Intensificación

Se aplica mediante una lista de soluciones elite, que son las *n* mejores soluciones encontradas en toda la historia de la búsqueda. Junto con estas soluciones se guarda la memoria Tabu asociada. La cantidad de soluciones elite, es un parámetro variable.

La estrategia trabaja de la siguiente forma: En la porción final de la búsqueda, si no se mejora el valor de la Función Objetivo, se reinicializa la búsqueda desde la mejor solución de la lista elite, con la memoria Tabu asociada.

La estrategia de Diversificación se utiliza en la primera parte de la búsqueda, y la Intensificación en la parte final.

Entorno de Desarrollo e Implementación

El algoritmo fue desarrollado en C++ versión 3.0 de Borland, pero las ejecuciones están realizadas bajo Microsoft Visual C++ 5.0, dado que al trabajar en Windows con un compilador de 16 bits se utiliza un límite de memoria insuficiente para la ejecución de problemas mayores a un tamaño determinado, por lo tanto fue necesario cambiar a un compilador de 32 bits.

Se implementó en una PC Pentium II 266 Mhz, 64 MB de RAM. Sistema Operativo MS Windows 95.

El algoritmo permite el ingreso de datos por archivo de texto o por teclado.

- Se podrán ingresar los pesos de las aristas y el grafo de precedencias por archivo de texto y todos los parámetros de la línea de comando se ingresan por pantalla.
- Si el ingreso es por pantalla se ingresarán los pesos, el grafo de precedencias y todos los parámetros de la línea de comando por pantalla.

La entrada por archivo de texto se usó para los ejemplos provistos por L. F. Escudero (a través de M.T. Ortuño, quien nos envió los problemas ESC por E-mail), y para aquellos citados en [3], como 'Rbg' que fueron obtenidos de la página Web de Zib (www.zib.de)

Se consultó bibliografía para la selección de las estructuras para el almacenamiento de los datos [15]. Para el ingreso del grafo se eligió una matriz de adyacencia donde cada elemento *(i,j)*, es el costo asociado a ir del nodo *i* al nodo *j*. Para el grafo de precedencias se usa una lista de adyacencias donde cada lista de cada elemento del arreglo son los nodos a los que debe preceder el nodo cabeza.

Costos: los costos asociados a la secuencia de ejecución de los trabajos, se almacenan en una matriz de *nxn*, con la diagonal con valor cero.

Lista Tabu: La Lista Tabu se almacena en una matriz, que contiene en cada fila, la permutación "prohibida" y el tiempo de permanencia restante en la Lista Tabu que le corresponde.

El diseño modular del programa permitió una manipulación relativamente sencilla del mismo a medida que iba creciendo.

Cada módulo descripto en el item 5.2 está guardado en su propio archivo fuente, exportándose hacia el módulo principal.

El código esta comentado para facilitar la comprensión del mismo.

5.2.- Módulos principales del programa

5.2.1.- Módulo principal

Es el módulo que contiene el cuerpo principal del programa, y realiza las llamadas a los módulos Entrada (ingreso de datos de entrada), Solución inicial y Solución.

Módulo Principal

Comenzar

- Entrada de los datos del problema
- Construir lista de mejores Soluciones Iniciales
- Seleccionar la mejor Solución Inicial de la lista
- Ingresar los parámetros de la búsqueda

Repetir

- Búsqueda de una mejor Solución
- Guardar resultados obtenidos
- Seleccionar nueva Solución Inicial
- Ingresar nuevos parámetros de la búsqueda

Hasta (no se seleccionen soluciones iniciales)

Fin Principal

5.2.2.- Módulo Entrada de datos

Es el módulo que realiza la entrada de los datos. Los datos necesarios de entrada para poder ejecutar la búsqueda son:

- Costos asociados a la ejecución de los trabajos
- Restricciones de precedencia que se deben respetar.

Este módulo tiene la facilidad de permitir la entrada de los datos, tanto por pantalla como por archivo de texto, lo que permite la ejecución de problemas con una mayor cantidad de nodos . Esta facilidad fue la utilizada en las ejecuciones con problemas de la vida real facilitados por Laureano Escudero.

Los valores de los costos asociados se almacenan en una matriz de adyacencias, y las restricciones de precedencias en una lista de adyacencias, donde cada uno de los elementos del arreglo representa al nodo predecesor, y donde los elementos que componen cada lista, son los nodos sucesores correspondientes.

Este módulo además verifica si existen ciclos en el grafo de

precedencias, en ese caso emite un mensaje de error indicándole donde y como se forma el ciclo, dando la posibilidad de volver a ingresar el grafo cuando el ingreso de los datos es por pantalla.

5.2.3.- Módulo Verificación de Precedencias

Este módulo verifica si una solución determinada cumple con las restricciones de precedencia ingresadas. Recibe como entrada un vector que contiene la solución a verificar y genera como salida una señal indicando si cumple o no la condición. Esta condición determina que la solución sea o no factible para el problema.

Módulo Precedencias
Verifica si un vector solución cumple las precedencias, es decir si es o no factible.
Devuelve un 1 si la solución verifica precedencias y 0 en caso contrario.
Comenzar
Pos1 : = 2
Mientras (pos1 ≤ N) **Hacer** /* Para todos los nodos menos el primero */
- t : = lista de nodos precedidos por solución[pos1]
- **Si** (t ≠ ∅) **Hacer**
  - **Mientras** (t ≠ ∅) **Hacer**
    - x : = cabeza (t)
    - t : = cola (t)
    - pos2 : = posición de x en vector solución
    - **Si** (pos2 < pos1) **Entonces**
      - Retornar (0) /* Precedencia Violada */

      Fin Si

    Fin Mientras

  Fin Si
- pos1 : = pos1 + 1

Fin Mientras
Retornar (1) /* Vector Solución verifica precedencias */
Fin Precedencias

5.2.4.- Módulo Solución Inicial

Este módulo genera, si es que existe una solución inicial factible, para comenzar la búsqueda. Para ello utiliza la versión de la heurística miope del vecino más próximo que se debe a Belmore y Nemhauser (1988).(Ver[4])

El algoritmo consiste en partir de un vértice inicial y elegir en cada iteración el vértice (vecino) más proximo, al vértice en consideración, eliminándolo del conjunto de posibles. Este algoritmo es del orden $O(n^2)$ y el camino Hamiltoniano obtenido sufre fuerte influencia de la elección del vértice inicial.

Se comparan los resultados obtenidos con cada vértice inicial y se selecciona el mejor. Recibe como entrada el número correspondiente al vértice inicial, y genera como salida el vector conteniendo la solución inicial.

Para ello arma una lista ordenada, cuyo tamaño depende de un parámetro, que son las mejores soluciones iniciales, comenzando por distintos vértices. El algoritmo permite que se seleccione cualquiera de las soluciones de la lista, no necesariamente la de menor valor de la Función Objetivo.

Este módulo se encarga de verificar la factibilidad de la solución, verifica las restricciones de precedencia ingresadas, mediante la llamada al módulo correspondiente.

Módulo Solución Inicial

Arma una lista ordenada, en función del valor de la función objetivo, de las mejores soluciones iniciales. Utiliza la Heurística Golosa Vecino más Próximo para construir una solución inicial factible desde cada uno de los nodos.

Comienzo

Para Cada ($1 \leq i \leq N$) **Hacer** /*N: cantidad de nodos del problema*/

- Construir Solución Inicial factible comenzando en nodo i

Si (existe Solución Inicial factible comenzando en nodo i) **Entonces**

- Insertar Solución Inicial factible en lista ordenada de soluciones iniciales

Fin Si

Fin Para Cada

Fin Solución Inicial

Heurística Vecino más Próximo con control de precedencias

Construye una solución inicial factible, utilizando la Heurística Vecino más Próximo, comenzando en el nodo inicial pasado como argumento.
Devuelve la solución inicial construida, ó ∅ si no existe solución factible comenzando en el nodo inicial.

Comenzar
Armar vector auxiliar conteniendo la cantidad de predecesores de cada nodo
Armar vector auxiliar de nodos ubicados
Si (cantidad de predecesores del nodo inicial > 0) **Entonces**

- Retornar ∅ /* No existe solución factible comenzando en el nodo inicial*/

Si no

- Ubicar el nodo inicial en primer posición del vector solución
- Restar 1 a la cantidad de predecesores de todos los nodos precedidos por el nodo inicial.
- Marcar nodo inicial en vector de nodos ubicados

Para Cada (nodo no ubicado) **Hacer**

- Escoger un nodo tal que el costo de ir desde el nodo corriente a dicho nodo sea mínimo, y cuyos predecesores estén ubicados
- Ubicar el nodo seleccionado en vector solución
- Restar 1 a la cantidad de predecesores de todos los nodos precedidos por el nodo seleccionado
- Marcar nodo seleccionado en vector de nodos ubicados

Fin Para Cada

Fin Si

Fin Heurística Vecino más Próximo con control de precedencias

5.2.5.- Módulo Generación de la Solución

Se encarga de buscar la mejor solución mejorando la obtenida por la heurística inicial. Para ello, genera una lista de soluciones candidatas, elige el mejor movimiento, si ningún movimiento es admisible usa el Criterio de Aspiración por defecto y una vez encontrado el mejor movimiento, lo ejecuta. Además verifica la factibilidad de la solución.

- Recibe como entrada la solución inicial y genera como salida la mejor solución encontrada para el problema.

MODULO SOLUCION

Busca una mejor solución partiendo de la solución inicial.

Comenzar

- Inicialización de variables y parámetros de la búsqueda dependientes de los parámetros ingresados.
- **Mientras** (# iteración actual < cant_máx de iteraciones solicitadas) y (cant_iteraciones de no mejora < cant_máx admisible de iteraciones de no mejora) **Hacer**
  - **Si** (cant_iteraciones con un mismo t_permanencia en lista tabu = cant_máx permitida de iteraciones con igual t_permanencia en lista tabu) **Entonces**
    - Obtener aleatoriamente un nuevo tiempo de permanencia en lista tabu

    Fin Si
  - **Si** (Etapa de Diversificación) y (cant_iteraciones de no mejora > cant_máx admisible de iteraciones de no mejora de diversificación) **Entonces**
    - Obtener aleatoriamente un nuevo factor de diversificación

    Si no
    - Factor de diversificación = 0

    Fin Si
  - Armar Lista de Candidatos
  - Seleccionar Mejor Movimiento
  - **Si** (no encontró mejor movimiento para seleccionar) **Entonces**

    /*Todos las permutaciones candidatas son tabu y su estado tabu no fue revocado por Criterio de Aspiración Global*/
    - Usar Criterio de Aspiración por Defecto para seleccionar movimiento

    Fin Si
  - Insertar solución seleccionada en lista de soluciones elite
  - **Si** (Etapa de Intensificación) y (fo del movimiento seleccionado > fo de la mejor solución de la lista de soluciones elite) y (cant_iteraciones desde última reinicialización > cant_máx de iteraciones en que puede dividirse la etapa de Intensificación) **Entonces**
    - Reinicializar búsqueda desde mejor solución elite
    - Ultima Reinicialización = # iteración actual

    Fin Si
  - Ejecutar Mejor Movimiento seleccionado
  - Actualizar matriz de Residencias
  - Incrementar número de iteración

Fin Mientras

Fin Solución

5.2.6.- Módulo Generación de Lista de Candidatos

Genera la lista de los *m* mejores movimientos, donde *m* es un parámetro. Los mejores movimientos son el resultado de generar y recorrer el vecindario de la solución corriente, verificando que se cumplen las restricciones de precedencia y seleccionando finalmente los *m*, que resultan en un mejor valor de la Función Objetivo. Para ello se utilizan:

- Verificación de precedencias (Módulo Precede)
- Evaluación de Función Objetivo (Módulo Función Objetivo)

La generación del Vecindario es el resultado de todas las permutaciones posibles de los nodos en la solución corriente, lo que da un total de $n(n-1)/2$ soluciones posibles a ser examinadas.

La lista de candidatos de los *m* mejores, se almacena en una matriz de *m* filas. Cada fila contiene un movimiento. Por cada movimiento se guardan las posiciones de los nodos que se permutaron para obtener la solución vecina. Paralelamente, se guarda en un vector el valor de la Función Objetivo obtenido para cada permutación de la lista de candidatos.

Recibe como entrada, la solución actual, y genera como salida la lista de los mejores candidatos, y la lista de los valores de la función objetivo correspondientes a cada puna de las permutaciones. Tanto la matriz de Candidatos como el vector que contiene los valores de los funcionales, se devuelven ordenadas por mejor valor de la Función Objetivo.

Módulo Candidatos

Construye el vecindario restringido para la solución corriente, consistente de las k mejores permutaciones, con respecto al valor de la función objetivo, donde k es un parámetro del sistema.
Devuelve lista de Candidatos ordenada de menor a mayor, según valor de la función objetivo, y otras estructuras asociadas a la misma.

Comenzar
Para cada (permutación (i, j) de nodos entre las N (N - 1) / 2 posibles) **Hacer**
/*N: Cantidad de nodos del problema*/
Armar solución vecina permutando los nodos correspondientes de la solución corriente
Si (solución vecina cumple precedencias) **Entonces**
- Calcular valor de la función objetivo de la solución vecina
- Insertar vecino en lista de candidatos

Fin Si
Fin Para
Fin Candidatos

Insertar Vecino en Lista de Candidatos

Inserta una permutación en la lista de candidatos, manteniéndola ordenada según valor de la función objetivo.
Inserta el valor de la función objetivo de la solución resultante de la permutación.
Inserta la pena del movimiento resultante de la permutación.

Comenzar

- Calcular pena de inserción del nodo i en la posición j y del nodo j en la posición i como:
  Pena(x,y) = (frecuencia del nodo x en la posición y) / (máxima frecuencia de la matriz de residencias)
- Calcular la pena del movimiento como (factor de diversificación) * (pena(i,j) + pena(j,i))
- Insertar en lista de candidatos la permutación (i,j) en la posición que corresponda
  manteniendo la lista ordenada de menor a mayor según valor de la función objetivo
- Insertar valor de la función objetivo de la permutación (i,j) en vector de función objetivo,
  manteniendo la estructura ordenada según lista de candidatos
- Insertar pena del movimiento en vector de penas manteniendo la estructura ordenada según lista de candidatos

Fin Insertar vecino en Lista de Candidatos

5.2.7.- Módulo Mejor Movimiento

Selecciona el mejor movimiento de la Lista de Candidatos, evaluando el mérito del mismo. Este mérito se refiere al valor de la Función Objetivo, del movimiento. Otra característica funcional de este módulo consiste en verificar si el movimiento es o no admisible. Un movimiento es admisible si no es Tabu o si

su estado Tabu se ignora utilizando criterio de aspiración global.

Recibe como entrada la lista de los mejores candidatos, y la lista de funcionales *Top* generadas por el módulo de Candidatos.

Genera como salida el mejor movimiento seleccionado, es decir, los nodos que componen la permutación que genera la mejor solución del vecindario.

Módulo Mejor Movimiento
Selecciona la mejor permutación de nodos de la lista de candidatos.
Devuelve la posición en la lista de candidatos, correspondiente a la mejor permutación seleccionada, ó (-1) si no encontró una permutación para seleccionar.
Comienzo
Mientras (existan permutaciones en la lista de candidatos) y (no encontró la mejor permutación) **Hacer**
    Si (permutación actual es tabu) **Entonces**
        Si (estado tabu puede ser revocado por criterio de aspiración global) **Entonces**
            - Encontró la mejor permutación tabu
        Si no
            - Avanzar en lista de candidatos
        Fin Si
    Si no
        - Encontró la mejor permutación no tabu
    Fin Si
Fin Mientras
Si (encontró mejor permutación) **Entonces**
    - Retornar (posición mejor permutación)
Si no
    - Retornar (-1)
Fin Si
Fin Mejor Movimiento

5.2.8.- Módulo Ejecución del Mejor Movimiento

Ejecuta el mejor movimiento que fue generado por el módulo Mejor Movimiento, reemplazando la solución actual por la que genera la permutación elegida. Además verifica, si el valor de la Función Objetivo de la solución generada es mejor que el valor histórico guardado desde el principio de la búsqueda, si es así actualiza tanto el valor de la Función Objetivo histórico como el valor de la mejor solución hallada. Además realiza la actualización de la Lista Tabu, por medio de la llamada al módulo correspondiente.

