FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN INGENIERÍA
Y CIENCIAS APLICADAS

SOLUCIÓN AL PROBLEMA DE MÁQUINAS EN PARALELO NO RELACIONADAS MEDIANTE UN ALGORITMO DE COLONIA DE HORMIGAS

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE:

MAESTRÍA EN INGENIERÍA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

P R E S E N T A:

L. I. ALINA MARTÍNEZ OROPEZA

ASESOR: DR. MARCO ANTONIO CRUZ CHÁVEZ

CUERNAVACA, MOR. AGOSTO 2010

Resumen

En este trabajo de investigación se desarrolló un algoritmo Colonia de Hormigas, para el cuál se realizó una estructura de vecindad aplicada a búsqueda local, con el objetivo de mejorar el desempeño de dicho algoritmo para el problema de Máquinas en Paralelo no Relacionadas. Se propuso e implementó una metodología de sintonización, la cuál permitió realizar el análisis de sensibilidad de los parámetros de control del algoritmo Colonia de Hormigas, lo que permitió trabajar el algoritmo con el mejor desempeño posible tanto en eficiencia como en eficacia.

Se tomó el modelo de programación lineal entera binaria para representar al problema de Máquinas en Paralelo no Relacionadas. Por su naturaleza, este modelo es considerado NP, por lo que se justifica el uso de métodos heurísticos para tratar dicho problema. La solución propuesta se compone de un algoritmo Colonia de Hormigas que involucra el empleo de una estructura de vecindad híbrida aplicada a búsqueda local, la cuál es propuesta en esta investigación. En este trabajo se realizaron dos versiones del algoritmo Colonia de Hormigas, cambiando el tipo de estructura de vecindad, con la finalidad de encontrar la mejor solución al problema tratado. Las pruebas fueron realizadas en equipo del laboratorio de Optimización del CIICAp. El análisis de resultados se llevó a cabo comparando las dos versiones del algoritmo propuesto con los resultados de un algoritmo de Recocido Simulado y con los de un Simplex Clásico utilizando variables artificiales, para finalmente comparar las dos versiones del algoritmo Colonia de Hormigas con el Algoritmo Colonia de Hormigas Clásico, para comprobar la mejora obtenida al implementar una estructura de vecindad. Las pruebas experimentales se realizaron tomando cuatro instancias generadas aleatoriamente. De acuerdo al análisis experimental realizado, el algoritmo propuesto demostró ser competitivo tanto en eficiencia como en eficacia.

Abstract

In this research, a heuristic algorithm known as Ant Colony implementing a neighborhood structure is developed to solve the Unrelated Parallel Machines Problem. A tuning methodology is proposed and implemented, which allowed performing the sensitivity analysis to the control parameters of the Ant Colony algorithm, allowing the algorithm to work with the best possible performance in both efficiency and effectiveness.

A Binary Integer Linear Programming Model is taken to represent the Unrelated Parallel Machines Problem, which, because its nature is considered as NP; therefore, the use of heuristic methods is justified to tackling this problem. The proposed solution consists of an Ant Colony algorithm with a hybrid neighborhood structure, which is proposed in this research. During this research two versions of the proposed algorithm were developed changing the neighborhood structure, in order to find the best solution to the tackled problem. The tests were realized using equipment of the Optimization Laboratory of CIICAP. The analysis of the results was performed comparing the proposed algorithm with a Simulated Annealing algorithm and with a Classic Simplex algorithm using the artifical variables method. Finally, the two versions of the proposed algorithm were compared with the Classic Ant Colony, to verify the improvement obtained by implementing a neighborhood structure. It is noteworthy that the experimental tests were realized taking four instances randomly generated. According to the performed experimental analysis, the proposed algorithm proved to be competitive in both efficiency and effectiveness.

Agradecimientos

A CONACYT por brindarme el apoyo económico sin el cuál no hubiera sido posible la realización de mis estudios de maestría.

Muy especialmente al Dr. Marco Antonio Cruz Chávez, director de este trabajo de investigación, por su orientación y apoyo para la culminación de este trabajo.

A los integrantes de mi comité tutoral y revisores de tesis: Dr. Marco Antonio Cruz Chávez, Dra. Margarita Tecpoyotl Torres, Dr. Martín G. Martínez Rangel, Dr. Martín Heriberto Cruz Rosales y Dr. David Juárez Romero, por sus acertados consejos y comentarios para la realización de este trabajo de investigación, así como para mi crecimiento personal e intelectual.

Dedicatorias

A Dios en primer lugar, por las oportunidades y ayuda que me ha brindado en el transcurso de mi vida.

A mis padres María Oropeza Díaz y Abel Martínez Pliego, por su apoyo moral, comprensión y consejos, así como por siempre impulsarme a conseguir mis metas.

A mi novio J. Gerardo Vera Dimas, por su ayuda y apoyo incondicional cuando más lo he necesitado, así como por su comprensión en tiempos difíciles y por los bellos momentos que hemos pasado juntos.

A mi asesor, el Dr. Marco Antonio Cruz, por su dirección, apoyo y sus acertados consejos para lograr la realización de este trabajo.

A mi abuelita que siempre me apoyó y estuvo conmigo, y ahora desde el cielo me manda sus bendiciones.

A mi familia que siempre me ha dado palabras de aliento para seguir adelante con las metas que me propongo.

A compañeros y amigos con los que he compartido muy gratos momentos y que me han apoyado y ayudado de manera desinteresada para lograr este reto.

Nomenclatura

NOMENCLATURA GENERAL

UPMP Siglas en inglés para el Problema de Máquinas en Paralelo no Relacionadas (Unrelated Parallel Machines Problem).

AS Siglas en inglés para el algoritmo de Recocido Simulado (Simulated Annealing).

ACO Siglas en inglés para Optimización por Colonia de Hormigas (Ant Colony Optimization).

GRASP Siglas en inglés para Procedimientos Voraces de Búsqueda Adaptables Aleatorizados (Greedy Randomized Adaptive Search Procedures).

JSSP Siglas en inglés para el Problema de Calendarización de Trabajos en un Taller de Manufactura (Job Shop Scheduling Problem).

TS Siglas en inglés para Búsqueda Tabú (Tabu Search).

NOMENCLATURA DEL PROBLEMA UPMP Y MÉTODO DE SOLUCIÓN

j Trabajo que necesita ser calendarizado.

i Máquina en la que va a ser procesado un trabajo *j.*

k Posición de una máquina *i*, en la que va a ser procesado un trabajo *j.*

m Número total de máquinas.

n	Número total de trabajos.
kPij	Tiempo de procesamiento de un trabajo *j* en la posición *k* de una máquina *i*.
N	Número total de Hormigas
A	Una hormiga del conjunto N
σ	Movimiento aplicado por una estructura de vecindad
t	Iteración

Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Dentro de la industria existen actualmente una gran variedad de problemas que requieren solución, ya que estos llevan a las empresas al gasto excesivo y uso poco eficiente de los recursos. Uno de los problemas más estudiados en esta área es la Calendarización de Trabajos en Talleres de Manufactura, el cuál se considera como parte medular de la industria, debido a que forma parte de las líneas de producción de diversas empresas a nivel mundial, del cuál se deriva el problema de Máquinas en Paralelo no Relacionadas (UPMP, por sus siglas en inglés *Unrelated Parallel Machines Problem*), mismo que ha sido estudiado por gran cantidad de investigadores. Este tipo de problemas con los que tenemos contacto en la vida diaria, pueden ser representados por medio de modelos matemáticos que involucren una función objetivo para ser tratados por la Optimización Combinatoria.

La *Optimización Combinatoria* es una rama muy importante de las ciencias computacionales, dedicada a la investigación de operaciones así como al estudio y tratamiento de problemas considerados difíciles de resolver [Papadimitriou y Steiglitz, 1998]; el objetivo a seguir en esta área es básicamente el desarrollo de métodos novedosos que permitan abordar problemas de optimización con el menor esfuerzo computacional posible. La característica principal de un problema de optimización es la búsqueda de la mejor solución mediante la implementación de métodos que permitan

reducir la dificultad para encontrar buenas soluciones al problema. Para representar un problema de optimización es necesario utilizar modelos matemáticos. Cada modelo cuenta con ciertas particularidades como es el caso de la función objetivo y las restricciones propias del problema. La función objetivo dependerá del problema tratado, de modo que se puede minimizar o maximizar.

Todos los problemas considerados en la literatura por la Optimización Combinatoria pueden ser clasificados de acuerdo al grado de dificultad para resolverlos, de aquí surge la *Teoría de la Complejidad*, la cuál se encarga de dar una clasificación a estos problemas, dividiéndolos en P, NP y NP-Duros. Los problemas P, son aquellos que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial, es decir, que la relación entre el tamaño del problema y su tiempo de ejecución es polinómica. Los problemas NP sólo pueden ser tratados por una máquina de Turing no determinista acotada en tiempo polinomial. Una de las características de estos problemas, es que el tamaño de su espacio de soluciones es de comportamiento exponencial conforme se incrementa el tamaño de la instancia [Pinedo, 2008]. Algunos de los problemas NP más conocidos son: el problema del Agente Viajero, Máquinas en Paralelo no Relacionadas [Garey et al., 1976], Caminos Hamiltonianos, etc. Los problemas NP-Duros presentan las mismas características que los NP, pero se diferencian en que estos son aún más difíciles de tratar y sus instancias resueltas hoy en día son más pequeñas en comparación con las resueltas para problemas NP. Para el caso de problemas clasificados como NP-Duros, tenemos que los más conocidos son el problema de Calendarización de Trabajos en Talleres de Manufactura y el de Calendarización de Horarios Escolares, entre otros.

En el presente trabajo de Investigación se propone una representación del UPMP para Colonia de Hormigas por medio de un grafo disyuntivo, para representar una solución particular al UPMP (clasificado como un problema de tipo NP) por medio de la realización de un algoritmo Colonia de Hormigas, para el cuál se desarrolló una estructura de vecindad híbrida, que al ser aplicada a búsqueda local, permite mejorar la explotación del espacio de soluciones.

Para la realización de pruebas experimentales fue necesario llevar a cabo un análisis de sensibilidad de los parámetros de control del algoritmo, para lo cuál se propone una metodología de sintonización.

Para mencionar algunos de los métodos de optimización más conocidos, a continuación se presenta una clasificación, así como una breve explicación de cada uno.

1.1 Métodos de Optimización Combinatoria

La solución de problemas de optimización consiste en encontrar el mejor valor de la función objetivo de modo que se satisfagan las restricciones propias de cada problema para una instancia dada. Una *instancia* es un problema de prueba que puede ser definido como el tamaño de entrada de los datos del problema.