Recibe como entrada la solución actual y el valor de la Función Objetivo actual y genera como salida la nueva solución, con el valor de Función Objetivo actualizado.

Módulo Ejecutar Movimiento
Ejecuta el mejor movimiento seleccionado.
Comenzar
Actualizar solución corriente realizando la permutación de nodos seleccionada.
Actualizar el valor de la función objetivo de la solución corriente.
Si (fo de mejor solución histórica < fo de solución corriente) **Entonces** /*fo : Función Objetivo*/

- Incrementar parámetro de no mejoría

Si no

- Parámetro de no mejoría = 0

Fin Si
Si (fo de solución corriente < fo de mejor solución histórica) **Entonces**

- Actualizar mejor solución histórica
- Actualizar valor de la función objetivo de mejor solución histórica
- Actualizar número de iteración de mejor solución histórica

Fin Si
Actualizar lista tabu
Fin Ejecutar Movimiento

5.2.9.- Módulo Actualización de la Lista Tabu

Actualiza la Lista Tabu agregándole un elemento, que es la permutación que produce el mejor movimiento, y actualizando el tiempo de permanencia en la Lista Tabu de todos los elementos de la lista, eliminando aquellos cuyo tiempo de permanencia es cero.

Recibe como entrada el elemento a agregar en la lista , es decir la permutación y genera como salida la Lista Tabu actualizada.

Módulo Tabu

Procedimiento Es_Tabu
Verifica si una permutación de nodos, pasada como argumento, es tabu

Comenzar
Mientras (existan elementos en la lista tabu) y (no encontró la permutación de nodos en la
lista tabu) **Hacer**
Si (permutación de nodos = elemento corriente de la lista tabu) **Entonces**
- Permutación es tabu

Si no
- Avanzar en lista tabu

Fin Si
Fin Mientras
Si (encontró permutación de nodos en lista tabu) **Entonces**
- Insertar en lista de candidatos cuyos elementos son tabu: la posición de la permutación en lista de candidatos y tiempo de permanencia de la permutación

Fin Si
Fin Es_Tabu

Procedimiento Actualizar Lista Tabu

Inserta el par de nodos, correspondientes a la permutación pasada como argumento, en la lista tabu y mantiene actualizada dicha lista.

Para Cada (par de nodos de la lista tabu) **Hacer**

Si (permanencia en la lista tabu del par de nodos = 0) **Entonces**

/*Fin permanencia de la permutación en la lista*/

- Eliminar par de nodos de la lista
- Restar 1 a cantidad de elementos tabu

Si no

- Restar 1 a permanencia en la lista tabu del par de nodos

Fin Si

Fin Para

Inserta par de nodos en lista tabu

Sumar 1 a cantidad de elementos tabu

Fin Actualizar Lista Tabu

Procedimiento Eliminar_Elemento de la Lista Tabu

Elimina de la lista tabu la permutación de nodos, que recibe como argumento.

Comenzar

Mientras (existan elementos en la lista tabu) y (no encontró permutación de nodos en la lista) **Hacer**

Si (elemento de la lista tabu = permutación de nodos) **Entonces**

/*Encontró permutación de nodos*/

- Eliminar permutación de nodos
- Decrementar cantidad de elementos tabu en 1

Si no

- Avanzar en la lista tabu

Fin Si

Fin Mientras

Fin Eliminar Elemento de la Lista Tabu

5.2.10.- Módulo Tiempo de Permanencia en la Lista Tabu

Este módulo calcula el Tiempo de permanencia en la Lista Tabu, que es la medida en cantidad de iteraciones, que un atributo permanece en esa lista. En este caso se calcula un Tiempo de Permanencia dinámico, que se selecciona en forma aleatoria entre dos parámetros *tmin* y *tmax* siguiendo una distribución uniforme. El Tiempo de Permanencia calculado en este módulo es un α tiempo de permanencia, siendo α la cantidad de iteraciones en que se mantiene constante, volviéndose a calcular al término de las mismas.

Recibe como entrada los parámetros *tmin* y *tmax*, y genera como salida el valor del tiempo de permanencia en la Lista Tabu.

5.2.11.- Módulo Criterio de Aspiración

La función principal de este módulo es determinar si una Regla Tabu puede ser violada. Para ello utiliza dos criterios de Aspiración:

- Aspiración Global: Revoca el Estado Tabu del movimiento si este, es mejor, en cuanto al valor de la Función Objetivo, con respecto al obtenido en la mejor solución histórica, que se actualiza en el módulo de ejecución del mejor movimiento.
- Aspiración por Defecto: Si todas las permutaciones no son admisibles, revoca el estado tabu de aquellos movimientos, que les resta la menor cantidad de iteraciones para salir de la lista Tabu.

Módulo Criterios de Aspiración

Aspiración Global

Revoca el estado tabu del movimiento, si este produce una solución mejor que la mejor obtenida hasta el momento.
Devuelve 1 si revoca el estado tabu del movimiento y 0 en caso contrario.

Comenzar
Si (fo del movimiento < fo del mejor movimiento histórico) **Entonces** /*fo : Función Objetivo*/

- Revocar el estado tabu del movimiento /* Eliminar elemento seleccionado de la lista tabu*/
- Retornar (1)

Si no

- Retornar (0) /*No se revoca estado tabu del movimiento*/

Fin Si
Fin Aspiración Global

Aspiración por Defecto

Selecciona una permutación de la lista de candidatos cuyo tiempo restante de permanencia en la lista tabu sea el menor, y revoca su estado tabu.

Retorna la posición, en la lista de candidatos, de la permutación cuyo estado tabu fue revocado.

Comenzar
Posición_seleccionada = posición primer permutación de la lista de candidatos cuyos elementos son tabu
Para cada (permutación de la lista de candidatos cuyos elementos son tabu) **Hacer**

Si (tiempo de permanencia en la lista tabu del candidato corriente < tiempo de permanencia en la lista tabu del candidato seleccionado) **Entonces**

- Posición_seleccionada = posición en lista de candidatos del candidato corriente

Fin Si

Fin Para
Revocar el estado tabu del movimiento /* Eliminar elemento seleccionado de la lista tabu*/
Retornar (posición en lista de candidatos de la permutación cuyo estado tabu fue revocado)
Fin Aspiración por Defecto

5.2.12.- Módulo Función Objetivo

Este módulo evalúa el valor de la Función Objetivo para una solución determinada.

5.2.13.- Módulo Intensificación

Guarda en una lista ordenada, según el valor de la Función Objetivo, las mejores soluciones encontradas durante toda la historia de la búsqueda. Esta lista se llama lista de Soluciones Elite. En el momento de almacenar una solución Elite, se resguarda también el entorno necesario, para poder reinicializar la búsqueda en ese punto. Este entorno consta de la Lista Tabu, el próximo movimiento posible de ser realizado desde dicha solución, el valor de la Función Objetivo y otras estructuras necesarias.

La estrategia de Intensificación, se activa cuando no se logra mejorar el valor de la Función Objetivo, en la etapa final de la búsqueda, y busca mejorar la solución teniendo en cuenta, las mejores soluciones encontradas hasta el momento, y tratando de reinicializar la búsqueda desde ese punto. En el momento de la reinicialización, se elige la primera solución de la Lista Elite, esto es la mejor, y se actualizan todas las estructuras con los datos de aquellas que fueron guardadas en el momento de ingresar dicha solución a la Lista Elite. Se reordena además dicha lista de soluciones y todas las estructuras del entorno asociados a la misma.

Módulo Soluciones Elite

Procedimiento Insertar Solución Elite

Inserta una solución en lista de soluciones elite.

Comenzar

Si (solución no esta en la lista elite) **Entonces**

/*Guardar entorno actual */

- Insertar solución en posición correspondiente manteniendo la lista ordenada de menor a mayor según valor de la función objetivo de cada solución
- Insertar en posición correspondiente el valor de la función objetivo de la solución elite a insertar
- Insertar en posición correspondiente la lista tabu actual.
- Insertar en posición correspondiente el próximo mejor movimiento a ser seleccionado, desde la solución elite a insertar

Fin Si

Fin Insertar Solución Elite

Procedimiento Reinicializar

Reconstruye el entorno para continuar la búsqueda desde la mejor solución elite de la lista.

Comenzar

- Solución corriente = mejor solución elite
- Valor función objetivo = valor función objetivo de solución elite seleccionada
- Lista tabu = lista tabu correspondiente a solución elite seleccionada
- Lista de candidatos = próximo mejor movimiento seleccionado desde solución elite
- Dejar marca de solución visitada en lista de soluciones elite

Fin Reinicializar

5.2.14.- Módulo Diversificación

Este módulo devuelve un parámetro ajustable de diversificación. Dicho parámetro se selecciona de manera aleatoria dentro del rango dmín, dmax siguiendo una distribución uniforme.

5.2.15.- Módulo Resultados

Este módulo almacena en una archivo de texto todos los resultados de la búsqueda.

Genera un archivo para cada problema , si aún no existe, y de lo contrario agrega al existente los resultados obtenidos. Los datos almacenados para cada ejecución de cada ejemplo son los siguientes:

- Fecha y hora de ejecución
- Nombre del problema
- Cantidad de nodos
- Solución obtenida con la heurística inicial
- Función Objetivo de la solución inicial
- Mejor solución encontrada
- Función Objetivo de la mejor Solución Encontrada
- Tiempo total de ejecución
- Tiempo de Procesamiento que demora en encontrar la mejor Solución
- Cantidad máxima de iteraciones
- Número de iteraciones realizadas
- Número de iteración donde se encontró la mejor solución
- Cantidad de reinicializaciones

6.- RESULTADOS

La descripción de las columnas en las tablas de resultados es la siguiente:

N : número de nodos

|R|: número de relaciones de precedencias (sin las precedencias derivadas transitivamente)

Solución: solución óptima encontrada

Heur. Inicial: valor de la solución obtenida por la heurística inicial

CPU : tiempo de CPU que se necesitó para resolver el problema.

Mejor iter: número de iteración en la que se obtuvo el mejor valor de Función Objetivo

Total de iter: número total de iteraciones de la ejecución

Algoritmo Branch & Cut: resultados obtenidos por el algoritmo desarrollado por N. Ascheuer , que se toma como referencia para la comparación de resultados

Algoritmo propio: resultados obtenidos por el algoritmo desarrollado, con una técnica de tipo Tabu Search, para el presente trabajo

Se presentan dos algoritmos Tabu Search, dependiendo de los tipos de memoria utilizados:

1) Tabu Search Básico (TSB): es el algoritmo propuesto usando solo estrategia de memoria a corto plazo. Los resultados obtenidos con este algoritmo se muestran en el item a)
2) Tabu Search enriquecido (TSE): es el algoritmo propuesto usando estrategias de memoria a corto y largo plazo, con estrategias de Intensificación y Diversificación. Los resultados obtenidos con este algoritmo se muestran en los items b) y c). En b) se ejecutaron los problemas en el entorno C++ 3.0. En c) se observan las ejecuciones de los mismos problemas que en b) agregados aquellos de mayor dimensión, procesando bajo entorno Visual C++ 5.0.

TABLAS DE RESULTADOS

a) Resultados obtenidos con el algoritmo Tabu Search básico:

Algoritmo Branch & Cut		Algoritmo propio TSB	
Problema	Solución	Solución	Heur. Inicial
Esc07	2125	2600	2625
Esc11	2075	2273	2715
Esc12	1675	1760	1904
Esc14	2125	2625	2625
Esc25	1681	2400	2400
Esc47	1288	3027	3505
Esc63	63	63	68
Esc78	18230	19265	22600
Rbg019a	198	198	198

b) Resultados obtenidos con el algoritmo básico enriquecido con estrategias de Intensificación y Diversificación, ejecutados bajo C++ 3.0 de Borland.

Algoritmo Branch&cut						Algoritmo propio TSE				
Problema	n	|R|	Solución	Heur. Inicial	CPU (Sun Sparc 10)	Solución	Heur. Inicial	CPU (Pentium II)	Mejor iter.	Total de iter
Esc07	7	6	2125	2550	0:00.05	2125	2625	0:00:00	15	15
Esc11	11	3	2075	2129	0:00.11	2075	2715	0:00:01	633	2000
Esc12	12	7	1675	1760	0:00.42	1675	1904	0:00:00	186	200
Esc14	14	12	2125	2125	0:00.18	2125	2625	0:00:00	1206	2000
Esc25	25	9	1681	2372	0:01.93	2003	2400	0:00:10	4790	5000
Esc47	47	10	1288	2609	1:00.77	2950	3503	0:01:13	3014	4000
Esc63	63	95	62	63	0:00.69	62	68	0:03:14	996	1000
Esc78	78	77	18230	20130	0:04.43	18575	22600	0:02:35	594	1000
Rbg019a	19	43	198	198	0:00.23	198	198	0:00:00	0	0
Rbg019b	19	57	199	214	0:00.07	199	243	0:00:03	306	1000
Rbg021a	21	68	158	183	0:00.29	159	236	0:00:01	377	500
Rbg023a	23	79	155	193	0:00.23	155	273	0:00:01	153	500
Rbg048a	48	192	351	396	0:28.85	396	483	0:00:50	340	400
Rbg049a	49	241	355	425	0:01.14	417	473	0:01:18	376	500
Rbg050a	50	225	400	460	0:11.69	457	500	0:01:06	290	500
Rbg050c	50	256	467	497	0:02.36	495	558	0:00:38	158	300
Rbg068a	68	249	609	689	0:01.45	726	917	0:05:12	198	400

c) Tabla de resultados obtenidos con el algoritmo Tabu Search básico enriquecido con Intensificación y Diversificación, ejecutado en entorno Visual C++ 5.0, dado que al trabajar en Windows en la versión 3.0 de C++ de Borland, se utiliza un compilador de 16 bits que utiliza un límite de memoria insuficiente para la ejecución de problemas de mayor tamaño.