Para obtener una buena solución a un problema de optimización clasificado como NP, es necesaria la implementación de un algoritmo no determinístico

de tipo heurístico [Papadimitriou y Steiglitz, 1998], debido a que este tipo de algoritmos permiten obtener soluciones cercanas al óptimo para instancias medianas y grandes en un tiempo computacional razonable. Una *heurística* es un método bien estructurado, desarrollado en base a la experiencia que permite obtener soluciones aproximadas de un problema sin garantizar la optimalidad [Wetzel, 1983]. Este tipo de métodos se aplican cuando no es posible encontrar una solución óptima por un método exacto, debido a la naturaleza y complejidad del problema. En la Figura 1-1 se muestra una clasificación de los principales métodos de optimización.

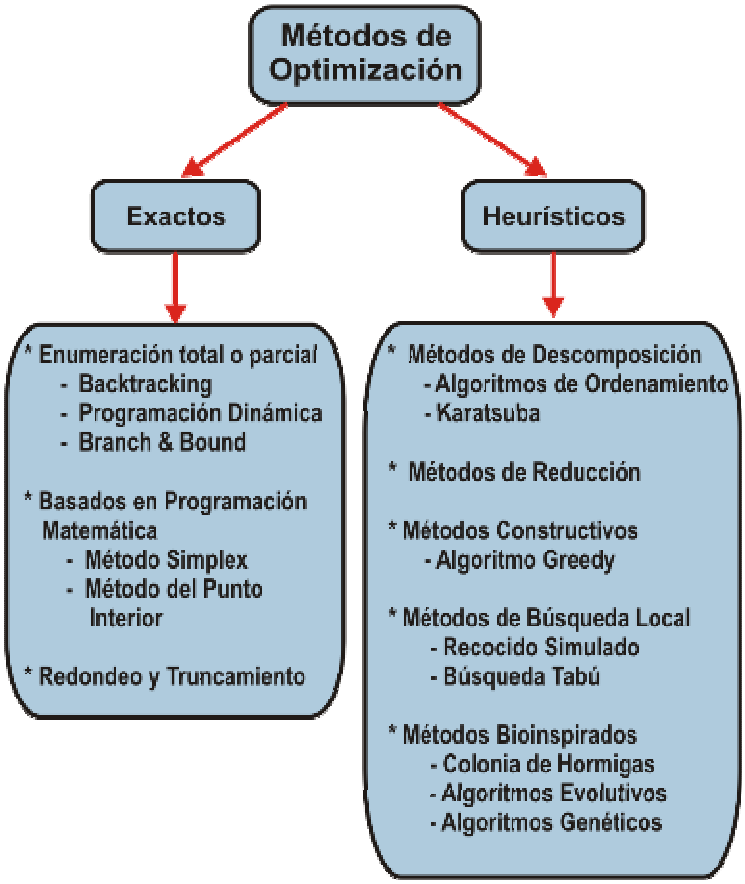


Figura 1-1. *Clasificación de los métodos de optimización*

El objetivo que persigue cada uno de los métodos mostrados en la Figura 1-1, es encontrar la mejor solución a un problema combinatorio. En el caso de métodos exactos, se puede mencionar que el Simplex Clásico [Hillier, Lieberman, 2008] es probablemente el más conocido, este método permite obtener el valor óptimo a un problema dado, aunque cuenta con ciertas limitaciones, debido a que en el caso del Simplex Clásico, está comprobado [Klee y Minty, 1972] que su complejidad es de orden exponencial, lo que significa que los tamaños de problemas que pueden ser resueltos por este método, dependerá directamente del número de variables manejadas por el problema, además de que deberá ser un problema de programación lineal.

Debido a esto, una gran cantidad de investigadores se han visto atraídos al estudio y mejora de métodos heurísticos para ser aplicados a problemas combinatorios, para los cuales no se conoce un algoritmo determinístico con comportamiento polinomial que los resuelva, puesto que los tiempos requeridos para encontrar una solución a instancias consideradas de tamaño grande son extremadamente largos.

Una *heurística* es un procedimiento para resolver un problema de optimización bien definido, mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución en un tiempo computacional razonable [Riojas, 2005] para instancias medianas y grandes de problemas clasificados como difíciles. A partir de ésto, surgen las denominadas *metaheurísticas*, las cuales son una clase de métodos aproximados que están diseñados para resolver problemas complejos de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos [Osman y Nelly, 1996]. Dentro de ésta clasificación se

encuentran una gran variedad de métodos de naturaleza muy diferente, por lo que se complica el dar una clasificación completa. A continuación se realiza una clasificación para ubicar a los métodos heurísticos más conocidos [Riojas, 2005] y [Alba, 2005].

- ***Métodos de Descomposición (Divide y Vencerás)***. Son métodos donde el problema original es descompuesto en subproblemas más pequeños y por ende más sencillos de resolver, teniendo en cuenta que todos pertenecen al mismo problema. Algunos ejemplos de métodos de este tipo son los algoritmos de Ordenamiento [Aho, et al., 1974] y [Michalewicz y Fogel, 2002].

- ***Métodos de Reducción***. Métodos que consisten en analizar e identificar las propiedades que cumplen generalmente las buenas soluciones e introducirlas como restricciones del problema. El objetivo es acotar el espacio de soluciones, simplificando el problema, aunque existe el riesgo de dejar fuera la solución óptima del problema.

- ***Métodos Constructivos***. Métodos que consisten en ir construyendo paso a paso una solución a un problema dado. Normalmente son métodos deterministas y suelen estar basados en la elección de la mejor solución en cada iteración. Estos métodos han sido muy utilizados en problemas de optimización clásicos, como es el caso del problema del Agente Viajero. Un ejemplo de este tipo de métodos son los algoritmos voraces (*Greedy*) [Michalewicz y Fogel, 2002].

- ***Métodos de Búsqueda Local.*** A diferencia de los métodos mencionados anteriormente, los procedimientos de búsqueda local comienzan con una solución inicial y la van mejorando progresivamente. El procedimiento se lleva a cabo realizando en cada

iteración, un movimiento de una solución a otra que mejore la solución anterior. Este movimiento puede ser una permutación, inserción o eliminación. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore. La diferencia con los métodos analíticos es que estos no necesariamente encontraran una solución óptima. Ejemplos de éste tipo de métodos son Búsqueda Tabú, Recocido Simulado, etc [Michalewicz y Fogel, 2002].

- ***Métodos Bioinspirados:*** Esta clasificación engloba el conjunto de algoritmos que simulan un proceso "inteligente" de los animales que viven en sociedad, como las abejas, las termitas, las hormigas, entre otros. En estas heurísticas no necesariamente se simula un proceso de evolución [Téllez, 2007], sino que pueden simular el comportamiento de dichas comunidades al realizar ciertas actividades, como es el caso de búsqueda de alimento. Ejemplos de estos métodos son Colonia de Hormigas, Algoritmos Evolutivos, Algoritmos Genéticos, etc.

1.2 Teoría de la Complejidad de los Algoritmos

La *Teoría de la Complejidad* se enfoca en dos aspectos principales: el tiempo y los recursos necesarios para resolver un problema computacional (*espacio*). Estos parámetros son de vital importancia para definir si un algoritmo es eficiente o no; ya que si este requiere mucho tiempo para resolver un problema, no será de utilidad, lo mismo sucede si requiere gran cantidad de memoria. En el caso de que un algoritmo requiera más tiempo que otro para converger en una solución, se puede decir que este algoritmo

tiene mayor *complejidad temporal*; para el caso de mayores requerimientos de memoria, se maneja el término de *complejidad espacial*.

Un algoritmo, independientemente de su funcionamiento o la forma en que realiza el proceso de solución de un problema, necesita datos de entrada, los cuales, dependiendo del problema tratado, influyen en el tiempo de convergencia del algoritmo, ya que mientras mayor sea el tamaño de la entrada, mayor será el tiempo de ejecución requerido para su procesamiento.

La complejidad temporal de un algoritmo se encuentra representada por medio de una función temporal, la cuál es dependiente del tamaño de la entrada y del número de instrucciones que requieren ser evaluadas para resolver el problema.

Un claro ejemplo de lo explicado anteriormente, es que si un algoritmo presenta un tiempo de ejecución constante $T(n) = k$, se puede decir que este algoritmo es eficiente, debido a que si el tamaño de la entrada aumenta, el tiempo necesario para su ejecución permanecerá constante. En el caso de un algoritmo con complejidad logarítmica $T(n) = log(n)$, también es considerado eficiente, debido a que, por ejemplo, si el tamaño de la entrada se hace 100 veces más grande, el tiempo requerido para su procesamiento únicamente se duplica; por el contrario, si un algoritmo presenta una complejidad de tipo exponencial $T(n) = 2^n$, éste es considerado ineficiente [Bisbal, 2009] y [Aho, et al., 1974]. En la Tabla 1-1 *se* muestran los tipos de complejidad existentes.

Tabla 1-1. *Tipos de complejidad para el estudio de eficiencia de los algoritmos [Aho, et al. 1974] y [Bisbal, 2009].*

Velocidad de Crecimiento	Nombre
k	constante
log(*n*)	logarítmica
n	lineal
n log(*n*)	cuasi-lineal
n^2	cuadrática
n^k	polinómica
2^n	exponencial

El estudio de eficiencia se lleva a cabo analizando el caso extremo de los algoritmos, es decir, el peor caso, el cuál se encuentra definido como el valor en el que el algoritmo tendrá que realizar la mayor cantidad de operaciones. En la Figura 1-2 se muestra el comportamiento de algunas de las funciones de complejidad con respecto al tiempo.

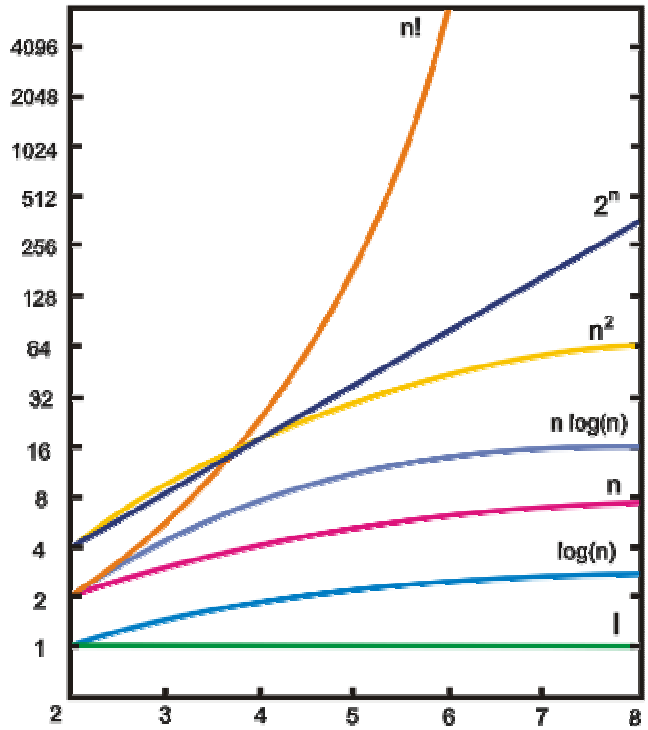


Figura 1-2. *Representación del crecimiento de funciones utilizadas comúnmente en las estimaciones con notación O. [Rosen, 2003]*

La importancia de los algoritmos en cuanto a la complejidad, se basa en diseñar algoritmos que cuenten con una complejidad logarítmica, debido a que independientemente del equipo en el que sea ejecutado, el tiempo de cómputo será razonable.

1.3 Estado del Arte

La utilización de metaheurísticas para resolver problemas de optimización se caracteriza principalmente por aplicar búsquedas en vecindades (conjunto de soluciones alcanzables a partir de una solución); razón principal para la construcción de algoritmos de búsqueda eficientes y eficaces que permitan la mejora considerable de soluciones obtenidas por las metaheurísticas. Para la búsqueda de soluciones al problema de Máquinas en Paralelo no Relacionadas, se han propuesto diversas metaheurísticas y algoritmos exactos por alrededor de 10 años. Para mejorar las soluciones encontradas por dichos algoritmos se han utilizado diferentes tipos de estructuras de vecindad aplicadas a búsqueda local, cuyas características influyen en la calidad de las soluciones obtenidas por el algoritmo.

En la literatura se pueden encontrar diversos enfoques que permiten obtener buenas soluciones para el problema de Máquinas en Paralelo no Relacionadas, ya sea considerando o no, los tiempos de preparación (*setup times*), además de implementar deversos tipos de estructuras de vecindad.

En Anastasova y Dror, 1998, se describe una aplicación enfocada en el análisis de un proceso soportado para un centro automatizado de

computación en línea. Se modela la actividad de consulta en línea del centro como un grupo de máquinas en paralelo no relacionadas. Los usuarios tienen un proceso de solicitudes de ayuda en línea, la cuál es una lluvia de trabajos entrantes. El índice automático es un tipo de Procesamiento Natural del Lenguaje, el cuál es aplicado a cada trabajo con el fin de calcular el tiempo de procesamiento dependiendo de cada máquina. El modelo de UPMP se ha utilizado para resolver el problema del transporte, y se han encontrado soluciones por medio de algoritmos de aproximación, demostrando tener mejor rendimiento que con formulaciones de programación lineal entera [Koranne, 2002]. Guo et al., 2007, implementan un recocido simulado modificado y un algoritmo de búsqueda tabú, donde se aplica una estructura de vecindad, cuyos movimientos son reasignar un trabajo de una máquina con mayor costo de procesamiento a otra, así como el intercambio ó permutación de dos trabajos asignados a diferentes máquinas, este proceso se lleva a cabo para mejorar la calidad de las soluciones encontradas.

Se desarrolló un algoritmo de redondeo para encontrar una calendarización en máquinas en paralelo no relacionadas [Anil y Marathe, 2005], este algoritmo trabaja bien con modelos de programación lineal, cuadrática y convexa expandida para optimizar el makespan (tiempo en el cuál la última operación es ejecutada dentro de un proceso tomando en cuenta el tiempo total de término de todos los trabajos). Chun y Chuen, 2006 presentan la construcción de un calendarizador que permite identificar un cuello de botella, al localizarlo, se provoca un paro en el calendarizador, mismo que se emplea en la asignación de trabajos entre intervalos identificados como cuellos de botella.

En algunos trabajos, se propone un modelo de programación lineal entera mixta que utiliza una metaheurística basada en GRASP (procedimiento iterativo, en el que cada caso consiste en una fase constructiva y una de mejora) e implementa una estructura de vecindad, la cuál es elegida de forma arbitraria [Gómez et al., 2007]; mientras que Zhou et al., 2007, utilizan un algoritmo de búsqueda local para optimizar el total de tardanza ponderada en el UPMP, para lo que se propone un algoritmo híbrido de colonia de hormigas. Por otro lado, Zhou et al., 2007 presentan un algoritmo de Optimización por Colonia de Hormigas (ACO), el cuál incorpora una estructura de vecindad de tipo 2-opt (movimiento de doble intercambio) para resolver el problema de UPMP. Otra solución propuesta es simplificar el problema a uno de asignación, con el objetivo de reducir los tiempos de preparación necesarios para el intercambio de la producción de un tipo de producto a otro, para lo cuál se utilizó un algoritmo Branch and Bound [Pessan et al., 2007]. Pei y Shih, 2007, tratan el problema de Máquinas en Paralelo no Relacionadas utilizando tiempos de preparación, con algoritmos genéticos, aplicando una estructura de vecindad con movimientos tipo 2-opt y 3-opt (movimiento de triple intercambio), de forma que obtienen el óptimo para instancias pequeñas y buenas soluciones para instancias grandes.

Otro método de optimización utilizado para resolver el UPMP dando muy buenos resultados es el Recocido Simulado, el cuál involucra el uso de una búsqueda local iterada con un procedimiento de movimiento e intercambio y un algoritmo de búsqueda tabú con vecindarios híbridos que involucra un operador de perturbación que consiste en intercambios aleatorios [Anagnostopoulos y Rabadi, 2002]. Para resolver el problema de máquinas en paralelo no relacionadas con tiempos de inicio diferentes y tiempos de preparación, se utilizó un procedimiento de

intercambio y un algoritmo de búsqueda tabú con vecindarios híbridos [Chun, 2008], ésto con la finalidad de minimizar el número total de trabajos tardíos ponderados. Un algoritmo de Optimización por Colonia de Hormigas (ACO por sus siglas en inglés), es utilizado para resolver el problema de máquinas en paralelo no relacionadas con máquinas dependientes y secuencia de trabajos dependiente de los tiempos de preparación y sin considerar interrupciones [Arnaout et al., 2008], con el objetivo de minimizar el makespan . Mönch, 2008, propone un algoritmo ACO utilizando la regla de despacho , ATC (*Apparent Tardiness Cost*), a la que se realizaron algunas mejoras, además de implementar una heurística de descomposición para dividir el problema y resolverlo en subprogramas.

Para mejorar las soluciones obtenidas para el UPMP, Vallada y Ruiz, 2009, utilizaron un algoritmo genético, al cuál se aplica una estructura de vecindad de inserción, misma que permite reducir tiempo de ejecución del algoritmo, así como mejorar la calidad de las soluciones. Otro método que ya había sido aplicado anteriormente para resolver este problema y que se retoma para mejorar un algoritmo de búsqueda local [Mateo, 2009], es el GRASP; este algoritmo combina un procedimiento de intercambio de una o distintas máquinas y un procedimiento de inserción, lo que permite mejorar las soluciones obtenidas.

El UPMP es una variante del problema de Calendarización de Máquinas en un Taller de Manufactura (JSSP), por lo que al realizar una relajación de éste, se obtiene un mapeo al UPMP [Cruz et al., 2009b] y [Cruz et al., 2010a], lo que permite modelarlo por medio de una grafo bipartita. En la literatura varios autores han incorporado diversos tipos de estructuras de vecindad a búsqueda local, con la finalidad de mejorar tanto el tiempo de ejecución del algoritmo, como la calidad de las soluciones, de acuerdo a

ésto, es propuesto el desarrollo de una estructura de vecindad híbrida para ser aplicada a búsqueda local, la cuál de acuerdo a las pruebas experimentales realizadas para el problema del Agente Viajero [Cruz et al., 2010b], obtiene mejores resultados que algunas estructura de vecindad sencillas.

Todos los enfoques mencionados requieren la evaluación en cuanto a la calidad de las soluciones para cumplir la función objetivo del problema, en algunos casos se trata de encontrar el makespan y en otros el tiempo total de término de los trabajos. El *makespan* se define como el tiempo en que la última operación O_i es completada y el tiempo total de término de todos los trabajos, como su nombre lo indica es el tiempo requerido para que todas las tareas sean completamente procesadas.

La mayoría de los trabajos relacionados con Máquinas en Paralelo no Relacionadas consideran el uso de tiempos de preparación [Gómez et al., 2007], [Anagnostopoulos y Rabadi, 2002], [Chun, 2008], [Arnaout et al., 2008], [Mönch, 2008]; mismos que no serán incluidos en esta investigación, para resolver el problema tratado. Zhou et al., 2007, proponen un algoritmo de Optimización por Colonia de Hormigas (ACO) con un gen operador de transferencia PGA , el cuál se encarga de controlar los movimientos realizados en una búsqueda local, donde la función objetivo es minimizar el total de tardanza ponderada. De acuerdo a la revisión bibliográfica realizada, se reconoce que los trabajos de investigación para el problema de Máquinas en Paralelo no Relacionadas sin tiempos de preparación (setup times) son muy escasos.

1.4 Objetivo de la Investigación

1. Desarrollo de un algoritmo Colonia de Hormigas secuencial aplicando una estructura de vecindad híbrida a búsqueda local con la finalidad de mejorar la eficiencia y eficacia del algoritmo, el cuál fue utilizado para resolver el Problema de Máquinas en Paralelo no Relacionadas sin contemplar tiempos de preparación (setup times) ni interrupciones (preemptions).

2. Realizar pruebas al algoritmo Colonia de hormigas propuesto y comparar la calidad de las soluciones obtenidas con los resultados obtenidos por un algoritmo de Recocido Simulado y un método exacto conocido como Simplex Clásico utilizando la técnica de variables artificiales [Hillier, Lieberman, 2008], los cuales ya fueron reportados en la literatura. Las pruebas se llevaron a cabo aplicando las mismas instancias de prueba para poder realizar una comparación directa y confiable.

3. Comparar los resultados del algoritmo Colonia de Hormigas propuesto con los obtenidos por un algoritmo Colonia de Hormigas Clásico [Dorigo et al., 1996], utilizando las mismas instancias de prueba para mostrar la mejora aplicada al algoritmo.

1.5 Alcance de la Investigación

1. El algoritmo de Colonia de Hormigas propuesto contará con dos partes fundamentales, la primera es la implementación de una

estructura híbrida aplicada en búsqueda por vecindad, que permita una mejor explotación del espacio de soluciones y la segunda parte comprende un estudio de sensibilidad a los parámetros de control del algoritmo propuesto, lo que mejorará el tiempo de convergencia y la calidad de las soluciones del algoritmo.

2. Los resultados obtenidos durante las pruebas realizadas al algoritmo Colonia de Hormigas se compararon con los obtenidos por un algoritmo de Recocido Simulado y un método Simplex Clásico utilizando la técnica de variables artificiales, en cuanto al tiempo de convergencia y calidad de las soluciones, tomando en cuenta que son evaluados con las mismas instancias.