			Algoritmo Branch&cut			Algoritmo propio TSE				
Problema	n	|R|	Solución	Heur. Inicial	CPU (Sun Sparc 10)	Solución	Heur. Inicial	CPU (Pentium II)	Mejor iter.	Total de iter
Esc07	7	6	2125	2550	0:00.05	2125	2625	0:00:00	15	25
Esc11	11	3	2075	2129	0:00.11	2075	2715	0:00:00	429	650
Esc12	12	7	1675	1760	0:00.42	1675	1907	0:00:00	69	250
Esc14	14	12	2125	2125	0:00.18	2125	2625	0:00:01	375	1300
Esc25	25	9	1681	2372	0:01.93	1747	3360	0:00:30	7509	10000
Esc47	47	10	1288	2609	1:00.77	2367	3503	0:01:57	4610	5000
Esc63	63	95	62	63	0:00.69	62	68	0:02:36	27	500
Esc78	78	77	18230	20130	0:04.43	18640	22600	0:01:55	414	600
Esc98	98	84	2125	2125	0:15.09	2625	2625	0:00:53	1	100
Rbg019a	19	43	198	198	0:00.23	198	198	0:00:00	0	1
Rbg019b	19	57	199	214	0:00.07	199	243	0:00:03	119	500
Rbg021a	21	68	158	183	0:00.29	158	236	0:00:06	564	1500
Rbg023a	23	79	155	193	0:00.23	155	273	0:00:04	286	500
Rbg029a	29	76	217	248	0:02.82	221	270	0:00:47	4730	5000
Rbg048a	48	192	351	396	0:28.85	389	483	0:02:06	269	500
Rbg049a	49	241	355	425	0:01.14	407	473	0:05:21	427	1000
Rbg050a	50	225	400	460	0:11.69	447	500	0:02:08	334	500
Rbg050b	50	258	397	465	0:02.93	421	558	0:04:51	148	600
Rbg050c	50	256	467	497	0:02.36	499	558	0:05:32	174	500
Rbg068a	68	249	609	689	0:01.45	738	917	0:02:50	56	100
Rbg088a	88	547	1130	1245	7:11.31	1240	1370	0:04:38	17	50

Rbg092a	92	573	[1036, 1037]	1098	(*)	1084	1246	0:51:34	186	250
Rbg094a	94	457	1336	1401	0:02.05	1398	1548	7:42:54	908	1700
Rbg105a	105	682	[994. 1029]	1224	(*)	1213	1357	3:18:25	363	500
Rbg109a	109	622	[1035, 1038]	1117	(*)	1165	1473	4:56:25	237	500
Rbg126a	124	369	1381	1587	31:17.62	1584	1760	2:22:51	149	200
Rbg148a	146	976	[1398, 1399]	1554	(*)	1624	1839	8:49:51	101	200
Rbg161a	159	472	1962	2178	0:20.64	2224	2337	3:02:06	23	70
Rbg174a	174	1113	[2030, 2033]	2193	(*)	2218	2240	6:58:11	35	70
Rbg190a	188	725	[2227, 2244]	2442	(*)	2585	2732	13:52:55	59	100
Rbg285a	283	820	3482	3857	102:22.40	3887	4268	47:09:26	41	50

(*): Cancela la ejecución por alcanzar el tiempo máximo de procesamiento permitido. En todos los casos mencionados en [3], el tiempo máximo permitido es de 5*[n/100] horas de CPU.

En estos casos se indica el rango de valores donde está la solución.

Los tiempos de CPU para el caso de Branch & Cut está medido en mm:ss.cc(minuto, segundo, centésima), y para el algoritmo propuesto está medido en hh:mm:ss (hora, minuto, segundo).

El algoritmo Branch & Cut fue desarrollado en C, en un entorno SUN SPARC 10 con 64 MB de memoria bajo SUN OS 5.5

Los problemas Esc07-Esc78 fueron provistos por Escudero y corresponden a datos de Producción de IBM. Los rbg019a - rbg285a corresponden a datos obtenidos de la rutina de una grúa apiladora en un sistema automático de almacenamiento. Los archivos de texto con que se ejecutaron estas pruebas, fueron provistos por María Teresa Ortuño, persona a la cual nos

derivó Laureano Escudero, ella nos envió los problemas Esc, y el resto de los problemas identificados por rbg, fueron extraídos de la página Web, indicada oportunamente por ellos.

Como conclusiones de las ejecuciones realizadas, con las diferentes versiones de la técnica tabu, tenemos que:

1. **Algoritmo Tabu search básico**: Los resultados en general no fueron buenos, se puede ver en la tabla presentada en el item a) de este mismo capítulo, que el algoritmo no mejora significativamente la solución obtenida con la Heurística Inicial.

2. **Algoritmo Tabu Search enriquecido con estrategia de Diversificación**: se aplicó la estrategia de Diversificación que se define en la Sección anterior y mejoraron los resultados con respecto al algoritmo básico. Se observó que se cambia el espacio de búsqueda forzando a los nodos a ocupar diferentes posiciones en el vector solución. En algunos casos se observaron cambios notables, ya que determinados nodos ocupaban siempre las mismas posiciones antes de la implementación de la estrategia. La diversificación se activa durante las iteraciones iniciales, y se desactiva en la parte final de la búsqueda. (Estos valores son parametrizables por código). Se hicieron pruebas variando el factor de diversificación. Este factor parece estar vinculado a los pesos de las aristas. Se propone un factor de diversificación aleatorio en un rango dmín, dmax con una distribución uniforme, donde dmin y dmax son iguales al valor de la Función Objetivo de la solución encontrada mediante la aplicación de la heurística inicial moviéndose en un rango que actualmente está implementado como el valor de la Función Objetivo de la solución hallada por la Heurística Inicial / 4. (Estos valores son parametrizables por código). Con la implementación de esta estrategia, aparecen los primeros resultados exactos para algunos problemas.

3. **Algoritmo Tabu Search enriquecido con Estrategia de Intensificación y Diversificación:** Se implementó la estrategia explicada en la Sección anterior. Esta estrategia se activa durante las iteraciones finales (Este valor es parametrizable por código). Se propone para la obtención de mejores resultados reinicializar más de una vez dejando una marca en la solución elite seleccionada, después de cierto número de iteraciones de no mejora. (Este valor también es parametrizable por código). Se propone además que la estrategia de Intensificación se active cuando ha transcurrido desde la última reinicialización una cantidad de iteraciones mayor a un parámetro. (También parametrizable en el código). En la experiencia de las ejecuciones se vio que en algunos ejemplos no reinicializó nunca en toda la búsqueda, mientras que en otros para lograr el mejor resultado fue necesario hacerlo hasta el maximo permitido. Con el enriquecimiento de esta estrategia, los resultados mejoraron notablemente, dando en algunos ejemplos resultados exactos. En algunos problemas donde, a pesar de la implementación de ambas estrategias, los resultados no eran buenos, se propuso y desarrolló un ranking de soluciones iniciales, ordenadas por el valor de la Función Objetivo. Las soluciones son las halladas con la Heurística Inicial (vecino más próximo, enriquecida con elección del vértice inicial y validación de restricciones de precedencia). El objetivo es comenzar la búsqueda por distintas soluciones iniciales. La cantidad de soluciones iniciales del ranking es parametrizable por código. Con esta implementación, se mejoraron algunos resultados, y se observó que la solución inicial seleccionada no fue necesariamente la mejor con respecto al valor de la Función Objetivo.

   Se observó, además, en otros ejemplos que fue necesario aumentar el número de iteraciones para hallar mejores resultados. Queda para un futuro trabajo implementar y mejorar aún más dichos resultados, un ranking de soluciones obtenidas por el algoritmo variando ciertos parámetros.

   Otra posibilidad para optimizar resultados es variar los tamaños de diferentes estructuras como Lista Tabu, Lista de Soluciones Elite, Cantidad de Soluciones Iniciales, etc. Los resultados de dichas variaciones se describen en el Capítulo 5- Parámetros de la Búsqueda.

Comparando los resultados de las tablas b) y c), se ve con claridad que además de haber podido ejecutarse ejemplos de mayores dimensiones, se pudo mejorar el tiempo de ejecución

Más detalles sobre los resultados obtenidos, y las soluciones encontradas tanto inicial como la lograda con la técnica Tabu, se muestran en el Apéndice. Allí se encuentran los archivos de resultados obtenidos para las últimas ejecuciones de cada problema bajo entorno Visual C++ 5.0.

7- CONCLUSIONES

Analizando las tablas de resultados presentadas en el capítulo anterior , se debe tener en cuenta que el algoritmo Branch & Cut es un método exacto y el algoritmo desarrollado en el presente trabajo es una heurística, por lo cual se concluye que las soluciones obtenidas son razonablemente buenas. Las mismas fueron logradas con un algoritmo de menor complejidad que el que requiere un método exacto.

El objetivo de este trabajo fue mostrar el potencial de una técnica Tabu Search, para un problema SOP con ejemplos de la vida real, y comparar sus resultados con los obtenidos con un método exacto.

La parametrización del algoritmo propuesto da la posibilidad de obtener mejores resultados variando los parámetros, como se describe en el Capítulo 5 en parámetros de la Búsqueda. Esto también permitió observar el comportamiento del algoritmo para los distintos problemas ejecutados.

La generación de una lista de soluciones iniciales, permitió mejorar los resultados en algunos casos, dado que tomando una solución distinta a la de menor valor de función objetivo, se aproximaba más al resultado exacto.

Los 31 problemas ejecutados, teniendo en cuenta los resultados obtenidos, comparados con la solución exacta, se dividen de la siguiente forma:

% De diferencia con la solución exacta	Cantidad de problemas	% del total de problemas
0 %	9	29 %
0-5 %	4	13 %
5-10 %	5	16 %
10-15 %	7	22,5 %
15-20 %	2	6,5 %
20-25 %	3	9,7 %
25-99 %	1	3,3 %

Algunas de las mejoras posibles a realizar al algoritmo son:

Heurística Inicial

Se podría intentar construir la Solución Inicial utilizando otra de las Heurísticas constructivas, citadas oportunamente.

Soluciones No Factibles

Resultaría interesante intentar permitir la exploración de regiones a partir de soluciones no factibles, estudiando la forma de transformar posteriormente la solución en factible.

Combinación de Movimientos

Una conocida manera de incrementar el poder de Tabu Search, es crear diferentes mecanismos de movimientos, que pueden usarse separada o simultáneamente, durante la búsqueda [11].

Una posibilidad en el contexto de nuestro problema es introducir movimientos de inserción, explicados oportunamente.

Reglas Tabu

Definir otras restricciones tabu agregando nuevas Listas Tabu.

Una posible regla es la que previene a un nodo de ocupar una determinada posición en la secuencia. Otras posibilidades son, prohibir a un determinado nodo moverse de posición, no permitir que dos nodos intercambien posiciones entre sí, prevenir que un nodo no ocupe una posición anterior en la secuencia.

Diversificación

Otros mecanismos que se pueden imponer están relacionados con la diversificación. Una estrategia frecuentemente usada es implementar una función de memoria a largo plazo que cuente la cantidad de veces que se realiza una permuta. El propósito es penalizar movimientos frecuentemente realizados, con el objetivo de dirigir la búsqueda hacia regiones no visitadas.

Criterio de Aspiración

Podría implementarse otro Criterio de Aspiración, como Aspiración Regional, descripto oportunamente.

Vecindario Restringido

Una alternativa puede ser implementar alguna de las estrategias para armar el vecindario restringido, entre las que fueron oportunamente citadas.

Tabu Search Probabilístico

Esta variante de Tabu Search permite una vez evaluado el vecindario, que la elección del mejor movimiento se realice en forma aleatoria, siguiendo una de las siguientes formas:

- Elegir aleatoriamente entre los k mejores movimientos
- Elegir aleatoriamente entre todos los movimientos con un valor dentro del α % de los mejores

La selección aleatoria puede seguir una distribución uniforme o seguir otra función de distribución de probabilidad, construida desde la evaluación asociada a cada movimiento.

Alternar Estrategias de Diversificación e Intensificación

Una alternativa posible es alternar la activación de las estrategias de Diversificación e Intensificación, estudiando la mejor medida para alternarlas.

Ejecución del Mejor Movimiento

Se podría intentar ejecutar un movimiento solo cuando este resulte admisible. El criterio para decidir si un movimiento es admisible podría ser definir un umbral de mejoría para el valor de la función objetivo, o una función de probabilidad. La iteración podría de esta manera, no resultar en un movimiento ejecutado, dado que este solo es aceptado cuando es admisible.

Enfoques Híbridos

Los enfoques híbridos son el resultado de combinar dos o más técnicas metaheurísticas, de manera tal que el procedimiento resultante sea más eficiente que la utilización de cada uno de estos.

Tabu Search Algoritmos Genéticos y Simulated Annealing son candidatos para la creación de procedimientos híbridos, [11]

BIBLIOGRAFÍA

[1] Ascheuer, N. - Escudero, L.F - Gröstschell, M. - Stoer, M."A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing)" - SIAM- Journal Optimization-1993

[2] Ascheuer, N. "Hamiltonian Path Problems in the on-line Optimization of Flexible Manufacturing Systems", Phd Thesis, Tech. Univ. Berlin, 1995

[3] Ascheuer, N.- Jünger, M.- Renuelt, G. "A Branch & Cut algorithm for the Asymmetric Hamiltonian Path Problem with Precedence Constraints" - diciembre 1997- Preprint SC 97/70, Konrad-Zuse-Zentrum fur Informatiosttechnik-Berlin

[4] Campello, R.- Maculan, N. "Algoritmos e heurísticas: Desenvolvimento e Avaliaçao de Algoritmos heurísticos", Editora da UFF, 1994

[5] Feo, T.A.- Resende, M.G.C. "Greedy Randomized Adaptive Search Procedures" - Junio 1994 (Journal of Global Optimization, 1-27 (1994)

[6] Garey, M.R.- Johnson, D.S. "Computers and intractability"- Ed. W.H. Freeman and Company- 1991

[7] Glover, F. "Tabu Search: A tutorial", Interfaces 20, pp 74-94, 1990

[8] Glover, F. "Multilevel Tabu Search and Embedded Search Neighborhoods for the Travelling Salesman Problem" - Technical Report- Junio 1991

[9] Harary- "Graph Theory"- Addison Wesley Publishing Company - 1972.

[10] Hertz, A.- Taillard, E. - de Werra, D. "Tabu Search" en "Local Search in

Combinatorial Optimization", Arrts, Lenstra(eds), Wiley,1997

[11] Laguna, M. "A guide to Implementing Tabu Search"- Investigación Operativa Vol Nro 1 PP 5-25 - Abril 1994.

[12] Laguna, M. "II Escuela de Verano Latinoamericana de Investigación Operativa- Tabu Search Tutorial"- Mendes, Brasil - 1995

[13] Reeves, C. "Modern heuristic Techniques for Combinatorial Optimization" , Backwell, 1993

[14] Ronconi, D.P. "Special strategies for Tabu Search to minimize total tardiness for the permutation flowshop problem"- 27 JAIIO- 1998

[15] Sedgewick, R. "Algoritmos en C++", Ed.Addison- Wesley/Diaz de Santos- 1995

[16] Szwarcfiter, J.L. "grafos e algoritmos computacionais"- Ed. Campus- 1988

[17] Werra, D. De- Hertz, A. "Tabu Search Techniques" - A Tutorial and an application to Neural Networks - Junio 1989

APENDICE

Este apéndice consta de dos partes:

PRIMERA PARTE : Impresión de los archivos de resultados de las ejecuciones con cada uno de los ejemplos.
SEGUNDA PARTE : Gráficos que muestran el comportamiento del algoritmo en cuanto al valor de la Función Objetivo a lo largo de las iteraciones.