3. Las instancias de prueba son generadas de forma aleatoria y se utilizan tanto para el algoritmo propuesto como para los algoritmos de Recocido Simulado, Simplex y Colonia de Hormigas Clásico, lo cuál servirá para realizar una comparación en cuanto a la eficiencia y eficacia del algoritmo.

4. El algoritmo se propone para resolver el modelo matemático de programación lineal entera binaria del UPMP, donde no se toman en cuenta interrupciones ni tiempos de preparación.

1.6 Contribución de la tesis

En este trabajo se desarrolló un algoritmo Colonia de Hormigas con una estructura de vecindad híbrida para resolver el problema de Máquinas en Paralelo no Relacionadas.

Una de las contribuciones presentadas en este trabajo de investigación es el modelo de grafo disyuntivo que se utiliza para representar al problema de Máquinas en Paralelo no Relacionadas para Colonia de Hormigas, además del desarrollo e implementación de una estructura de vecindad híbrida aplicada a búsqueda local.

Una parte fundamental del algoritmo colonia de hormigas es el análisis de sensibilidad de los parámetros de control (importancia relativa de la feromona, importancia de la distancia heurística, coeficiente de evaporación y cantidad de feromona por unidad de tiempo), los cuales son términos independientes de las restricciones. De acuerdo a lo anterior, se propone una metodología para la realización del análisis de sensibilidad, la cuál lleva a cabo la sintonización de las variables de entrada del algoritmo; esto permite conocer que tan sensible es el algoritmo al cambio de valor de ciertos parámetros y de esta forma determinar el rango de valores dentro del cuál, la solución sigue siendo buena.

1.7 Organización de la tesis

En el capítulo 1 se da una introducción del problema a tratar; se da una breve explicación del estado del arte y se enumeran los objetivos y alcances de la investigación, así como la organización general de la tesis. En el capítulo 2 se da una introducción al problema de Máquinas en Paralelo no Relacionadas y la descripción conceptual del problema, además de presentar detalladamente la representación del problema tratado para Colonia de Hormigas mediante un grafo disyuntivo, su representación matricial, así como su modelado por medio de un grafo bipartita y la formulación matemática del mismo.

En el capítulo 3 se da una explicación introductoria del algoritmo Colonia de Hormigas, se explica ampliamente la estructura de vecindad híbrida propuesta aplicada a búsqueda local, se muestra un esquema generalizado del algoritmo Colonia de Hormigas propuesto para mejorar las soluciones obtenidas para el problema de Máquinas en Paralelo no Relacionadas; se explica la metodología propuesta para la realización del análisis de sensibilidad utilizado para obtener la sintonización de los parámetros de control, además de presentar la función temporal correspondiente a la complejidad del algoritmo propuesto.

En el capítulo 4 se da una explicación del proceso realizado para la sintonización de parámetros. Se explica la aplicación de la estructura de vecindad propuesta aplicada a búsqueda local para el algoritmo Colonia de Hormigas, se muestran los resultados experimentales obtenidos en las pruebas realizadas al algoritmo propuesto, además de presentar los resultados obtenidos en cuanto a eficiencia y eficacia, comparado con los

resultados obtenidos por un algoritmo de Recocido Simulado y un algoritmo Simplex Clásico utilizando el método de variables artificiales, para finalmente llevar a cabo el análisis de los resultados obtenidos en las pruebas mencionadas anteriormente. Además de realizar una comparación en cuanto a eficacia del algoritmo Colonia de Hormigas Clásico [Dorigo et al., 1996] vs. Colonia de Hormigas con una estructura híbrida y Colonia de Hormigas con una estructura sencilla de un par aleatorio, esto se realiza con la finalidad de probar la mejora del algoritmo al aplicar una estructura de vecindad para búsqueda local.

En el capítulo 5 se dan las conclusiones finales del trabajo de investigación, así como los trabajos futuros, mismos que se enumeran en el punto 5.2.

Capítulo 2

Máquinas en Paralelo no Relacionadas

En este capítulo se da una explicación del problema de Máquinas en Paralelo no Relacionadas, se presenta su formulación matemática por medio de un modelo de programación entera binaria, así como su representación para Colonia de Hormigas y su modelado por medio de un grafo bipartita.

El problema de Máquinas en Paralelo no Relacionadas, es una variante del problema clásico de Calendarización de Trabajos en Talleres de Manufactura, debido a que a partir de éste, se puede conseguir un mapeo al UPMP [Cruz et al., 2009b] [Cruz et al., 2010a]. De acuerdo a la naturaleza del problema tratado, al ser máquinas no relacionadas, significa que las máquinas tienen capacidades diferentes, por lo que el tiempo de procesamiento para cada operación (costo) dependerá de la máquina así como de la posición a la que ésta sea asignada.

La solución al problema tratado se encuentra en base a la función objetivo, misma que será explicada más detalladamente en el punto concerniente a la formulación matemática del problema.

2.1 Modelado del Problema de Máquinas en Paralelo no Relacionadas por Medio de un Grafo Bipartita

Un grafo bipartita se define en la literatura como un grafo no dirigido que tiene la característica de que su conjunto total de vértices *V* puede ser dividido en dos subconjuntos disjuntos $G = \{V_1 \cup V_2, A\}$, de tal forma que cada arco del conjunto *A* conecta un vértice de cada subconjunto, es decir, un arco conecta un elemento del subconjunto V_1 y uno del V_2, tomando en cuenta que no deben existir adyacencias entre elementos de un mismo subconjunto [Rosen, 2003]. A continuación se enumeran las características básicas de un grafo bipartita.

- Los subconjuntos V_1 y V_2 son disjuntos y no vacíos.
- Cada arco de *A* une un vértice de V_1 con uno de V_2.
- No existen arcos uniendo dos elementos de V_1; análogamente para V_2.

Tomando en cuenta las características mencionadas anteriormente, se representó una solución a una instancia de 3 x 3 (Tabla 2-1).

Tabla 2-1. *Solución al problema de JSSP para una instancia de 3 máquinas y 3 trabajos con 3 operaciones cada uno.*

Trabajos	*Operaciones* Máquinas (Tiempo de Procesamiento) 1	2	3
1	1(1)	2(2)	3(1)
2	1(3)	3(1)	2(3)
3	2(2)	1(2)	3(3)

Una solución a esta instancia se llevó a cabo por medio de un grafo tipo bipartita mostrado en la Figura 2-1. Esto se realizó debido a que existen algoritmos exactos de programación lineal que resuelven grafos bipartitas, como es el caso del método Simplex Clásico utilizando la técnica de variables artificiales [Hiller, Lieberman, 2002].

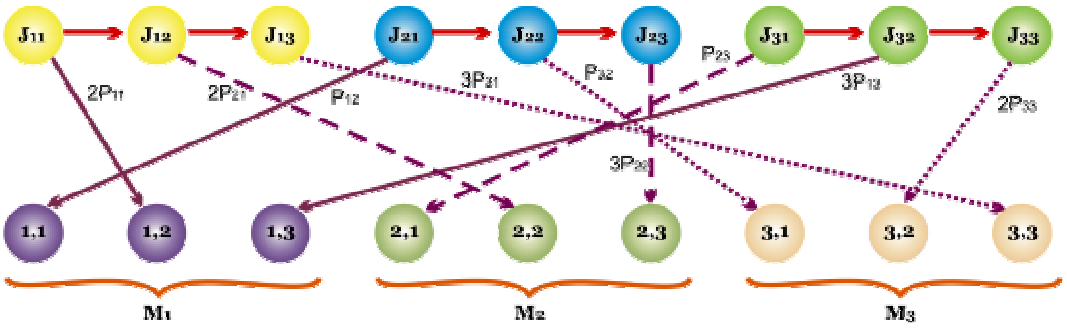


Figura 2-1. *Modelado de una solución a una instancia del JSSP por medio de un grafo tipo bipartita, donde prevalecen las restricciones de precedencia de las operaciones de cada trabajo.*

El grafo está compuesto por *n* trabajos y m_k posiciones, donde cada máquina puede procesar solamente un trabajo; por lo que si el trabajo *j* es asignado a la posición i_k, su tiempo de procesamiento estará dado por kP_{ij}, donde *k* es la posición de la máquina *i* donde será procesado el trabajo *j*, lo que indica que el tiempo de procesamiento estará en función de la máquina y la posición a la que sea asignando el trabajo; de acuerdo a esto, el objetivo es determinar la calendarización i_k para cada trabajo *j* utilizando un grafo bipartita, de forma que se obtenga el mínimo costo para el tiempo total de término de todos los trabajos.

De acuerdo a la representación mostrada en la Figura 2-1, se dice que es un grafo *tipo* bipartita, debido a que prevalece la restricción de precedencia

entre las operaciones de cada trabajo, por lo que es necesario realizar una relajación de esta restricción del problema para poder representarlo como un grafo bipartita auténtico y de esta forma tratar de resolverlo por medio de algoritmos de programación lineal.

Para llevar a cabo la representación del problema, es necesario realizar un análisis de todas y cada una de las restricciones, con lo que se llegó a la conclusión de que algunas de las restricciones son elementales, por lo que pueden se tomadas de forma implícita, de modo que no es necesario representarlas gráficamente; al realizar esto, la restricción de precedencia es tomada implícitamente, de forma que el grafo queda representado como se muestra en la Figura 2-2.

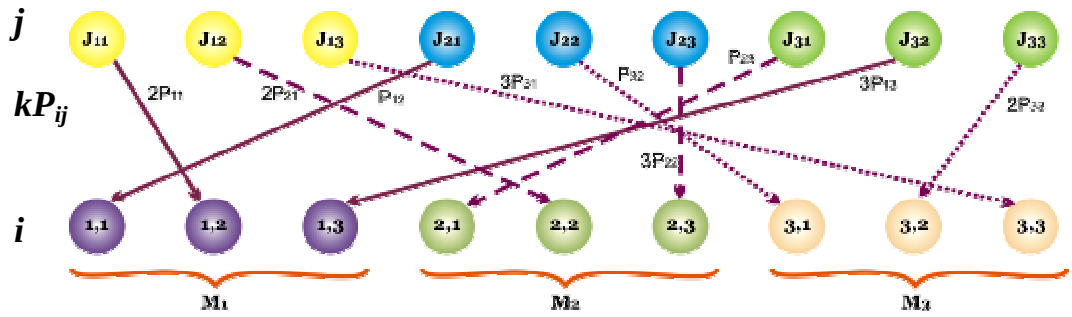


Figura 2-2. *Relajación de la restricción de precedencia para ser representado a través de un grafo bipartita auténtico [Cruz et al., 2009b]* y *[Cruz et al., 2010a].*

La representación de una posible solución a una instancia del problema tratado se puede interpretar de la siguiente forma. Por ejemplo, a la máquina $i = 1$ se asignan en $k = 1$, la operación *1* del trabajo *2* (P_{12}), en $k = 2$ se asigna la operación *1* del trabajo *1* ($2P_{11}$) y en $k = 3$ será procesada la operación *2* del trabajo *3* ($3P_{13}$); para $i = 2$, en $k = 1$ se llevará a cabo la operación *1* del trabajo *3* (P_{23}), en $k = 2$ es asignada la operación *2* del

trabajo *1* ($2P_{21}$) y en *k = 3* se realizará la operación *3* del trabajo *2* ($3P_{22}$); para el caso de *i = 3*, se asigna a *k = 1* la operación *2* del trabajo *2* (P_{32}), en *k = 2* es asignada la operación *3* del trabajo *3* ($2P_{33}$) y finalmente la operación *3* del trabajo *1* será procesada en *k = 3*($3P_{31}$).

Analizando la figura 2-2 donde se llevó a cabo la representación del problema, se puede observar que se obtiene un mapeo al UPMP.

Tomando la definición del problema mencionada al inicio de este capítulo, la representación para la instancia mostrada en la tabla 2-1, es un grafo bipartita no dirigido, mostrado en la figura 2-3 [Pinedo, 2008], donde cada trabajo puede ser procesado en cualquier máquina, por lo que de esta forma se estaría relajando la restricción de precedencia y se tendría representada de forma gráfica únicamente la restricción de capacidad de recursos [Pinedo, 2008].

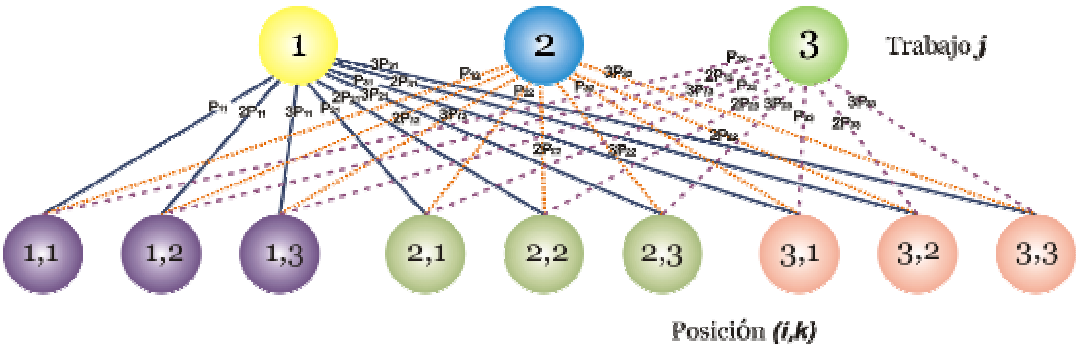


Figura 2-3. *Grafo bipartita no dirigido para UPMP para una instancia de 3 x 3 [Pinedo, 2008].*

Dando una posible solución al modelado por un grafo bipartita, se tendría que la máquina i = 1 podría procesar el trabajo 2, donde se ejecutaría en primer lugar (P_{12}), en segundo lugar ($2P_{12}$) y finalmente ($3P_{12}$).

2.3 Descripción Conceptual del Problema de Máquinas en Paralelo no Relacionadas

El problema de Máquinas en Paralelo no Relacionadas puede ser descrito de forma general como el conjunto de *n* trabajos independientes que necesitan ser calendarizados en *k* posiciones correspondientes a *m* máquinas en paralelo no relacionadas, de forma que se cumpla la función objetivo del problema, que es minimizar el tiempo total de término de todos los trabajos, para ello se debe considerar que un trabajo no puede ser procesado mas de una vez, además de que los trabajos no pueden ser interrumpidos una vez que se han asignado [Pinedo, 2008]. Para realizar la asignación de los trabajos es necesario tomar en cuenta las siguientes características propias del problema, de las cuales algunas son tomadas de forma implícita:

1. Un trabajo no puede ser procesado más de una vez
2. No existen restricciones o precedencias entre trabajos diferentes.
3. Un trabajo puede ser procesado en cualquier máquina.
4. Cada posición puede ser ocupada a lo más por un trabajo (Restricción de Capacidad de Recursos).
5. El tiempo de procesamiento de cada trabajo varía de acuerdo a la máquina y posición al que sea asignado.
6. No se consideran interrupciones (preemptions) ni tiempos de preparación.

El tiempo en el que se termina el procesamiento de todos los trabajos para UPMP es mejor conocido como el tiempo total de término de todos los trabajos.

El UPMP hereda algunas de las características de su antecesor, y modifica otras por la misma naturaleza del problema. Tomando en cuenta esta definición conceptual, el problema de UPMP puede ser definido de manera formal como el conjunto de trabajos $J = \{1, 2, \ldots, n\}$ que necesitan ser calendarizados en las posiciones $K = \{1, 2, \ldots, n\}$, correspondientes a cada una de las máquinas del conjunto $I = \{1, 2, \ldots, m\}$; donde no se permiten interrupciones y el tiempo de procesamiento P de cada trabajo j, depende de la máquina a la que el trabajo sea asignado, teniendo como función objetivo encontrar la calendarización que minimice el tiempo total de término de todos los trabajos.

Las características antes mencionadas encajan de manera adecuada con los requerimientos de los sistemas de manufactura utilizados actualmente en la industria, ya que mientras la demanda va creciendo, los requerimientos en cuanto a maquinaria se refiere se hacen mayores, lo que hace indispensable la adquisición de equipo nuevo; ésta es la razón principal por lo que las capacidades de las máquinas son diferentes; esta característica es parte fundamental del problema, ya que se toman en cuenta las capacidades de las máquinas para llevar a cabo una calendarización eficiente y de esta forma evitar el gasto excesivo e innecesario de adquisición de maquinaria y equipo.

2.4 Modelo Matemático de Máquinas en Paralelo no Relacionadas

El problema de Máquinas en Paralelo no Relacionadas, es un modelo de tipo NP [Garey et al., 1976] que puede ser formulado por medio de Programación Lineal Entera Binaria, lo cuál permite tratar al problema por medio de métodos no determinísticos, mejor conocidos como heurísticas [Papadimitriou y Steiglitz, 1998]. El modelo de programación lineal entera binaria presenta al problema como un conjunto *n* de trabajos que deben ser procesados en un conjunto *m* de máquinas, las cuales cuentas con *k* posiciones cada una. Cada trabajo puede ser procesado en cualquier máquina y cada máquina puede procesar cualquier trabajo, donde el tiempo de procesamiento de cada trabajo dependerá de la máquina y la posición en la que éste sea ejecutado, tomando en cuenta las restricciones básicas del problema, mismas que se muestran en la formulación matemática y se enlistan a continuación.

(2) Cada trabajo *j* puede ser procesado solo una vez.

(3) Cada posición *k* de cada máquina *i*, puede procesar a lo más un trabajo.

(4) Indica si un trabajo *j* fue asignado ó no a la posición *k* de una máquina *i*.

A continuación, en la figura 2-4 se muestra la formulación matemática utilizada para el problema de UPMP.

$$\min f = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} kP_{ij} X_{ikj} \quad (1)$$

Sujeto a:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} X_{ikj} = 1 \quad j = 1,\ldots,n \quad (2)$$

$$\sum_{j=1}^{n} X_{ikj} \leq 1 \qquad i = 1,\ldots,m,\ k = 1,\ldots,n \quad (3)$$

$$X_{ikj} \in \{0,1\} \qquad i = 1,\ldots,m,\ k = 1,\ldots,n,\ j = 1,...,n \quad (4)$$

Figura 2-4. *Modelo de Programación Lineal Entera Binaria para el problema de Máquinas en Paralelo no Relacionadas [Pinedo, 2008].*

El problema de Programación Lineal Entera Binaria presentado en Pinedo, 2001 (Figura 2-4), formula la función objetivo del problema con la expresión 1, para minimizar el tiempo total de término de todos los trabajos, de modo que al asignar un trabajo *j*, el tiempo de procesamiento dependerá de la posición *k*, correspondiente a la máquina *i*, al que éste sea asignado. Por lo que el tiempo de procesamiento de cada trabajo asignado kP_{ij} contribuye de forma directa al valor de la función objetivo.

El conjunto de restricciones representado por la expresión 2 asegura que cada trabajo sea procesado solamente una vez, de modo que antes de asignar un trabajo, es necesario revisar en todas las posiciones de las máquinas, que dicho trabajo no haya sido calendarizado previamente.

El conjunto de restricciones descrito por la expresión (3), garantiza que cada posición *k* de cada máquina *i* procese a lo más un trabajo *j*. De modo que una posición fue utilizada, ésta es marcada, a modo de señalar que dicha posición no puede volver a ser utilizada.

En el caso de la expresión (4), ésta indica si un trabajo *j* fue calendarizado en una posición *k* correspondiente a una máquina *i*. Donde X_{ikj}, solo puede tener valores de tipo binario, es decir 0 ó 1, de forma que si $X_{ikj}=1$, el trabajo fue asignado a la posición *k* de la máquina *i*, de lo contrario $X_{ikj}=0$.

Este modelo se representa por medio de un grafo bipartita, donde uno de los subconjuntos corresponde a los *n* trabajos y el otro a las *nm* posiciones, donde cada máquina puede procesar hasta *n* trabajos, es decir, que el número de posiciones correspondientes a cada máquina está en función del número de trabajos. El tiempo de procesamiento de cada trabajo, como ya se mencionó anteriormente, se encuentra en base a la posición y la máquina al que éste sea asignado, de modo que sí un trabajo *j* es asignado a *ik,* es decir, a la posición *k* de una máquina *i,* se tiene que el costo corresponde a $kP_{ij,}$ impacta directamente en el valor de la función objetivo.

La formulación presentada en la Figura 2-4 corresponde a un modelo relajado, debido a que UPMP se deriva del Problema de Calendarización de Máquinas en Talleres de Manufactura, del cuál se relajaron algunas de las restricciones para obtener un mapeo a este problema, por lo que es posible el uso de un algoritmo exacto como es el caso de Simplex Clásico utilizando el método de variables

artificiales, para encontrar una solución al modelo presentado, cabe mencionar que esto es posible solo para instancias pequeñas del problema, debido a la demostrada complejidad del método Simplex Clásico [Klee y Minty, 1972], la cuál se ha comprobado es de orden exponencial.

Capítulo 3

Algoritmo Colonia de Hormigas

3.1 Introducción

En este capítulo se propone una representación del problema de Máquinas en Paralelo no Relacionadas para Colonia de Hormigas por medio de un grafo disyuntivo, además, se presenta un algoritmo Colonia de Hormigas secuencial, así como el desarrollo e implementación de una estructura de vecindad híbrida aplicada en búsqueda local para resolver el problema de Máquinas en Paralelo no Relacionadas. Se incluye una amplia descripción del procedimiento utilizado para el desarrollo de la estructura de vecindad híbrida mencionada con anterioridad y finalmente se anexa la función temporal del algoritmo y su complejidad.