PRIMERA PARTE

Tue Apr 20 11:36:21 PM
Problema: esc07
Cantidad de nodos: 7
Solución inicial:
2 0 3 1 6 5 4
Función Objetivo Solución inicial: **2625**
Mejor solución encontrada:
0 3 6 1 5 4 2

Función Objetivo mejor soluc. encontrada: **2125**
Tiempo de ejecución total: 0.000000 seg

Tiempo de ejec. mejor solución: 0.000000 seg

Número de iteraciones solicitadas: 25
Número de iteraciones realizadas: 25
Mejor iteración: 15
Cantidad de reinicializaciones: 3

Tue Apr 20 11:39:20 PM

Problema: esc11
Cantidad de nodos: 11
Solución inicial:
3 5 10 6 9 8 0 1 2 7 4

Función Objetivo Solución inicial: **2715**
Mejor solución encontrada:
3 0 8 1 4 2 5 9 6 7 10

Función objetivo mejor soluc. encontrada: **2075**
Tiempo de ejecución total: 0.000000 seg

Tiempo de ejec. mejor solución: 0.000000 seg

Número de iteraciones solicitadas: 650
Número de iteraciones realizadas: 650
Mejor iteración: 429
Cantidad de reinicializaciones: 2

Tue Apr 20 11:41:20 PM

Problema: esc12

Cantidad de nodos: 12

Solución inicial:

3 7 9 6 0 4 2 1 5 10 11 8

Función Objetivo Solución inicial: **1907**

Mejor solución encontrada:

3 7 9 8 6 0 2 4 10 1 5 11

Función objetivo mejor soluc. encontrada: **1675**

Tiempo de ejecución total: 0.000000 seg

Tiempo de ejec. mejor solución: 0.000000 seg

Número de iteraciones solicitadas: 250

Número de iteraciones realizadas: 250

Mejor iteración: 69

Cantidad de reinicializaciones: 4

Tue Apr 20 11:42:19 PM

Problema: esc14

Cantidad de nodos: 14

Solución inicial:

9 2 0 7 3 10 1 6 8 13 5 12 4 11

Función Objetivo Solución inicial: **2625**

Mejor solución encontrada:

7 0 10 3 6 8 13 1 12 5 4 11 2 9

Función objetivo mejor soluc. encontrada: **2125**

Tiempo de ejecución total: 1.000000 seg

Tiempo de ejec. mejor solución: 0.000000 seg

Número de iteraciones solicitadas: 1300

Número de iteraciones realizadas: 1300

Mejor iteración: 375

Cantidad de reinicializaciones: 4

Sun Apr 25 03:09:32 PM

Problema: esc25

Cantidad de nodos: 25

Solución inicial:

0 12 19 7 3 10 15 1 20 16 23 4 6 5 21 24 2 8 18 14 17 11 13 9 22

Función Objetivo Solución inicial: **3360**

Mejor solución encontrada:

23 20 16 8 0 12 19 7 3 10 15 1 9 18 4 6 13 5 21 22 11 24 17 14 2

Función objetivo mejor soluc. encontrada: **1747**

Tiempo de ejecución total: 30.000000 seg

Tiempo de ejec. mejor solución: 24.000000 seg

Número de iteraciones solicitadas: 10000

Número de iteraciones realizadas: 10000

Mejor iteración: 7509

Cantidad de reinicializaciones: 1

Sun Apr 25 03:57:12 PM

Problema: esc47

Cantidad de nodos: 47

Solución inicial:

41 36 46 15 24 45 42 43 19 22 18 32 13 29 17 21 20 44 35 11 30 25 27 34 26 37 12 0 31 39 33 23 1 16 40 38 28 2 3 4 7 8 10 5 14 6 9

Función Objetivo Solución inicial: **3503**

Mejor solución encontrada:

0 44 35 40 34 1 46 2 39 38 12 4 30 3 23 17 5 10 25 27 13 9 19 22 37 11 14 20 7 29 41 36 33 45 6 42 43 31 18 32 21 15 24 28 8 26 16

Función objetivo mejor soluc. encontrada: **2367**

Tiempo de ejecución total: 117.000000 *seg*

Tiempo de ejec. mejor solución: 110.000000 *seg*

Número de iteraciones solicitadas: 5000

Número de iteraciones realizadas: 5000

Mejor iteración: 4610

Cantidad de reinicializaciones: 35

Tue Apr 20 11:56:00 PM

Problema: esc63

Cantidad de nodos: 63

Solución inicial:

56 2 6 4 8 1 5 9 3 14 43 0 18 11 12 16 22 27 19 25 29 13 20 30 31 32 28 7 10 21 36 38 15 17 44 24 41 49 26 34 23 40 42 57 35 46 37 50 52 33 51 45 48 59 39 47 53 58 55 54 60 62 61

Función Objetivo Solución inicial: **68**

Mejor solución encontrada:

12 11 54 61 0 1 28 15 18 46 4 8 49 40 3 56 7 13 19 2 34 6 37 50 38 42 5 9 25 21 36 22 27 14 51 52 30 44 26 57 35 17 10 16 33 23 43 20 32 31 59 45 24 41 39 47 53 58 55 48 29 60 62

Función objetivo mejor soluc. encontrada: **62**

Tiempo de ejecución total: 156.000000 *seg*

Tiempo de ejec. mejor solución: 8.000000 *seg*

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 27

Cantidad de reinicializaciones: 1

Tue Apr 20 11:59:36 PM

Problema: esc78

Cantidad de nodos: 78

Solución inicial:

0 2 1 9 6 4 13 7 17 10 21 16 5 14 31 3 11 8 20 12 22 37 18 59 32 38 24 60 42 66 45 76 46 15 25 28 75 19 29 23 33 55 39 26 35 27 41 64 40 30 44 72 62 73 52 47 34 50 56 36 53 58 43 65 48 67 49 70 51 74 54 71 57 61 63 68 69 77

Función Objetivo Solución inicial: **22600**

Mejor solución encontrada:

0 2 1 9 6 7 19 10 17 3 31 11 23 22 37 4 15 59 42 8 44 72 39 12 24 66 40 43 20 5 33 55 18 34 74 56 69 38 32 54 14 21 62 68 29 27 25 28 16 30 35 13 26 45 52 73 77 50 76 36 41 58 61 65 48 53 49 70 51 67 60 71 57 75 63 47 64 46

Función objetivo mejor soluc. encontrada: **18640**

Tiempo de ejecución total: 166.000000 *seg*

Tiempo de ejec. mejor solución: 115.000000 *seg*

Número de iteraciones solicitadas: 600

Número de iteraciones realizadas: 600

Mejor iteración: 414

Cantidad de reinicializaciones: 1

Wed Apr 21 12:01:10 AM

Problema: esc98

Cantidad de nodos: 98

Solución inicial:

93 2 9 16 23 30 37 44 51 58 65 72 79 86 0 7 14 21 28 35 42 49 56 63 70 77 84 91 3 10 17 24 31 38 45 52 59 66 73 80 87 94 1 6 8 13 15 20 22 27 29 34 36 41 43 48 50 55 57 62 64 69 71 76 78 83 85 90 92 97 5 12 19 26 33 40 47 54 61 68 75 82 89 96 4 11 18 25 32 39 46 53 60 67 74 81 88 95

Función Objetivo Solución inicial: **2625**

Mejor solución encontrada:

93 2 9 16 23 30 37 44 51 58 65 72 79 86 0 7 14 21 28 35 42 49 56 63 70 77 84 91 3 10 17 24 31 38 45 52 59 66 73 80 87 94 1 6 8 13 15 20 22 27 29 34 36 41 43 48 50 55 57 62 64 69 71 76 78 83 85 90 92 97 5 12 19 26 33 40 47 54 61 68 75 82 89 96 4 11 18 25 32 39 46 53 60 67 74 81 88 95

Función objetivo mejor soluc. encontrada: **2625**

Tiempo de ejecución total: 53.000000 seg

Tiempo de ejec. mejor solución: 0.000000 seg

Número de iteraciones solicitadas: 100

Número de iteraciones realizadas: 100

Mejor iteración: 1

Cantidad de reinicializaciones: 4

Sat Apr 17 06:00:34 PM

Problema: rbg019a

Cantidad de nodos: 19

Solución inicial:

3 2 0 1 6 5 4 9 7 8 12 11 10 13 15 16 14 18 17

Función Objetivo Solución inicial: **198**

Mejor solución encontrada:

3 2 0 1 6 5 4 9 7 8 12 11 10 13 15 16 14 18 17

Función objetivo mejor soluc. encontrada: **198**

Tiempo de ejecución total: 0.000000 seg

Tiempo de ejec. mejor solución: 0.000000 seg

Número de iteraciones solicitadas: 1

Número de iteraciones realizadas: 1

Mejor iteración: 0

Cantidad de reinicializaciones: 0

Wed Apr 21 12:05:51 AM

Problema: rbg019b

Cantidad de nodos: 19

Solución inicial:

1 0 2 4 3 7 5 6 8 11 10 12 13 9 14 15 16 17 18

Función Objetivo Solución inicial: **243**

Mejor solución encontrada:

1 0 2 5 7 4 6 3 8 13 9 14 12 11 10 15 16 17 18

Función objetivo mejor soluc. encontrada: **199**

Tiempo de ejecución total: 3.000000 seg

Tiempo de ejec. mejor solución: 1.000000 seg

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 119

Cantidad de reinicializaciones: 4

Sun Apr 25 06:07:57 PM

Problema: rbg021a

Cantidad de nodos: 19

Solución inicial:

1 0 4 2 5 7 3 6 11 12 8 9 10 17 16 15 18 13 14

Función Objetivo Solución inicial: **236**

Mejor solución encontrada:

1 0 4 8 7 3 6 5 2 11 14 13 12 10 9 16 15 18 17

Función objetivo mejor soluc. encontrada: **158**

Tiempo de ejecución total: 6.000000 seg

Tiempo de ejec. mejor solución: 2.000000 seg

Número de iteraciones solicitadas: 1500

Número de iteraciones realizadas: 1500

Mejor iteración: 564

Cantidad de reinicializaciones: 0

Wed Apr 21 12:08:02 AM

Problema: rbg023a

Cantidad de nodos: 21

Solución inicial:

1 0 4 2 5 7 3 6 11 12 8 9 10 17 16 15 18 13 14 19 20

Función Objetivo Solución inicial: **273**

Mejor solución encontrada:

1 0 4 8 7 3 6 5 2 11 14 13 12 10 9 16 15 19 18 17 20

Función objetivo mejor soluc. encontrada: **155**

Tiempo de ejecución total: 4.000000 seg

Tiempo de ejec. mejor solución: 2.000000 seg

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 286

Cantidad de reinicializaciones: 4

Wed Apr 21 12:10:06 AM

Problema: rbg029a

Cantidad de nodos: 27

Solución inicial:

0 8 13 1 10 3 20 21 22 5 25 2 15 16 4 6 7 24 12 9 14 11 26 18 17 23 19

Función Objetivo Solución inicial: **270**

Mejor solución encontrada:

0 2 1 24 25 5 6 7 12 9 3 4 8 13 15 20 22 21 23 10 11 26 14 16 17 18 19

Función objetivo mejor soluc. encontrada: **221**

Tiempo de ejecución total: 47.000000 seg

Tiempo de ejec. mejor solución: 44.000000 seg

Número de iteraciones solicitadas: 5000

Número de iteraciones realizadas: 5000

Mejor iteración: 4730

Cantidad de reinicializaciones: 3

Wed Apr 21 12:12:46 AM

Problema: rbg048a

Cantidad de nodos: 48

Solución inicial:

46 0 2 8 27 24 47 17 20 3 1 30 5 11 28 13 32 33 9 15 16 21 12 18 26 19 31 23 25 10 4 22 6 7 14 35 34 29 38 36 44 40 37 43 39 41 42 45

Función Objetivo Solución inicial: **483**

Mejor solución encontrada:

46 0 27 47 8 24 16 21 12 4 1 22 5 29 14 28 32 13 11 26 10 19 33 9 17 25 15 2 23 20 3 30 6 7 18 34 31 36 35 41 38 45 40 37 44 43 39 42

Función objetivo mejor soluc. encontrada: **389**

Tiempo de ejecución total: 126.000000 seg

Tiempo de ejec. mejor solución: 68.000000 seg

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 269

Cantidad de reinicializaciones: 4

Wed Apr 21 05:15:09 PM

Problema: rbg049a

Cantidad de nodos: 49

Solución inicial:

1 0 8 2 9 25 26 16 32 22 3 17 30 18 5 4 23 11 20 29 13 6 12 24 7 14 15 21 34 19 31 10 27 28 39 33 42 46 44 36 35 37 38 40 41 47 45 43 48

Función Objetivo Solución inicial: **473**

Mejor solución encontrada:

1 0 8 9 2 26 25 4 32 22 3 17 30 18 5 16 10 23 15 11 20 14 34 24 6 12 21 29 13 31 19 7 28 27 39 33 42 46 44 37 43 41 35 40 36 47 45 38 48

Función objetivo mejor soluc. encontrada: **407**

Tiempo de ejecución total: 321.000000 *seg*

Tiempo de ejec. mejor solución: 128.000000 *seg*

Número de iteraciones solicitadas: 1000

Número de iteraciones realizadas: 1000

Mejor iteración: 427

Cantidad de reinicializaciones: 4

Sun Apr 25 07:47:06 PM

Problema: rbg050a

Cantidad de nodos: 50

Solución inicial:

2 0 1 33 24 6 28 7 47 31 4 3 17 8 44 21 34 5 29 10 22 23 25 18 35 32 9 16 20 11 41 12 26 13 45 27 15 49 14 19 36 37 30 38 46 42 39 40 43 48

Función Objetivo Solución inicial: **500**

Mejor solución encontrada:

2 1 0 18 3 19 6 7 47 31 4 14 27 8 44 21 5 29 22 23 24 33 25 10 49 17 9 16 20 11 13 28 12 41 32 34 15 26 37 45 36 35 30 38 39 43 42 48 46 40

Función objetivo mejor soluc. encontrada: **447**

Tiempo de ejecución total: 128.000000 *seg*

Tiempo de ejec. mejor solución: 87.000000 *seg*

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 334

Cantidad de reinicializaciones: 2

Sun Apr 25 08:17:56 PM

Problema: rbg050b

Cantidad de nodos: 50

Solución inicial:

1 0 8 7 2 16 30 4 17 10 26 19 35 15 34 27 13 24 14 18 21 25 22 11 20 12 28 9 29 23 3 5 6 33 32 31 38 48 43 44 49 47 36 37 42 39 45 41 40 46

Función Objetivo Solución inicial: **558**

Mejor solución encontrada:

1 0 8 7 2 3 30 4 17 34 10 26 6 12 28 22 13 24 14 20 25 18 27 19 5 16 23 9 29 11 21 35 15 32 31 38 33 48 44 43 49 47 42 37 36 46 45 41 40 39

Función objetivo mejor soluc. encontrada: **421**

Tiempo de ejecución total: 291.000000 *seg*

Tiempo de ejec. mejor solución: 121.000000 *seg*

Número de iteraciones solicitadas: 600

Número de iteraciones realizadas: 600

Mejor iteración: 148

Cantidad de reinicializaciones: 4

Sun Apr 25 08:36:25 PM

Problema: rbg050c

Cantidad de nodos: 50

Solución inicial:

1 0 10 2 19 34 25 4 32 15 14 27 16 17 3 23 22 20 40 28 6 5 42 41 37 7 26 8 18 29 9 31 45 33 39 11 12 13 24 21 35 47 48 30 46 36 49 44 38 43

Función Objetivo Solución inicial: **558**

Mejor solución encontrada:

1 0 10 2 19 25 4 40 14 27 5 42 31 17 3 23 22 15 45 28 7 39 12 41 26 9 37 8 18 29 6 33 32 20 16 13 34 11 24 21 38 48 47 30 46 43 36 49 35 44

Función objetivo mejor soluc. encontrada: **499**

Tiempo de ejecución total: 332.000000 seg

Tiempo de ejec. mejor solución: 115.000000 seg

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 174

Cantidad de reinicializaciones: 4

Wed Apr 21 05:38:07 PM

Problema: rbg068a

Cantidad de nodos: 66

Solución inicial:

1 0 4 2 5 7 3 6 11 12 8 9 10 17 16 15 18 13 14 19 20 24 22 23 21 26 28 25 27 30 31 29 32 33 34 35 38 44 37 40 39 36 42 41 43 45 46 47 48 49 50 51 52 55 54 56 53 57 61 64 63 65 58 59 60 62

Función Objetivo Solución inicial: **917**

Mejor solución encontrada:

1 0 4 6 8 7 3 2 11 12 5 10 9 18 16 14 13 15 19 17 20 21 23 22 27 26 24 28 25 31 30 29 35 32 34 33 38 44 40 43 39 36 42 41 37 46 47 45 51 50 48 49 52 55 54 56 53 61 57 60 59 65 64 63 62 58