Para llevar a cabo la selección de las estructuras de vecindad que conforman la estructura híbrida, se llevó a cabo una búsqueda en diversas publicaciones que abordan el problema tratado (Capítulo 2), para, de esta forma, identificar aquellas estructuras que cuentan con menor complejidad computacional y que han obtenido buenos resultados en su aplicación.

Como primer paso se desarrolló el algoritmo Colonia de Hormigas para resolver el problema de Máquinas en Paralelo no Relacionadas, para mejorar la solución obtenida por dicho algoritmo, se llevó a cabo la implementación de una estructura de vecindad híbrida a búsqueda local. En las secciones

siguientes se explica ampliamente el procedimiento realizado para el desarrollo de dicha estructura.

3.2 Algoritmo Colonia de Hormigas

El Algoritmo Colonia de Hormigas es una metaheurística inspirada en el comportamiento inteligente y estructurado de las hormigas reales, la cuál fue propuesta por Marco Dorigo en 1991 y fue aplicada por primera vez al problema del Agente Viajero [Dorigo et. al., 1991a, 1991b], [Dorigo, 1992], [Dórigo et. al. 1996], [Dorigo et. al., 2004].

Métodos bioinspirados como Colonia de Hormigas han motivado a muchos investigadores a tratar de entender la interacción que existe entre las hormigas para construir ordenadas comunidades, además de llevar a cabo una especie de comunicación indirecta, denominada *Stigmergy* [Grassé, 1959], la cuál describe una comunicación entre los individuos de una colonia por medio de elementos o sustancias químicas.

De esta forma, las hormigas van explorando de forma iterativa diversos caminos, hasta llegar a donde se encuentra el alimento; mientras cada hormiga se encuentra moviéndose de un lugar a otro, cada una va depositando a su paso una sustancia química denominada *feromona*. Este rastro hace que el camino sea más deseable a las demás hormigas, es decir, que tendrá mayor probabilidad de que las demás hormigas se vean atraídas hacia los caminos con mayor monto de feromona, así mismo se lleva a cabo una evaporación natural de esta sustancia de acuerdo al tiempo y la distancia, por lo que al final, el camino más corto será el que contenga mayor intensidad de feromona y por ende el más transitado. A continuación se

describe el proceso por medio del cuál las hormigas son capaces de construir el camino más corto del nido a la fuente de alimento.

Al inicio del procedimiento, se tiene una distribución de las hormigas en cada uno de los nodos para iniciar el proceso de construcción de soluciones.

Cada una de las hormigas cuenta con una *lista tabú*, la cuál funciona como la memoria de la hormiga, es decir, va almacenando los nodos recorridos. Además, cada hormiga cuenta con un registro de aquellos nodos que no han sido visitados, esto se realiza con la finalidad de que las soluciones construidas cumplan con las restricciones especificadas en la formulación matemática del problema tratado (Capítulo 2).

Mientras cada hormiga va recorriendo el conjunto de nodos permitidos, va dejando a su paso un rastro de feromona, de modo que mientras más transitado sea un camino, mayor será el monto de feromona y por ende se evitará su rápida evaporación. En este proceso, la feromona tiene un papel fundamental, debido a que al llevar a cabo la evaporación se genera una especie de olvido para las hormigas, con lo que se evita la convergencia prematura del algoritmo. De modo que, cuando una hormiga encuentra un buen recorrido, este tendrá mayor probabilidad de ser elegido por otras hormigas, por lo que el rastro de feromona se va reforzando en los caminos más cortos, haciendo finalmente que el mejor recorrido sea encontrado. A continuación, en la figura 3-1 se muestra un ejemplo del recorrido que realizan las hormigas artificiales para la exploración y búsqueda de soluciones.

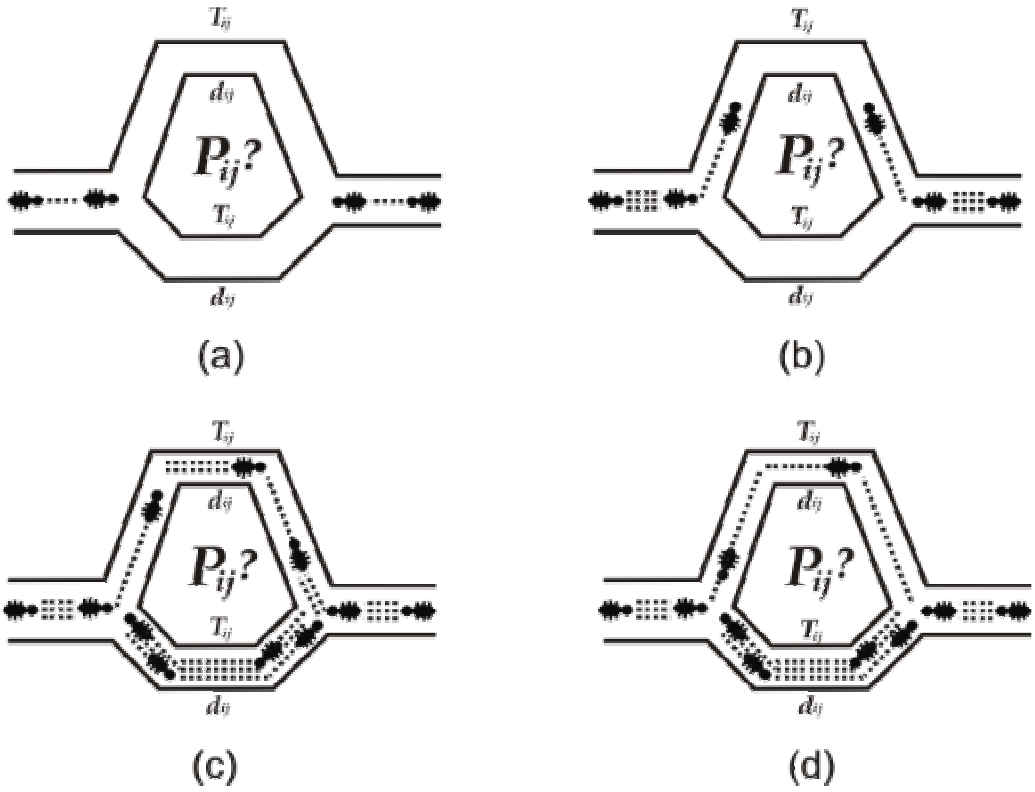


Figura 3-1. *Ejemplo del recorrido realizado por las hormigas, donde se evalúa la probabilidad de transición de un nodo a otro. [Dorigo et. al. 1996].*

Como se puede observar en la figura 3-1, en (*a*) cuando t = 0 (iteración = 0), no existe aún un rastro de feromona que las hormigas artificiales puedan seguir, así que comienzan con una exploración aleatoria, conforme el proceso de construcción de soluciones se va llevando a cabo, (b), (c) y (d), los montos de feromona van cambiando, lo que hace que la hormiga pueda realizar una evaluación de los caminos y tomar decisiones probabilísticas de acuerdo a los criterios establecidos en la formulación matemática y a las características propias del algoritmo Colonia de Hormigas.

Para tener una idea más clara del procedimiento explicado anteriormente en la figura 3-1 correspondiente al proceso de construcción de soluciones, se presenta en la figura 3-2 un diagrama de funcionamiento que muestra de

forma esquemática los procesos que se efectúan durante la ejecución del algoritmo Colonia de Hormigas Clásico, donde al principio se inicializan las hormigas y los montos de feromona, posteriormente cada hormiga va construyendo una solución de manera probabilística, hasta que la hormiga llega a la fuente de alimento, es decir, hasta que cada hormiga construye una solución de acuerdo a las características y restricciones propias del problema. Durante la ejecución del algoritmo, en cada iteración se llevan a cabo:

- *Incremento de feromona:* Se lleva a cabo de manera local, es decir, cada hormiga incrementa la feromona en los arcos almacenados en su lista tabú.
- *Evaporación:* Proceso global que se realiza en todos los arcos, independientemente de si se han recorrido o no.
- *Actualización de la feromona:* Se realiza únicamente en los arcos correspondientes a la mejor solución hasta el momento.

Cabe mencionar que las dos últimas acciones son llevadas a cabo por un proceso denominado para este trabajo de investigación como *Administrador de Acciones*. Todo el procedimiento explicado anteriormente, se realiza de forma iterativa hasta que el mejor camino a la fuente de alimento sea encontrado.

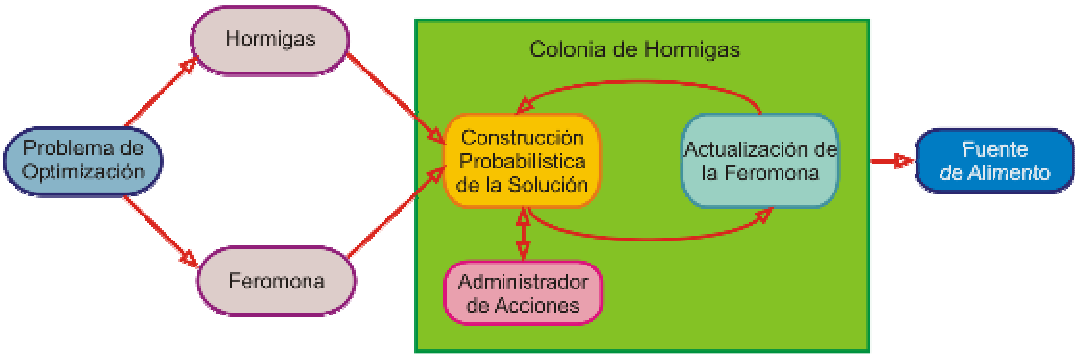


Figura 3-2. *Diagrama esquemático del Algoritmo Colonia de Hormigas.*

Para mejorar la calidad de las soluciones encontradas, diversos autores han recurrido al uso de estructuras de vecindad aplicadas a búsqueda local [Guo et al., 2007], [Gómez et al., 2007], [Zhou et al., 2007], [Zhou et al., 2007], [Pei y Shih, 2007], [Chun, 2008], [Chun, 2008] y [Mateo, 2009].

3.3 Estructura de Vecindad Híbrida

Todo problema de optimización tiene un conjunto, ya sea finito o infinito de soluciones posibles, por lo que se requiere la utilización de técnicas que permitan la mejor explotación del espacio de soluciones, y por ende obtener soluciones de buena calidad. Este tipo de técnicas aplicadas a búsqueda local son mejor conocidas como estructuras de vecindad.

El funcionamiento de las estructuras de vecindad es iterativo, debido a que permiten la mejor explotación del espacio de soluciones, de modo que define la vecindad así como la forma de acceder a soluciones vecinas de una solución *s* mediante una operación denominada *movimiento*, donde el tipo de movimiento aplicado define la estructura y tamaño de una vecindad.

Las estructuras de vecindad se han implementado ampliamente a búsqueda local en diversos métodos heurísticos para tratar problemas de optimización, los cuales, de acuerdo a la naturaleza de los mismos, llegan a ser intratables mediante técnicas de cálculo determinístico, debido a su complejidad; o bien se han implementado para minimizar el tiempo computacional requerido para resolver este tipo de problemas.

En este tipo de técnicas se sigue un camino dirigido por pasos estocásticos con la finalidad de llevar a cabo una mayor explotación del espacio de soluciones. Para mejorar una solución, es necesario realizar una búsqueda local, además de moverse dentro de una vecindad desde una solución inicial factible hacia una solución vecina, la cuál debe ser evaluada de acuerdo al criterio establecido por la función objetivo, misma que generalmente involucra costos (a maximizar o minimizar), con la finalidad de encontrar una solución que mejore el valor de la solución anterior.

Una *Vecindad* de una solución es el conjunto de soluciones que pueden ser alcanzables a partir una solución *s* por medio de un movimiento σ [Papadimitriou y Steiglitz, 1998] y [Martínez, 2006], este movimiento puede ser un intercambio, inserción o eliminación entre los elementos de una solución *s*. De acuerdo a esto, una vecindad se puede definir de la siguiente manera.

$$N(S) = \{s' \in S : s \xrightarrow{\sigma} s'\}$$

El punto fundamental para explotar el espacio de soluciones, consiste en empezar desde un punto factible *s,* del conjunto de soluciones en *N*(*s*) se elige mediante un movimiento una solución vecina *s'* que mejore *C (mejor valor de la función objetivo encontrado hasta el momento)*, esto es *C(s')* < *C(s)*, con esto, se hace un reemplazo de la solución *s* y se posiciona *s'* en este nuevo punto tal y como se puede observar en la figura 3-3, de modo que la estructura de vecindad es la función de vecindad o tipo de movimiento que aplicado a búsqueda local, permite mejorar la explotación del espacio de soluciones [Gu, Huang, 1994] y [Stattenberger et al., 2007].

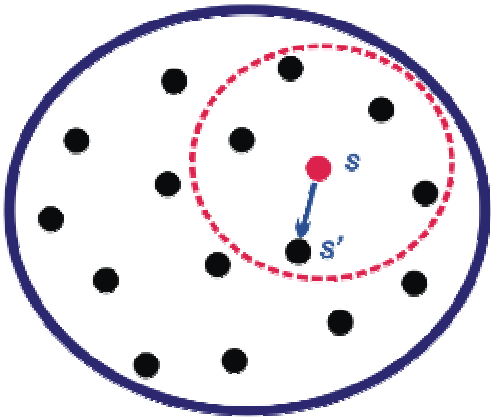


Figura 3-3. *Representación del espacio de soluciones con una vecindad N(s) = {s'}.*

Un aspecto crítico del diseño de algunos algoritmos de optimización es la elección de una estructura de vecindad adecuada, es decir, elegir aquella estructura de vecindad que permita la mejor explotación del espacio de soluciones de acuerdo al algoritmo utilizado y al problema tratado.

Para determinar el funcionamiento de una estructura de vecindad, se debe definir el criterio de selección de un vecino, es decir, el movimiento que se llevará a cabo para alcanzar una solución *s'* desde una solución *s,* de modo que si el criterio de selección se cumple, se lleva a cabo el movimiento, el proceso se repite hasta que la solución encontrada no pueda ser mejorada, por lo que se dice que se ha llegado a un *óptimo local*. A continuación, en la Figura 3-4 se muestra el algoritmo general de una búsqueda por vecindad.

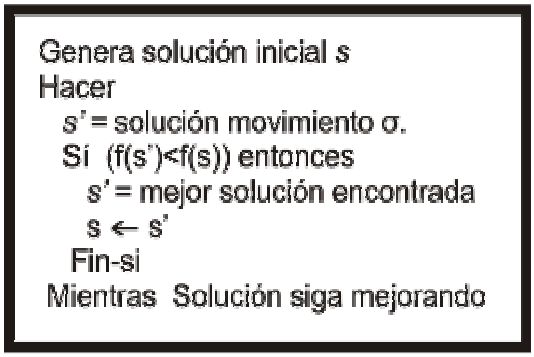


Figura 3-4. *Algoritmo general de una búsqueda por Vecindad.*

Se dice que *s* es un óptimo local si no se encuentra una solución en *N(s)* que la mejore. La idea básica de la estructura de vecindad es que a partir de una solución inicial *s*, se explore y evalúe a sus vecinos *N(s),* para encontrar una nueva mejor solución *s' ϵ N(s).*

Para desarrollar una estructura de vecindad híbrida, se realizó una revisión en la literatura, para buscar las estructuras de vecindad que contaran con baja complejidad computacional y que hayan obtenido buenos resultados en su aplicación. Las pruebas experimentales se realizaron para el problema del Agente Viajero. Se tomó este problema debido a que aunque es un problema más complejo comparado con el UPMP, es más sencillo en cuanto a la implementación de una estructura de vecindad. A continuación se mencionan las características (movimientos realizados) de cada una de las estructuras de vecindad, de acuerdo al UPMP.

- ***Un Par Adyacente***. Este procedimiento, aplicado a búsqueda local, elige aleatoriamente una posición perteneciente a la mejor solución hasta el momento, de modo que el trabajo correspondiente a dicha posición es permutado con la posición (trabajo) inmediato siguiente. Finalmente la búsqueda local calcula el costo de la nueva solución y lo

compara con el obtenido anteriormente, si este es mejor, (dependiendo de la función objetivo) se toma como el nuevo valor provisional de la función objetivo hasta el momento, de lo contrario se mantiene el costo anterior. Este procedimiento es realizado hasta que la solución ya no sea mejorada o se cumpla el criterio de paro. En la figura 3-5 se muestra el movimiento realizado por la estructura de vecindad utilizada.

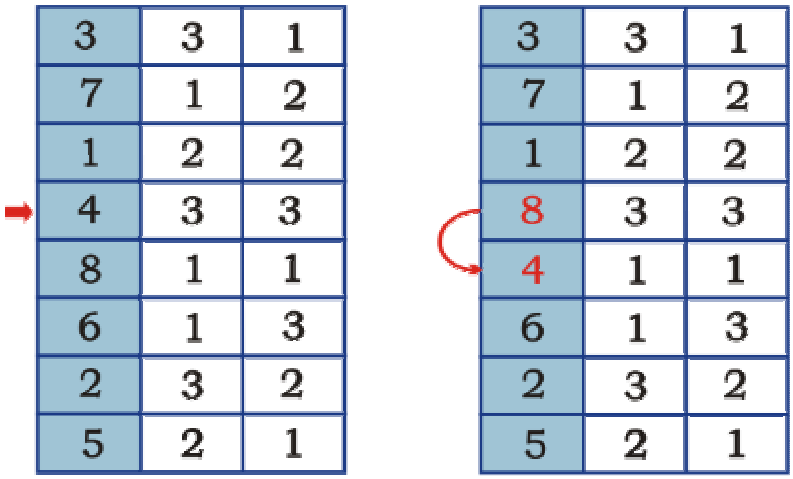


Figura 3-5. *Movimiento realizado por la estructura de vecindad de un par adyacente.*

- ***Un Par Aleatorio***. Al igual que la estructura de un par adyacente, los trabajos a permutar se eligen de forma aleatoria, la diferencia radica en que el número de trabajos a elegir son dos, y estos deben cumplir con ciertas restricciones, las cuales son, no ser el mismo trabajo y no ser adyacentes ni a la derecha ni a la izquierda. Una vez terminadas las funciones de la estructura de vecindad, la búsqueda local compara los costos obtenidos. Todo este proceso se lleva a cabo hasta que la solución ya no sea mejorada o se cumpla el criterio de paro. La figura

3-6 muestra el movimiento realizado por la estructura de vecindad empleada.

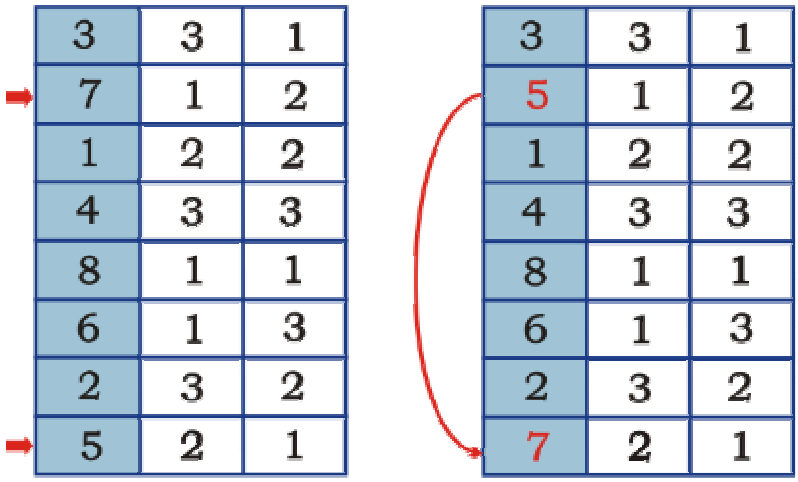


Figura 3-6. *Movimiento realizado por la estructura de vecindad de un par aleatorio.*

- ***Dos Pares Adyacentes***. Esta técnica inicialmente elige un posición de forma aleatoria, correspondiente a un trabajo de la mejor solución hasta el momento y la permuta con el trabajo inmediato siguiente, de la misma forma, elige una segunda posición, la cuál es comparada con la elegida en la primera permutación, ya que no debe ser la misma posición, ni adyacente a la derecha ni a la izquierda. Al término de las funciones de la estructura de vecindad, la búsqueda local compara la calidad de las soluciones obtenidas con la mejor hasta el momento. Este procedimiento es repetido de forma iterativa hasta que la solución ya no sea mejorada o bien, se cumpla el criterio de paro. En la figura 3-7 se observa el movimiento realizado por la estructura.