Función objetivo mejor soluc. encontrada: **738**

Tiempo de ejecución total: 170.000000 seg

Tiempo de ejec. mejor solución: 86.000000 seg

Número de iteraciones solicitadas: 100

Número de iteraciones realizadas: 100

Mejor iteración: 56

Cantidad de reinicializaciones: 3

Wed Apr 21 05:51:46 PM

Problema: rbg088a

Cantidad de nodos: 88

Solución inicial:

5 0 1 2 3 4 6 7 8 11 9 10 12 13 15 14 18 23 16 17 19 20 24 21 25 22 32 26 27 28 38 30 29 31 40 41 33 43 49 34 48 35 36 37 39 42 44 51 45 46 54 53 47 50 58 57 52 61 60 63 55 59 56 62 66 67 68 70 64 65 69 72 75 71 78 79 76 73 77 74 83 84 81 80 82 85 86 87

Función Objetivo Solución inicial: **1370**

Mejor solución encontrada:

5 0 1 2 3 4 6 7 8 9 11 13 12 10 15 14 18 20 16 17 19 23 24 21 25 22 32 27 26 28 38 30 29 31 49 35 33 43 40 34 48 41 36 37 39 42 46 51 45 44 54 53 47 50 58 57 52 61 60 56 55 59 63 64 62 68 67 70 66 65 69 71 75 72 74 77 76 73 78 79 83 84 85 80 82 81 87 86

Función objetivo mejor soluc. encontrada: **1240**

Tiempo de ejecución total: 278.000000 *seg*

Tiempo de ejec. mejor solución: 105.000000 *seg*

Número de iteraciones solicitadas: 50

Número de iteraciones realizadas: 50

Mejor iteración: 17

Cantidad de reinicializaciones: 3

Sat Apr 24 11:08:48 PM

Problema: rbg092a

Cantidad de nodos: 92

Solución inicial:

2 1 0 3 4 5 6 7 10 8 9 11 12 13 19 14 22 24 16 15 18 26 20 17 27 25 21 28 29 23 36 30 31 32 39 33 34 35 40 37 42 41 38 51 50 46 45 47 44 43 48 49 54 59 56 52 53 55 57 58 63 61 60 67 69 75 62 64 83 65 74 66 77 68 80 70 81 71 72 73 76 87 85 86 78 88 79 82 84 91 90 89

Función Objetivo Solución inicial: **1246**

Mejor solución encontrada:

2 1 0 3 4 5 10 7 6 9 8 11 13 12 20 14 22 19 16 15 23 25 18 17 27 26 21 30 29 24 32 28 33 31 39 34 35 36 40 37 42 41 38 51 50 46 45 47 44 43 48 49 59 58 56 54 53 57 55 52 63 61 64 62 60 65 67 73 68 72 74 66 69 76 80 75 71 77 83 78 70 87 85 81 84 82 88 86 79 91 90 89

Función objetivo mejor soluc. encontrada: **1084**

Tiempo de ejecución total: 3094.000000 seg

Tiempo de ejec. mejor solución: 2336.000000 seg

Número de iteraciones solicitadas: 250

Número de iteraciones realizadas: 250

Mejor iteración: 186

Cantidad de reinicializaciones: 1

Wed Apr 21 08:07:49 AM

Problema: rbg094a

Cantidad de nodos: 94

Solución inicial:

6 5 3 0 9 1 11 12 17 19 2 15 4 16 7 8 10 20 21 13 22 14 18 23 26 25 24 27 28 29 30 31 35 32 33 34 38 36 37 39 40 41 42 43 44 45 46 47 50 48 49 51 52 54 53 57 62 55 56 58 59 63 60 64 61 71 65 66 67 77 69 68 70 79 80 72 82 88 73 87 74 75 76 78 81 83 90 84 85 93 92 86 89 91

Función Objetivo Solución inicial: **1548**

Mejor solución encontrada:

0 19 3 1 5 6 11 4 17 9 2 10 12 13 7 15 8 20 16 14 23 22 21 18 26 25 24 28 29 27 30 31 34 33 32 36 35 41 37 38 44 42 43 39 40 47 45 46 49 52 51 48 50 58 53 57 55 56 54 59 60 62 63 64 61 71 66 65 67 77 69 68 70 88 74 72 82 79 73 87 80 75 76 78 81 83 91 85 84 93 92 86 89 90

Función objetivo mejor soluc. encontrada: **1398**

Tiempo de ejecución total: 27774.000000 seg

Tiempo de ejec. mejor solución: 14845.000000 seg

Número de iteraciones solicitadas: 1700

Número de iteraciones realizadas: 1700

Mejor iteración: 908

Cantidad de reinicializaciones: 4

Tue Apr 27 11:13:42 AM

Problema: rbg105a

Cantidad de nodos: 105

Solución inicial:

1 10 4 7 5 6 8 3 2 0 9 17 12 11 18 19 20 16 21 13 14 15 27 28 24 23 30 25 26 22 29 34 33 37 31 32 41 35 46 36 40 44 43 45 42 38 47 51 48 39 54 53 55 52 56 49 50 59 62 60 57 58 61 63 64 65 66 67 68 69 74 75 72 70 77 71 76 73 78 79 87 80 89 91 81 88 82 83 84 85 94 86 90 97 96 92 93 95 98 99 100 102 104 103 101

Función Objetivo Solución inicial: **1357**

Mejor solución encontrada:

21 10 4 7 5 6 8 3 2 0 1 17 12 11 18 19 20 16 15 13 14 9 22 28 29 23 35 25 24 32 26 34 33 27 31 30 37 41 45 46 42 44 43 36 40 39 38 48 49 47 53 50 51 52 56 55 57 54 62 60 61 59 58 64 63 66 67 65 68 69 75 73 72 70 77 71 74 76 80 78 81 82 83 79 89 88 87 84 86 85 94 91 90 97 96 92 93 95 98 101 100 102 104 103 99

Función objetivo mejor soluc. encontrada: **1213**

Tiempo de ejecución total: 11905.000000 seg

Tiempo de ejec. mejor solución: 7858.000000 seg

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 363

Cantidad de reinicializaciones: 2

Tue Apr 27 04:56:59 AM

Problema: rbg109a

Cantidad de nodos: 109

Solución inicial:

1 5 3 2 0 7 4 6 9 10 8 19 16 11 18 15 21 17 12 13 20 14 23 24 22 25 26 27 28 29 30 32 31 35 34 33 37 36 40 43 42 44 38 39 47 41 51 52 45 49 46 50 65 55 56 58 48 53 57 60 54 63 59 71 68 61 70 62 69 64 73 72 74 66 67 78 75 79 80 81 77 83 82 76 84 86 85 87 92 90 88 91 89 95 93 102 97 96 100 99 94 98 103 101 104 105 107 108 106

Función Objetivo Solución inicial: **1478**

Mejor solución encontrada:

0 5 3 2 1 7 6 4 10 9 8 19 12 11 18 15 21 17 14 13 20 16 23 22 24 26 25 29 28 27 30 32 31 35 34 38 33 39 40 36 42 44 43 37 47 41 51 52 45 49 46 48 53 58 50 54 65 55 56 57 70 63 59 71 68 61 60 62 64 67 69 72 74 66 73 76 75 79 80 78 77 83 82 81 84 87 85 86 92 90 88 91 89 95 93 97 98 96 100 99 94 101 102 107 108 105 104 106 103

Función objetivo mejor soluc. encontrada: **1165**

Tiempo de ejecución total: 17785.000000 seg

Tiempo de ejec. mejor solución: 7994.000000 seg

Número de iteraciones solicitadas: 500

Número de iteraciones realizadas: 500

Mejor iteración: 237

Cantidad de reinicializaciones: 5

Wed Apr 21 01:07:42 PM

Problema: rbg126a

Cantidad de nodos: 124

Solución inicial:

0 40 2 1 3 6 4 7 5 8 9 10 12 11 15 13 14 19 16 17 18 21 20 23 22 25 26 24 27 28 31 29 30 34 32 33 35 38 37 36 39 42 41 57 43 46 49 47 86 87 58 83 74 72 79 48 53 65 61 54 45 44 59 50 51 52 62 55 56 60 64 66 63 67 68 69 70 71 73 78 75 76 77 81 92 80 85 82 84 88 89 90 91 96 93 94 95 97 98 99 100 101 103 104 102 106 107 105 109 108 111 112 113 110 117 119 115 116 120 114 118 121 122 123

Función Objetivo Solución inicial: **1760**

Mejor solución encontrada:

0 40 2 1 3 5 6 4 7 8 9 11 12 10 15 13 14 17 16 19 18 21 22 20 25 24 26 23 28 27 30 29 32 34 31 33 35 38 36 37 39 44 65 79 41 42 58 47 86 87 49 72 57 83 61 48 53 54 74 43 45 46 51 59 50 52 55 62 56 60 64 66 63 67 69 68 70 71 73 75 77 76 78 80 81 92 85 82 84 88 89 90 91 96 93 94 95 97 98 99 100 101 103 104 102 106 107 105 109 108 111 112 113 110 114 118 115 116 120 117 119 121 122 123

Función objetivo mejor soluc. encontrada: **1584**

Tiempo de ejecución total: 11931.000000 *seg*

Tiempo de ejec. mejor solución: 8571.000000 *seg*

Número de iteraciones solicitadas: 200

Número de iteraciones realizadas: 200

Mejor iteración: 149

Cantidad de reinicializaciones: 0

Tue Apr 20 08:30:40 AM

Problema: rbg148a

Cantidad de nodos: 146

Solución inicial:

1 0 2 27 17 3 39 44 41 5 13 10 11 12 14 15 6 4 35 16 23 25 24 33 26 8 9 38 43 7 18 22 34 29 37 20 21 19 32 28 30 31 36 40 42 48 47 45 46 49 50 54 53 51 52 59 55 60 57 56 58 61 63 64 62 66 68 65 67 69 71 77 70 78 80 74 72 75 81 83 79 73 76 87 86 82 84 92 91 85 97 94 88 98 89 90 93 100 95 104 96 99 108 101 106 111 102 110 103 105 107 109 116 112 119 113 114 115 121 117 123 118 125 124 120 127 122 129 126 128 132 133 135 137 130 131 141 134 139 140 136 138 142 143 144 145

Función Objetivo Solución inicial: **1839**

Mejor solución encontrada:

1 0 2 26 17 6 43 44 41 5 11 13 12 10 14 15 3 4 35 16 23 25 24 33 27 8 9 38 39 7 18 22 34 29 37 20 21 36 19 28 30 31 32 46 42 48 47 40 45 49 50 54 53 51 52 57 56 60 59 58 55 61 63 64 62 66 68 67 65 69 70 77 71 78 80 74 72 75 81 83 79 73 76 87 86 84 82 92 91 85 97 94 90 89 88 99 95 100 98 93 103 96 102 101 106 104 109 110 105 111 108 107 115 112 119 116 118 117 114 113 124 120 125 121 128 123 122 129 127 126 132 133 135 137 131 130 141 134 143 139 140 136 138 144 142 145

Función objetivo mejor soluc. encontrada: **1624**

Tiempo de ejecución total: 31791.000000 seg

Tiempo de ejec. mejor solución: 15781.000000 seg

Número de iteraciones solicitadas: 200

Número de iteraciones realizadas: 200

Mejor iteración: 101

Cantidad de reinicializaciones: 4

Tue Apr 20 06:50:48 PM

Problema: rbg161a

Cantidad de nodos: 159

Solución inicial:

2 1 0 3 5 6 20 4 8 9 7 11 10 13 14 15 12 16 17 18 19 31 32 21 24 22 23 26 27 39 25 38 29 37 28 30 34 33 36 35 41 42 40 43 44 45 46 47 48 49 50 51 52 53 54 56 58 55 61 57 60 59 62 63 64 65 66 67 68 69 71 72 70 74 75 73 77 76 79 99 78 85 87 83 84 80 81 82 86 89 90 98 88 94 91 95 92 93 103 96 102 97 105 100 101 104 107 106 108 110 109 111 112 114 113 115 116 118 117 119 120 121 125 124 122 123 126 130 128 129 127 132 131 134 133 135 137 136 139 138 142 140 141 145 143 144 148 152 146 147 150 149 155 151 153 154 157 158 156

Función Objetivo Solución inicial: **2337**

Mejor solución encontrada:

0 1 2 3 5 6 20 4 8 7 9 11 10 14 12 15 13 16 18 17 19 31 39 21 24 22 23 37 38 27 25 28 26 29 32 30 36 33 35 34 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 57 58 55 61 56 60 59 62 64 63 65 66 67 68 69 71 72 70 74 75 73 77 76 79 99 78 85 87 83 84 80 81 82 86 89 88 98 90 91 94 95 92 93 103 96 102 97 105 100 101 104 107 106 108 109 110 111 112 114 113 115 116 118 117 119 120 121 125 124 122 123 126 127 128 129 130 132 131 134 133 135 137 136 139 138 142 140 141 145 143 144 148 152 146 147 150 149 155 151 153 154 157 158 156

Función objetivo mejor soluc. encontrada: **2224**

Tiempo de ejecución total: 10926.000000 seg

Tiempo de ejec. mejor solución: 4263.000000 seg

Número de iteraciones solicitadas: 70

Número de iteraciones realizadas: 70

Mejor iteración: 23

Cantidad de reinicializaciones: 4

Mon Apr 26 06:36:37 AM

Problema: rbg174a

Cantidad de nodos: 174

Solución inicial:

1 0 2 3 8 6 4 7 5 11 9 18 13 12 16 15 10 14 19 17 20 25 28 23 24 21 27 26 22 29 30 31 32 33 34 35 36 41 39 40 37 38 46 42 48 47 43 44 45 55 50 49 51 54 52 71 58 53 70 56 57 59 60 61 62 63 64 65 73 66 74 67 78 77 68 69 87 80 72 75 84 83 76 88 85 79 81 82 94 93 86 89 90 91 92 107 101 95 96 108 97 98 99 100 102 109 103 104 105 106 110 111 112 113 114 120 117 115 116 121 118 119 122 123 124 127 125 129 143 130 126 131 135 132 128 136 139 137 151 141 134 133 144 138 152 140 147 142 149 150 145 153 146 155 148 167 158 161 156 162 160 157 159 163 166 154 169 168 165 164 171 170 173 172

Función Objetivo Solución inicial: **2440**

Mejor solución encontrada:

1 0 3 2 8 6 4 7 5 11 9 13 14 12 16 15 17 18 10 21 19 23 22 20 25 28 27 26 24 29 30 31 32 33 36 35 34 41 38 40 39 37 43 42 48 47 45 44 46 55 50 52 51 54 49 71 58 53 56 70 57 59 60 61 62 63 64 65 73 66 68 67 78 77 69 74 87 80 72 75 84 83 76 88 85 82 81 79 94 93 92 89 90 91 86 107 101 95 96 105 102 98 99 100 97 109 103 104 108 106 110 113 112 111 114 120 115 117 116 122 118 123 121 119 124 127 125 129 143 130 126 131 135 132 128 136 141 137 151 139 144 133 134 138 152 140 147 150 149 142 145 153 146 155 148 167 158 161 156 162 154 160 159 163 166 157 169 168 165 164 171 170 173 172

Función objetivo mejor soluc. encontrada: **2218**

Tiempo de ejecución total: 25091.000000 *seg*

Tiempo de ejec. mejor solución: 12099.000000 *seg*

Número de iteraciones solicitadas: 70

Número de iteraciones realizadas: 70

Mejor iteración: 35

Cantidad de reinicializaciones: 5

Thu Apr 22 08:24:07 AM

Problema: rbg190a

Cantidad de nodos: 188

Solución inicial:

2 1 0 3 5 6 20 4 8 9 7 11 10 13 14 15 12 16 17 18 19 31 32 21 24 22 23 26 27 39 25 38 29 61 37 28 30 34 33 36 35 41 42 40 43 44 45 46 47 48 49 50 51 52 53 54 56 58 55 59 57 60 100 99 91 62 66 95 103 63 64 98 68 73 74 65 88 67 71 72 70 76 69 85 108 75 90 94 87 83 77 81 107 78 84 79 97 106 80 82 112 96 86 104 89 93 92 110 102 126 105 101 116 128 113 114 131 111 122 115 109 117 118 119 121 120 123 124 125 127 129 130 133 132 135 134 138 136 137 141 139 140 144 148 142 143 146 145 149 147 152 151 150 155 153 160 154 158 156 157 163 159 162 161 165 164 169 179 166 182 167 168 170 171 178 172 173 174 175 176 177 180 181 183 184 185 186 187

Función Objetivo Solución inicial: **2732**

Mejor solución encontrada:

0 1 2 3 5 6 20 4 8 7 9 11 10 13 12 15 14 18 17 16 19 31 32 21 24 22 23 61 39 27 25 26 29 28 37 38 30 36 33 35 34 40 41 42 43 45 44 47 46 49 48 50 51 52 53 54 56 58 55 59 57 60 100 99 78 62 66 95 103 63 64 98 68 73 74 65 88 69 70 72 71 76 67 85 108 75 97 106 87 83 77 81 107 94 84 79 90 91 80 82 112 96 86 104 89 93 92 101 102 109 105 115 116 128 113 114 126 131 110 111 122 117 118 119 120 121 123 124 125 127 129 130 133 132 135 134 138 136 137 141 139 140 144 148 142 143 146 145 149 147 152 151 150 155 153 160 154 156 158 157 163 159 162 161 165 164 169 167 166 182 179 168 170 172 178 171 173 174 175 176 177 180 181 183 184 185 187 186

Función objetivo mejor soluc. encontrada: **2585**

Tiempo de ejecución total: 49975.000000 seg

Tiempo de ejec. mejor solución: 28370.000000 seg

Número de iteraciones solicitadas: 100

Número de iteraciones realizadas: 100

Mejor iteración: 59

Cantidad de reinicializaciones: 3

Mon Apr 19 06:39:59 PM

Problema: rbg285a2

Cantidad de nodos: 283

Solución inicial:

1 70 0 2 3 5 4 7 6 8 9 10 13 11 12 14 15 22 16 17 18 23 19 21 20 25 24 27 28 26 29 32 31 30 33 36 34 37 35 38 39 40 42 41 45 43 44 49 46 47 48 51 50 53 52 55 56 54 57 58 61 59 60 64 62 63 65 68 67 66 69 72 71 87 73 76 79 77 116 117 88 113 104 102 109 78 83 95 91 84 75 74 89 80 81 82 92 85 86 90 94 96 93 97 98 99 100 101 103 108 105 106 107 111 122 110 115 112 114 118 119 120 121 126 123 124 125 127 128 129 130 131 133 134 132 136 137 135 139 138 141 161 140 147 149 145 146 142 143 144 148 151 152 160 150 156 153 157 154 155 165 158 164 159 167 162 163 166 169 168 170 172 171 173 174 176 175 177 178 180 179 181 182 183 187 186 184 185 188 192 190 191 189 194 193 196 195 197 199 198 201 200 204 202 203 207 205 206 210 214 208 209 212 211 217 213 215 216 219 220 218 224 226 221 222 223 229 225 227 228 245 246 234 230 232 231 233 240 238 235 236 237 239 241 247 242 243 244 248 249 250 251 252 253 254 255 256 259 257 260 258 262 263 264 261 266 268 269 267 265 270 273 271 272 275 274 276 277 278 282 279 280 281

Función Objetivo Solución inicial: **4268**

Mejor solución encontrada:

0 3 70 2 1 5 4 7 6 8 10 9 12 13 11 14 15 22 18 17 16 23 19 21 20 25 24 27 28 26 29 32 31 30 33 36 34 37 35 38 39 41 42 40 45 43 44 46 49 47 48 51 52 50 55 54 56 53 58 57 59 60 62 64 61 63 65 68 66 67 69 74 95 87 71 72 79 77 116 117 88 113 104 102 109 78 83 84 91 73 75 76 81 89 80 82 85 92 86 90 94 96 93 97 98 99 100 101 103 105 107 106 108 110 111 122 115 112 114 118 119 120 121 126 123 124 125 127 128 129 130 131 133 134 132 136 137 135 139 138 141 161 140 147 149 145 146 142 143 144 148 151 150 160 152 153 156 157 154 155 165 158 164 159 167 162 163 166 169 168 170 171 172 173 174 176 175 177 178 180 179 181 182 183 187 186 184 185 188 189 190 191 192 194 193 196 195 197 199 198 201 200 204 202 203 207 205 206 210 214 208 209 212 211 217 213

215 216 219 220 218 224 226 221 222 223 229 225 227 228 245 233 234 230 246 231 232 240 238 235 236 237 239 241 247 242 243 244 248 249 250 251 253 252 254 255 256 259 257 260 258 262 263 264 261 266 268 265 269 267 270 273 272 271 275 274 277 276 278 282 279 280 281

Función objetivo mejor soluc. encontrada: **3887**

Tiempo de ejecución total: 169766.000000 *seg*

Tiempo de ejec. mejor solución: 142639.000000 *seg*

Número de iteraciones solicitadas: 50

Número de iteraciones realizadas: 50

Mejor iteración: 41

Cantidad de reinicializaciones: 0

SEGUNDA PARTE

El estudio de los resultados se dividió en tres partes:

- Problemas que arrojaron resultados exactos
- Problemas que arrojaron resultados aproximados (no más del 10 % de la solución exacta)
- Problemas que arrojaron resultados distantes de la solución exacta.

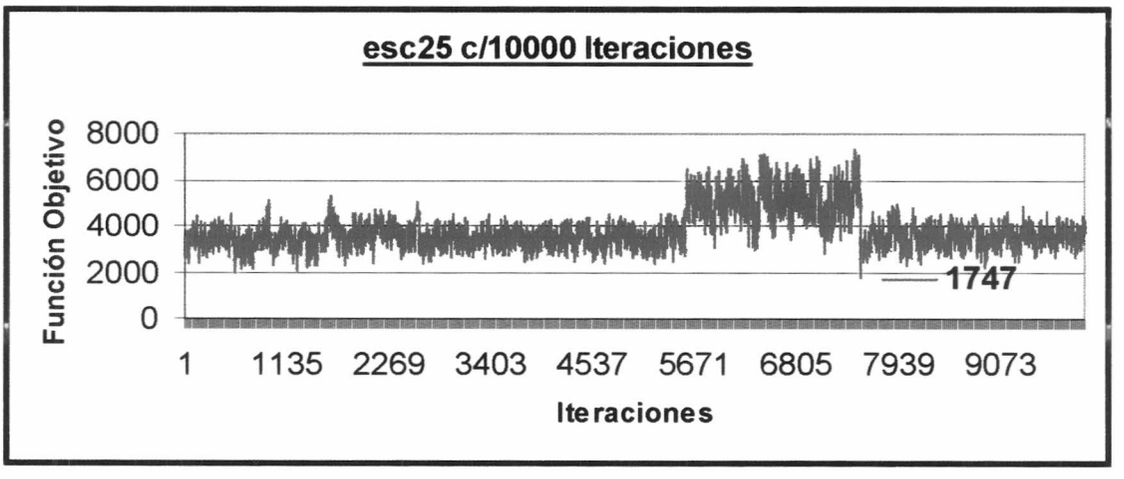
En todos los casos las pruebas se realizaron variando y combinando los parámetros ajustables de la búsqueda, y tamaños de ciertas estructuras importantes para la misma.

A continuación se presentan los gráficos y las tablas de valores de cada iteración, correspondientes al comportamiento del algoritmo para cada problema estudiado.

En todos los problemas se tomaron para cada iteración los valores de la Función Objetivo y el valor del reloj. El número de iteraciones varia en cada problema, dado que en algunos casos fue necesario aumentar esta cantidad, con el objetivo de encontrar la mejor solución.

Estudio de problemas cuyo resultado resultó distante de la solución exacta:

- Se observó que para el problema esc25 aumentar la cantidad de iteraciones totales mejoraba el valor de la función objetivo obtenida.
  La mejor solución obtenida para este problema se obtuvo partiendo de la solución inicial número catorce de la lista de mejores soluciónes iniciales, con 10000 iteraciones.
  El comportamiento del algoritmo para este problema resultó:



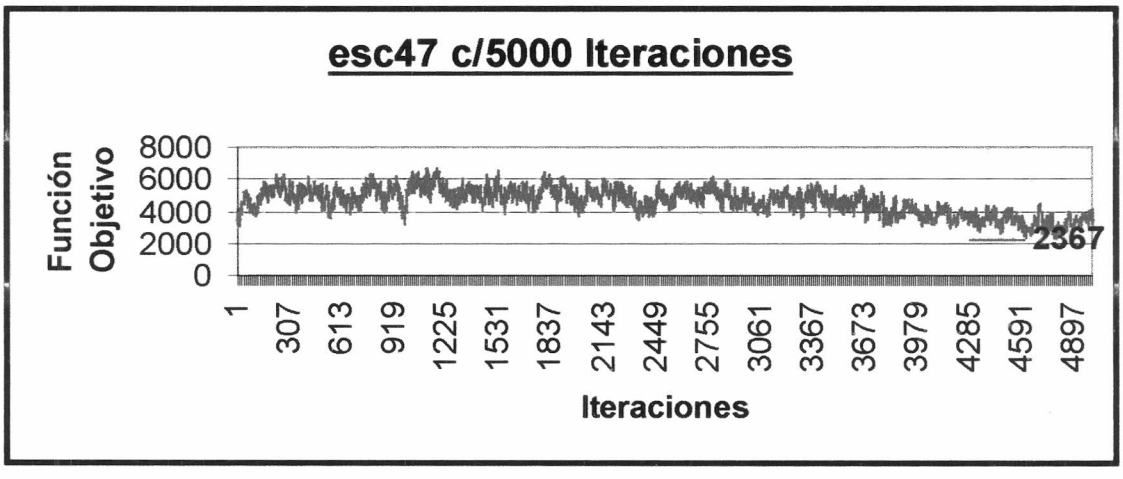
ITER	FO	CPU
1	3360	00:00:00
1135	4200	00:00:03
2569	4235	00:00:09
3403	3674	00:00:11
4537	2573	00:00:15
5671	5514	00:00:18
6805	5353	00:00:21
7509	1747	00:00:24
7939	3765	00:00:25
9073	3414	00:00:28
10000	4105	00:00:30

Estudio de problemas cuyo resultado resultó distante de la solución exacta:

- Para el problema esc47 dado que la solución obtenida en la mayoría de las pruebas no era buena, se aumentó la diversificación y la intensificación durante la búsqueda.

  La mejor solución se obtuvo partiendo de la solución inicial número siete de la lista de mejores soluciones iniciales, con 5000 iteraciones.

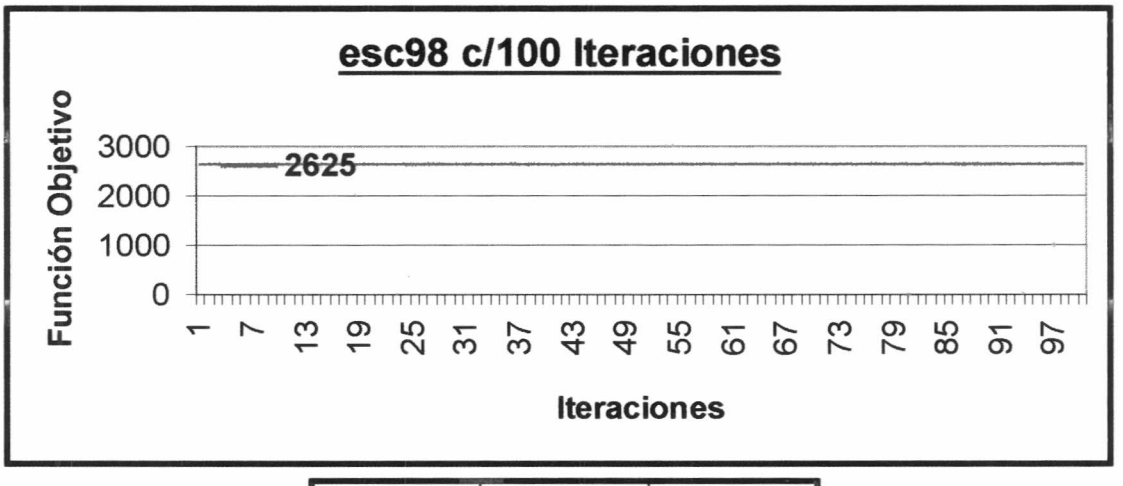
  El comportamiento del algoritmo para este problema resultó:



ITER	FO	CPU
1	3503	00:00:00
307	5586	00:00:11
613	5277	00:00:18
919	5596	00:00:25
1225	5220	00:00:31
1531	5676	00:00:35
1837	5217	00:00:40
2143	5364	00:00:45
2449	4957	00:00:52
2755	5401	00:01:01
3061	4532	00:01:11
3367	4747	00:01:21
3673	4615	00:01:30
3979	4087	00:01:39
4285	3738	00:01:44
4591	3288	00:01:50
4610	2367	00:01:50
4897	3192	00:01:55
5000	3234	00:01:57

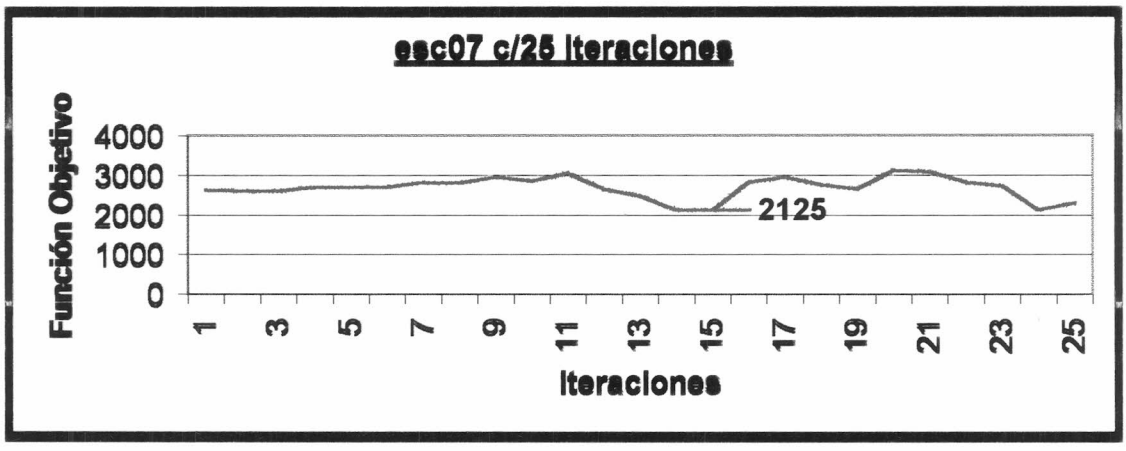
Estudio de problemas cuyo resultado resultó distante de la solución exacta:

- El problema esc98 en la mayoría de las pruebas terminó encontrando su mejor solución en la primera iteración, esto es, la solución inicial. Se probó comenzar con todas las posibles soluciones iniciales, halladas con la heurística inicial, sin obtener mejoría en los resultados. Se observó que aumentar la cantidad de iteraciones no mejoraba la solución obtenida. Igual resultado se obtuvo al aumentar la diversificación e intensificación durante la búsqueda. De igual manera se probaron sin mejoría distintas combinaciones de los parámetros de la búsqueda. El comportamiento del algoritmo para este problema resultó:



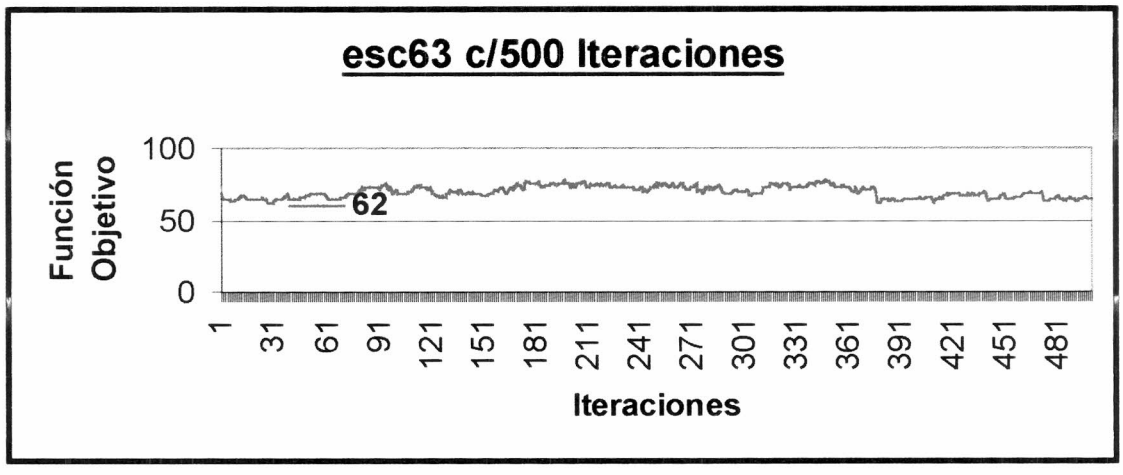
ITER	FO	CPU
1	2625	00:00:00
7	2625	00:00:04
13	2625	00:00:07
19	2625	00:00:11
25	2625	00:00:14
31	2625	00:00:17
37	2625	00:00:20
43	2625	00:00:23
49	2625	00:00:26
55	2625	00:00:29
61	2625	00:00:33
67	2625	00:00:36
73	2625	00:00:38
79	2625	00:00:41
85	2625	00:00:44
91	2625	00:00:48
97	2625	00:00:52
100	2625	00:00:53

Estudio de problemas cuyos resultados obtenidos resultaron exactos:



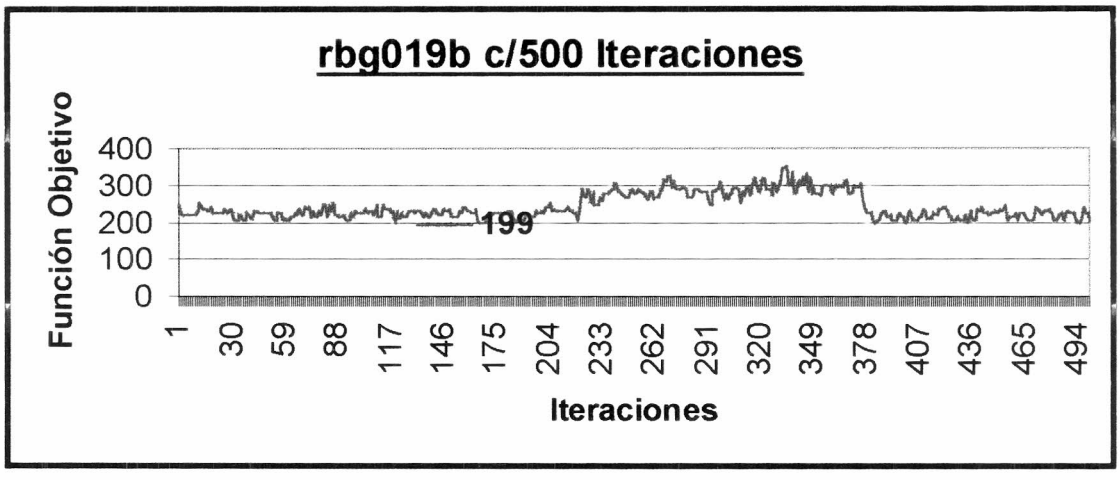
ITER	FO	CPU
1	2625	00:00:00
3	2600	00:00:00
5	2700	00:00:00
7	2825	00:00:00
9	2950	00:00:00
11	3050	00:00:00
13	2475	00:00:00
15	2125	00:00:00
17	2950	00:00:00
19	2650	00:00:00
21	3075	00:00:00
23	2725	00:00:00
25	2300	00:00:00

Estudio de problemas cuyos resultados obtenidos resultaron exactos:



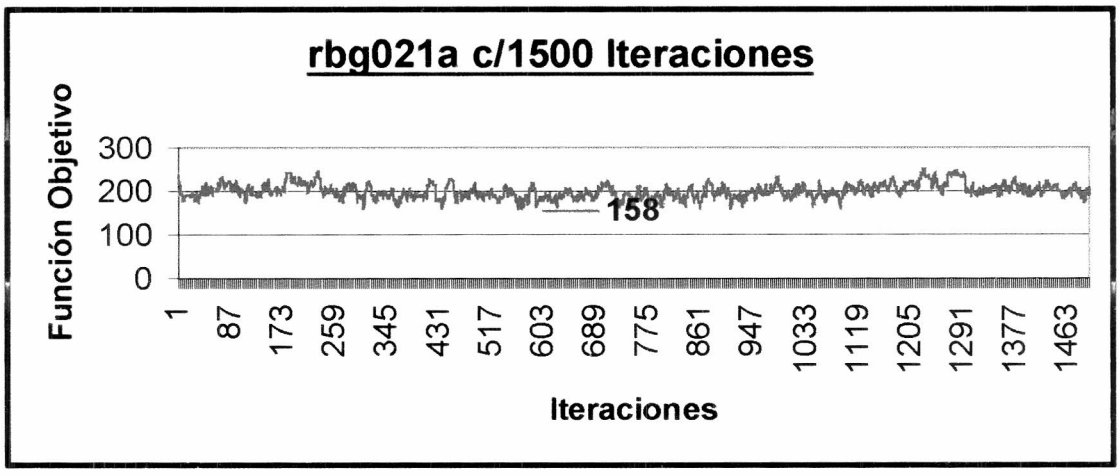
ITER	FO	CPU
1	68	00:00:00
27	62	00:00:08
31	63	00:00:09
61	65	00:00:18
91	71	00:00:28
121	70	00:00:37
151	67	00:00:47
181	75	00:00:57
211	73	00:01:07
241	72	00:01:16
271	73	00:01:26
301	70	00:01:35
331	73	00:01:44
361	73	00:01:54
391	63	00:02:03
421	68	00:02:12
451	65	00:02:21
481	67	00:02:30
500	65	00:02:36

Estudio de problemas cuyos resultados obtenidos resultaron exactos:



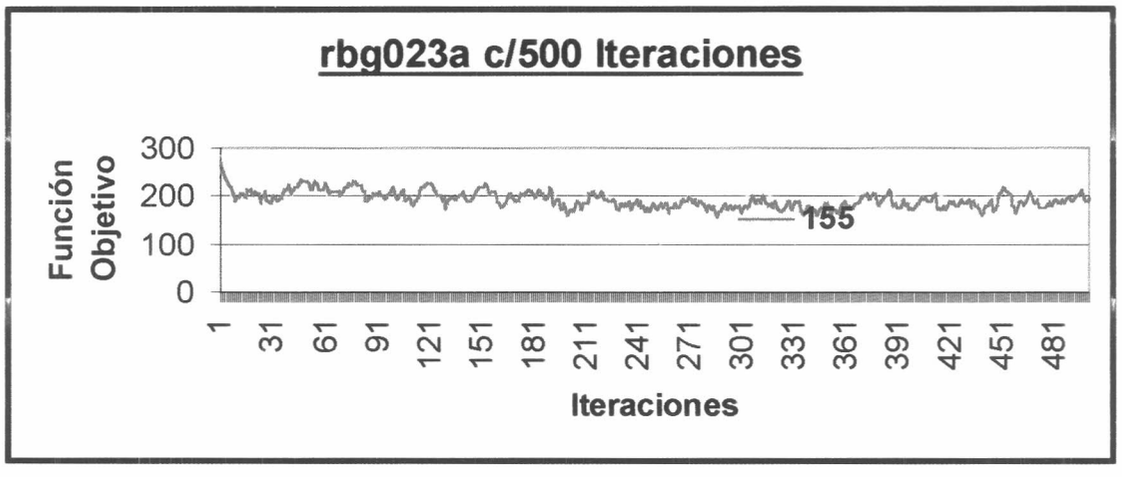
ITER	FO	CPU
1	243	00:00:00
30	214	00:00:01
59	210	00:00:01
88	217	00:00:01
117	225	00:00:01
119	199	00:00:01
146	233	00:00:01
175	225	00:00:01
204	251	00:00:02
233	256	00:00:02
262	266	00:00:02
291	263	00:00:02
320	290	00:00:02
349	298	00:00:02
378	223	00:00:02
407	214	00:00:02
436	214	00:00:03
465	214	00:00:03
494	199	00:00:03
500	205	00:00:03

Estudio de problemas cuyos resultados obtenidos resultaron exatos:



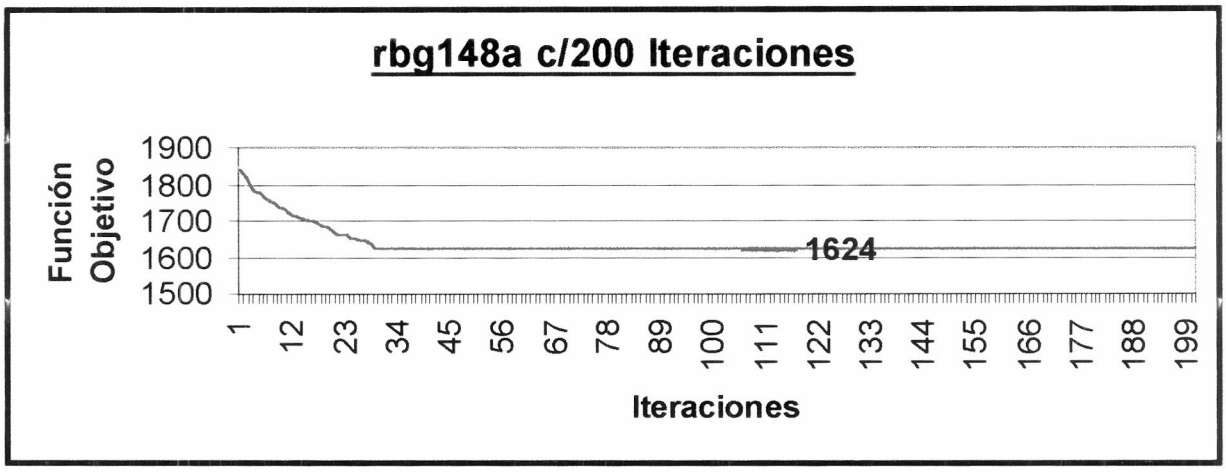
ITER	FO	CPU
1	236	00:00:00
87	205	00:00:01
173	210	00:00:01
259	191	00:00:01
345	180	00:00:02
431	182	00:00:02
517	202	00:00:02
564	158	00:00:02
603	184	00:00:03
689	184	00:00:03
775	181	00:00:03
861	173	00:00:03
947	204	00:00:04
1033	198	00:00:04
1119	206	00:00:04
1205	221	00:00:05
1291	234	00:00:05
1377	224	00:00:05
1463	189	00:00:06
1500	203	00:00:06

Estudio de problemas cuyos resultados obtenidos resultaron exactos:



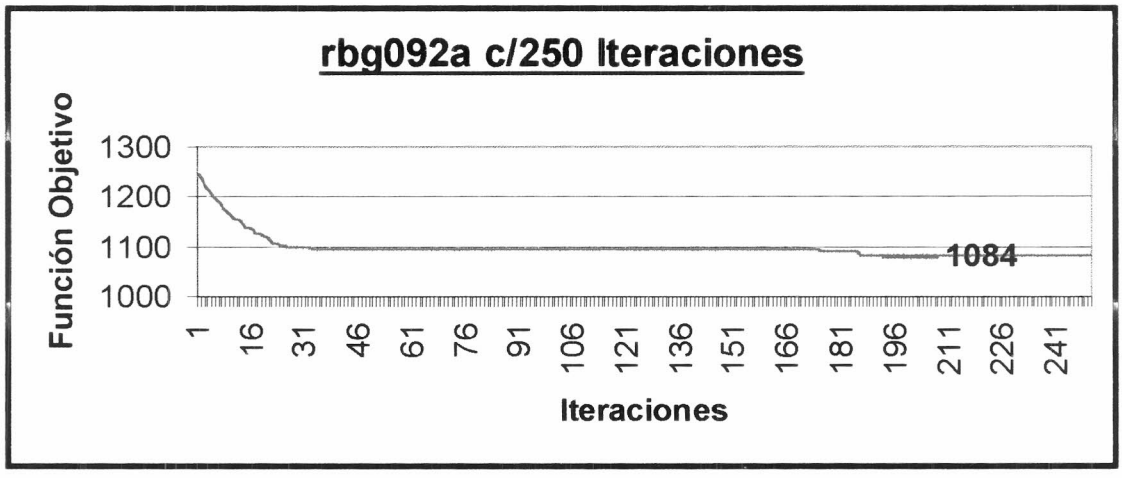
ITER	FO	CPU
1	273	00:00:00
31	201	00:00:01
61	218	00:00:01
91	203	00:00:01
121	225	00:00:02
151	219	00:00:02
181	199	00:00:02
211	186	00:00:02
241	181	00:00:02
271	195	00:00:02
286	155	00:00:02
301	178	00:00:03
331	187	00:00:03
361	176	00:00:03
391	176	00:00:03
421	185	00:00:03
451	217	00:00:03
481	194	00:00:03
500	196	00:00:04

Estudio de resultados de problemas varios:



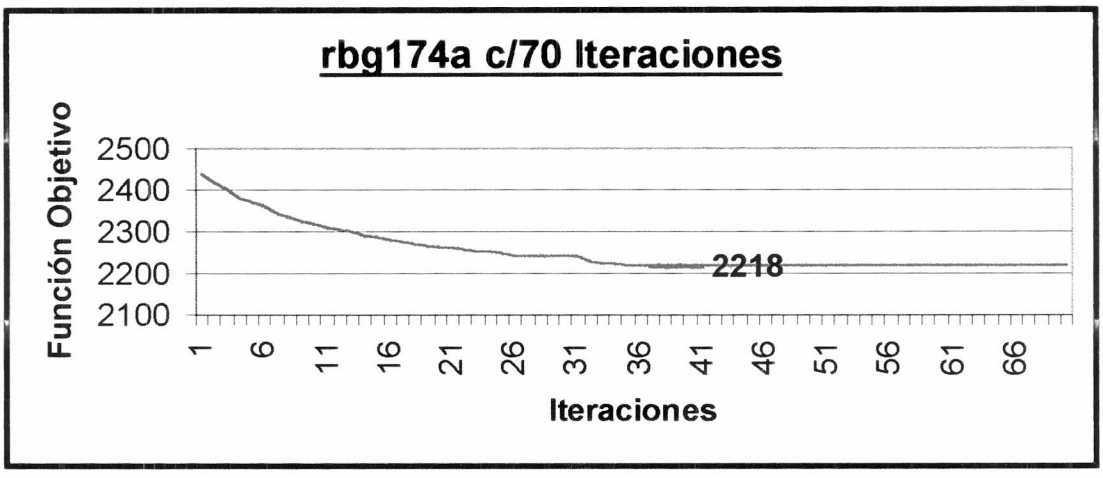
ITER	FO	CPU
1	1839	00:00:00
12	1718	00:22:37
23	1660	00:52:11
34	1626	01:21:48
35	1626	01:24:29
45	1626	01:51:49
56	1626	02:21:28
67	1626	02:51:09
78	1626	03:20:39
89	1626	03:50:37
100	1626	04:20:20
101	1624	04:23:01
111	1624	04:49:52
122	1624	05:19:27
133	1624	05:49:19
144	1624	06:18:57
155	1624	06:48:29
166	1624	07:18:08
177	1624	07:48:02
188	1624	08:17:36
199	1624	08:47:09
200	1624	08:49:51

Estudio de resultados de problemas varios:



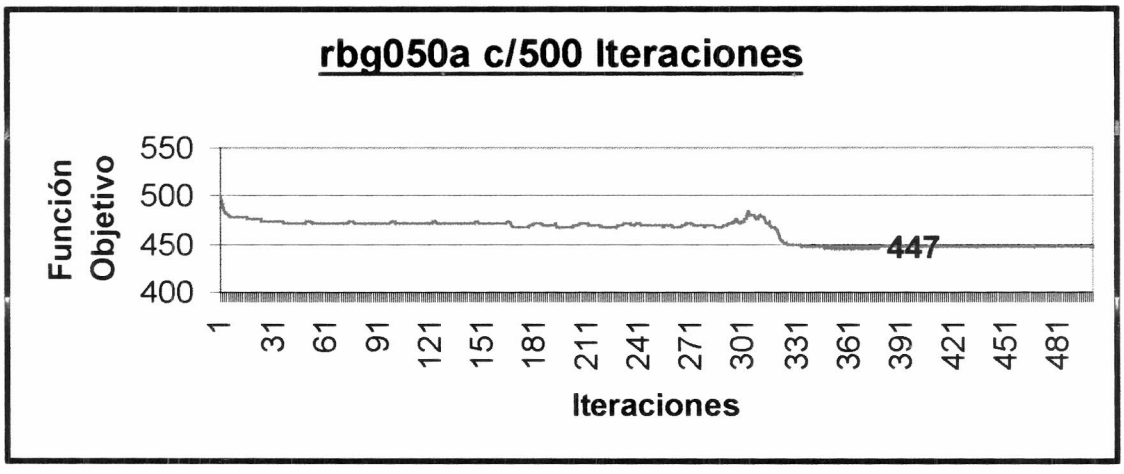
ITER	FO	CPU
1	1246	00:00:00
16	1135	00:01:37
31	1097	00:03:13
46	1096	00:04:48
61	1096	00:07:31
76	1096	00:11:18
91	1096	00:15:03
106	1096	00:18:48
121	1095	00:22:35
136	1095	00:26:22
151	1095	00:30:09
166	1095	00:33:53
181	1091	00:37:40
186	1084	00:38:56
196	1084	00:41:26
211	1084	00:45:12
226	1084	00:48:54
241	1084	00:50:34
250	1084	00:51:34

Estudio de resultados de problemas varios:



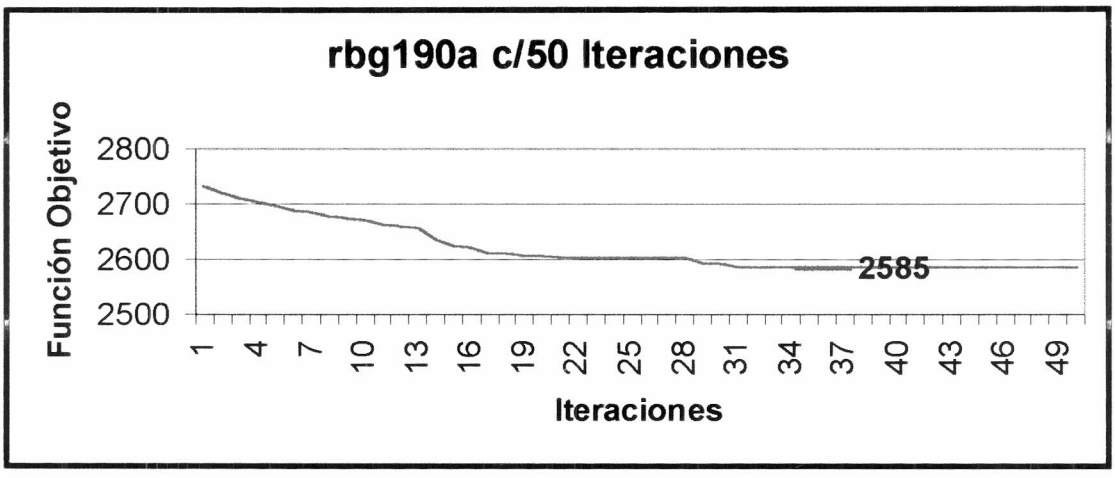
ITER	FO	CPU
1	2440	00:00:00
6	2360	00:21:15
11	2311	00:52:09
16	2281	01:23:43
21	2261	01:54:33
26	2244	02:25:23
31	2241	02:56:14
35	2218	03:21:39
36	2218	03:27:51
41	2218	03:58:43
46	2218	04:29:30
51	2218	05:00:17
56	2218	05:31:47
61	2218	06:02:41
66	2218	06:33:29
70	2218	06:58:11

Estudio de resultados de problemas varios:



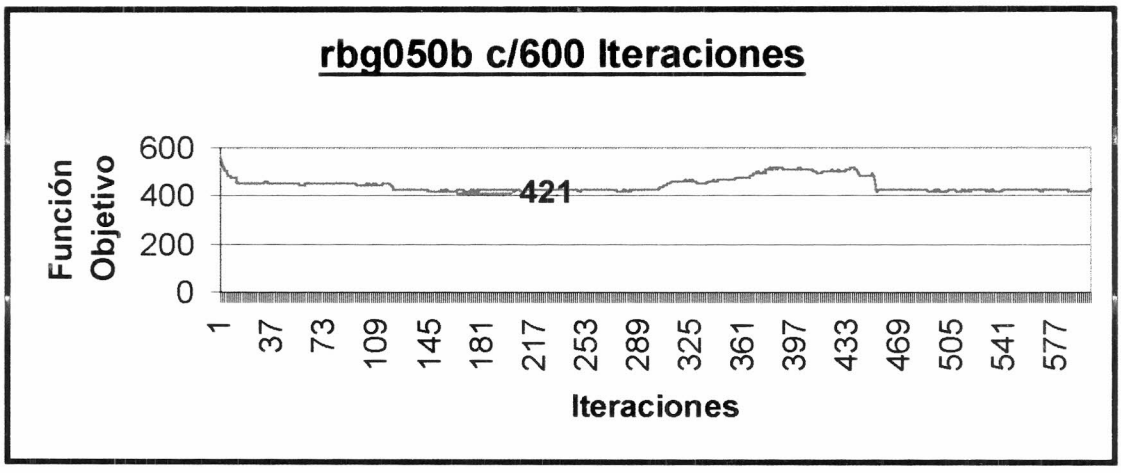
ITER	FO	CPU
1	500	00:00:00
31	473	00:00:08
61	472	00:00:16
91	471	00:00:24
121	472	00:00:32
151	472	00:00:41
181	471	00:00:49
211	471	00:00:56
241	471	00:01:04
271	471	00:01:11
301	477	00:01:19
331	450	00:01:26
334	447	00:01:27
361	448	00:01:34
391	448	00:01:41
421	448	00:01:49
451	447	00:01:56
481	448	00:02:04
500	447	00:02:08

Estudio de resultados de problemas varios:



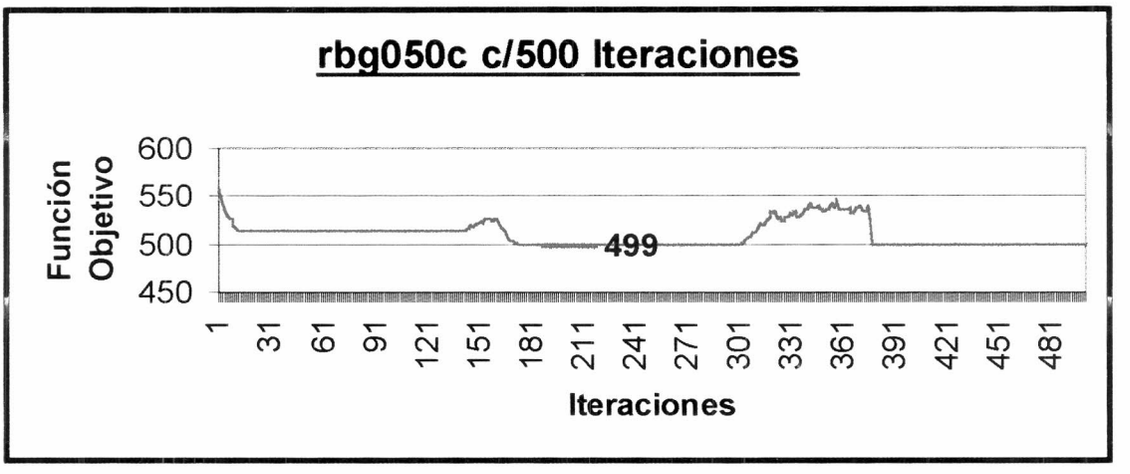
ITER	FO	CPU
1	2732	00:00:00
4	2704	00:19:51
7	2684	00:34:58
10	2670	00:54:51
13	2656	01:17:16
16	2620	01:33:15
19	2607	01:44:52
22	2604	01:57:35
25	2604	02:09:17
28	2604	02:34:17
31	2587	03:01:32
33	2585	03:19:28
34	2586	03:28:25
37	2586	03:55:21
40	2586	04:22:19
43	2586	04:49:36
46	2586	05:16:29
49	2585	05:43:17
50	2585	05:48:29

Estudio de resultados de problemas varios:



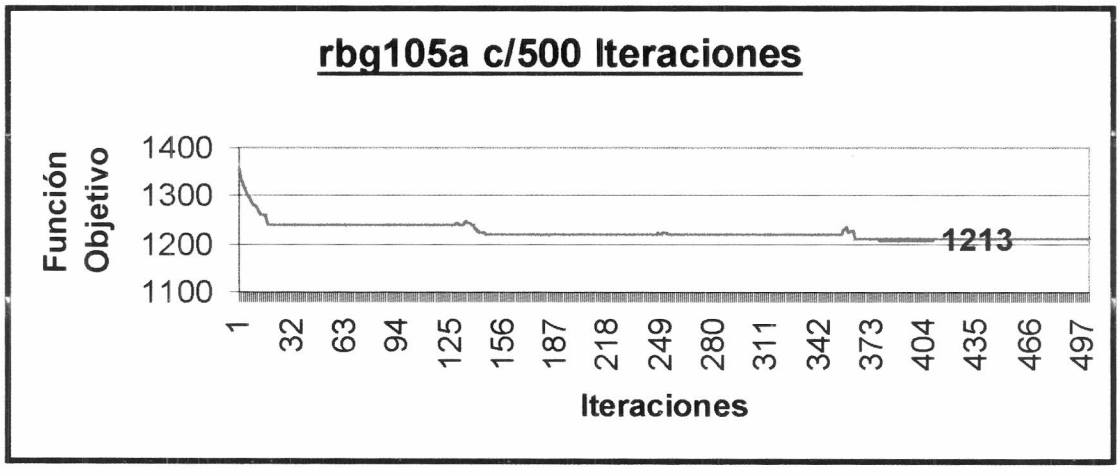
ITER	FO	CPU
1	558	00:00:00
37	451	00:00:16
73	451	00:00:58
109	446	00:01:42
145	422	00:01:57
148	421	00:02:01
181	425	00:02:37
217	424	00:02:50
253	426	00:03:05
289	425	00:03:18
325	464	00:03:29
361	478	00:03:39
397	513	00:03:50
433	513	00:04:01
469	426	00:04:12
505	424	00:04:23
541	424	00:04:34
577	425	00:04:44
600	424	00:04:51

Estudio de resultados de problemas varios:



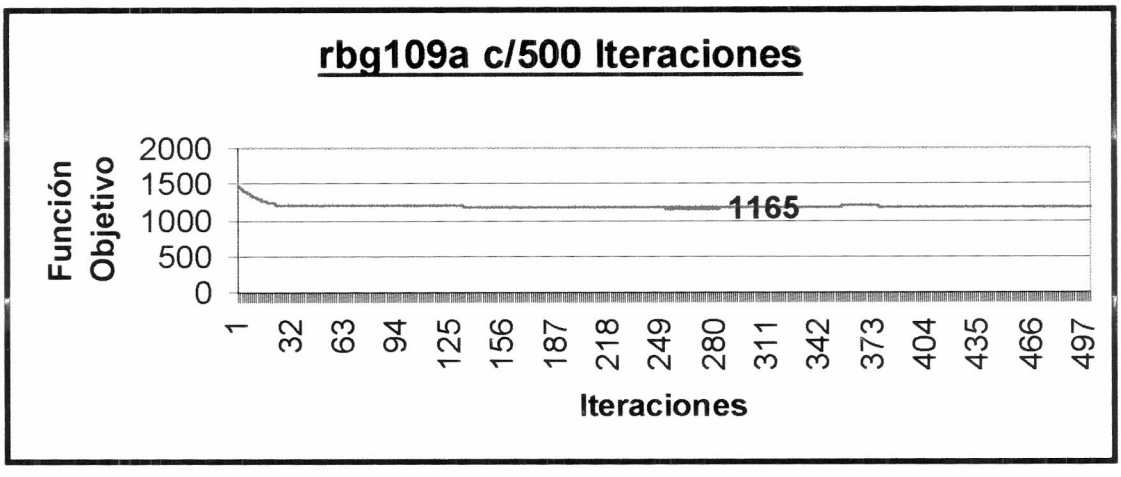
ITER	FO	CPU
1	558	00:00:00
31	514	00:00:11
61	514	00:00:20
91	514	00:00:41
121	514	00:01:00
151	523	00:01:30
174	499	00:01:55
181	499	00:02:02
211	499	00:02:22
241	499	00:02:37
271	499	00:02:48
301	500	00:03:10
331	531	00:03:39
361	536	00:04:10
391	499	00:04:27
421	499	00:04:59
451	499	00:05:20
481	499	00:05:28
500	499	00:05:32

Estudio de resultados de problemas varios:



ITER	FO	CPU
1	1357	00:00:00
32	1238	00:06:31
63	1238	00:14:56
94	1238	00:21:30
125	1238	00:29:42
156	1221	00:44:44
187	1221	00:53:03
218	1221	01:05:04
249	1222	01:15:17
280	1220	01:30:23
311	1220	01:45:33
342	1220	02:00:41
363	1213	02:10:58
373	1213	02:15:51
404	1213	02:31:01
435	1213	02:46:12
466	1213	03:01:21
497	1213	03:16:57
500	1213	03:18:25

Estudio de resultados de problemas varios:



ITER	FO	CPU
1	1478	00:00:00
32	1202	00:07:56
63	1202	00:24:20
94	1202	00:43:31
125	1202	01:03:09
156	1171	01:22:25
187	1171	01:41:40
218	1171	02:00:52
238	1165	02:13:14
249	1165	02:20:04
280	1165	02:39:16
311	1165	02:59:12
342	1166	03:18:22
373	1209	03:37:30
404	1165	03:56:39
435	1165	04:15:50
466	1165	04:35:03
497	1165	04:35:03
500	1165	04:56:25