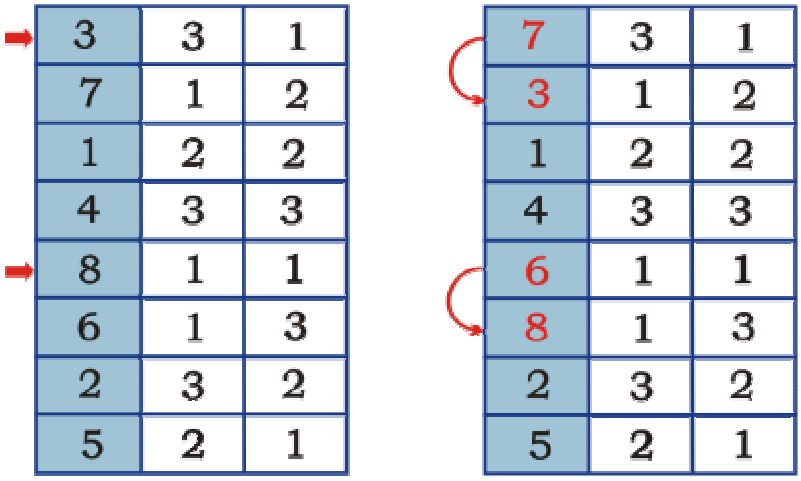


Figura 3-7. *Movimiento realizado por la estructura de vecindad de dos pares adyacentes.*

- ***Dos Pares aleatorios***. Esta estructura de vecindad tiene una complejidad computacional mayor en comparación con las explicadas anteriormente, debido a que requiere elegir cuatro trabajos de forma aleatoria, los cuales son comparados entre si para evitar que sea el mismo trabajo o bien que sean adyacentes entre sí, tal y como se muestra en la figura 3-8.

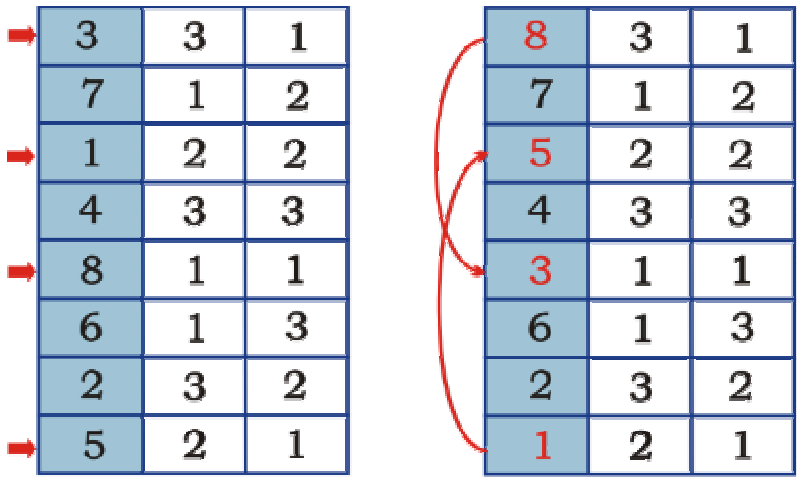


Figura 3-8*. Movimiento realizado por la estructura de vecindad de dos pares aleatorios.*

Tomando en cuenta las ventajas de las estructuras presentadas, así como su desempeño reportado en la literatura, se propuso el desarrollo de una *estructura de vecindad híbrida* que permitiera la incorporación de las ventajas de cada una de las cuatro estructuras, de modo que los movimientos realizados por la estructura de vecindad híbrida interactúan de forma aleatoria. A continuación, en la figura 3-9 se explica en que consisten.

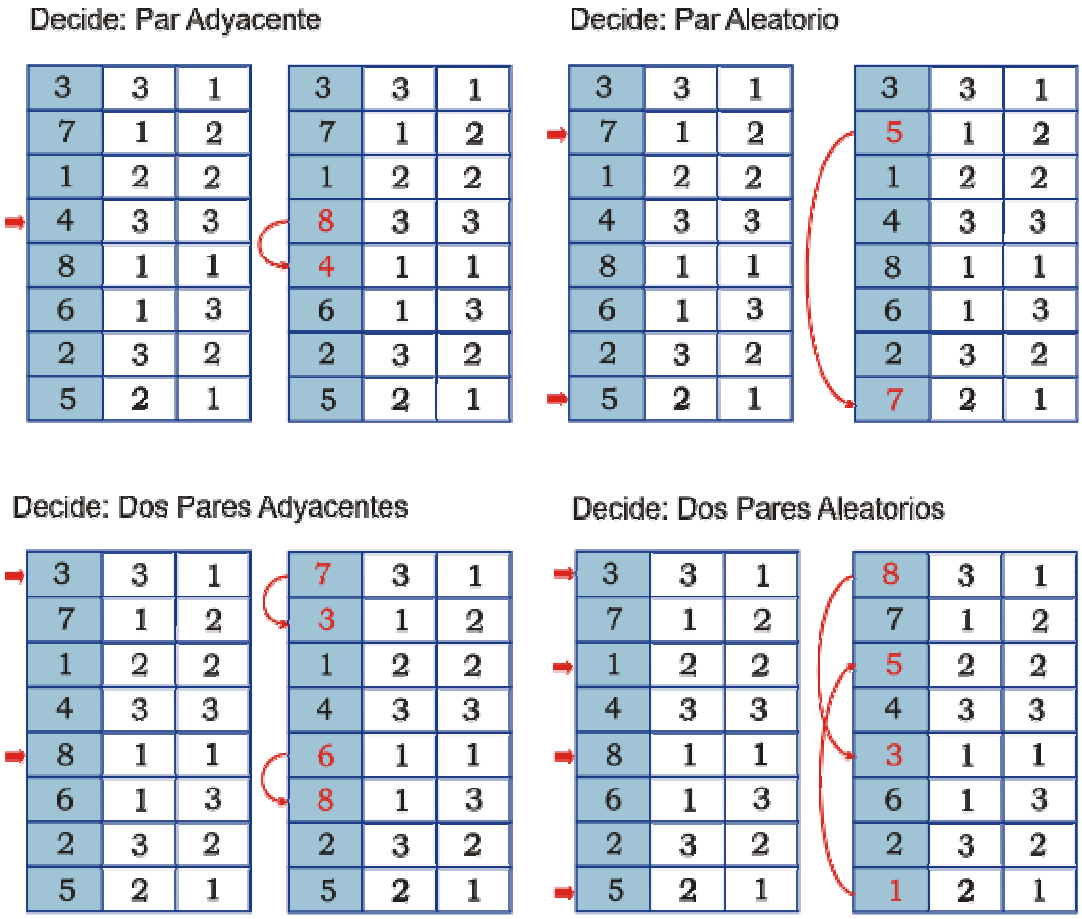


Figura 3-9. *Movimientos realizados por la estructura de vecindad híbrida. Antes de realizar cualquier intercambio, el algoritmo decide de forma aleatoria el tipo de estructura a aplicar.*

Inicialmente, la búsqueda local por cooperación de las hormigas, utilizada en el método de solución propuesto para el UPMP, obtiene una mejor solución

hasta el momento, tomando la solución obtenida al término de la primera iteración. Posteriormente se procede a elegir de forma aleatoria el módulo (tipo de estructura) a aplicar (de las cuatro explicadas con anterioridad), de forma que en cada iteración se vaya explotando el espacio de soluciones, es decir se van obteniendo soluciones vecinas, las cuales requieren ser evaluadas de acuerdo a la función objetivo y restricciones del problema.

La característica principal de la estructura híbrida propuesta aplicada a búsqueda local, consiste en incorporar las características básicas de las cuatro estructuras sencillas explicadas anteriormente [Cruz et al., 2010b], las cuales interactúan de forma aleatoria, hasta que se cumpla el criterio de paro establecido (número máximo de iteraciones). En la figura 3-10 se muestra el funcionamiento de forma general de la estructura híbrida.

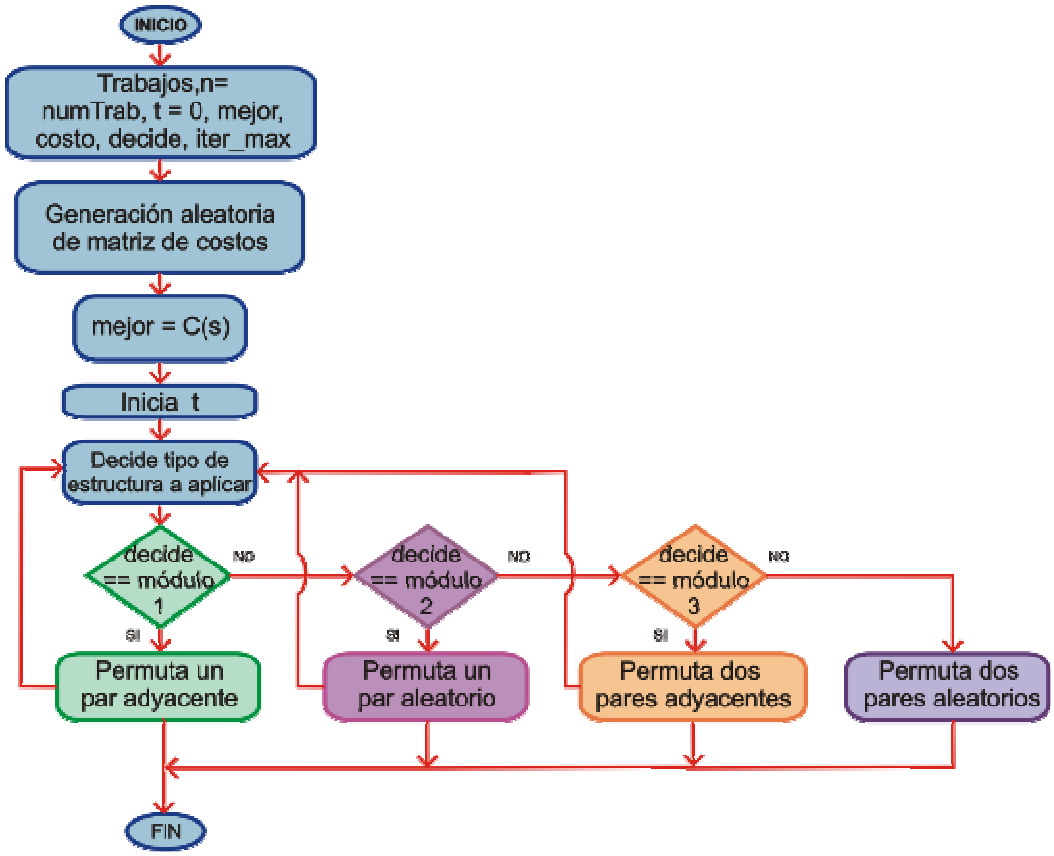


Figura 3-10. *Diagrama general de la estructura híbrida propuesta.*

Tomando en cuenta el diagrama mostrado por la figura 3-10, se tiene el tipo de estructura a aplicar en cada iteración durante la ejecución de la estructura híbrida se lleva a cabo de forma estocástica. A continuación se explican cada uno de los módulos involucrados en la estructura híbrida aplicada a búsqueda local, donde el primer modulo corresponde a la estructura sencilla de una permutación de un par adyacente y cuyo funcionamiento se muestra en el diagrama de la Figura 3-10a.

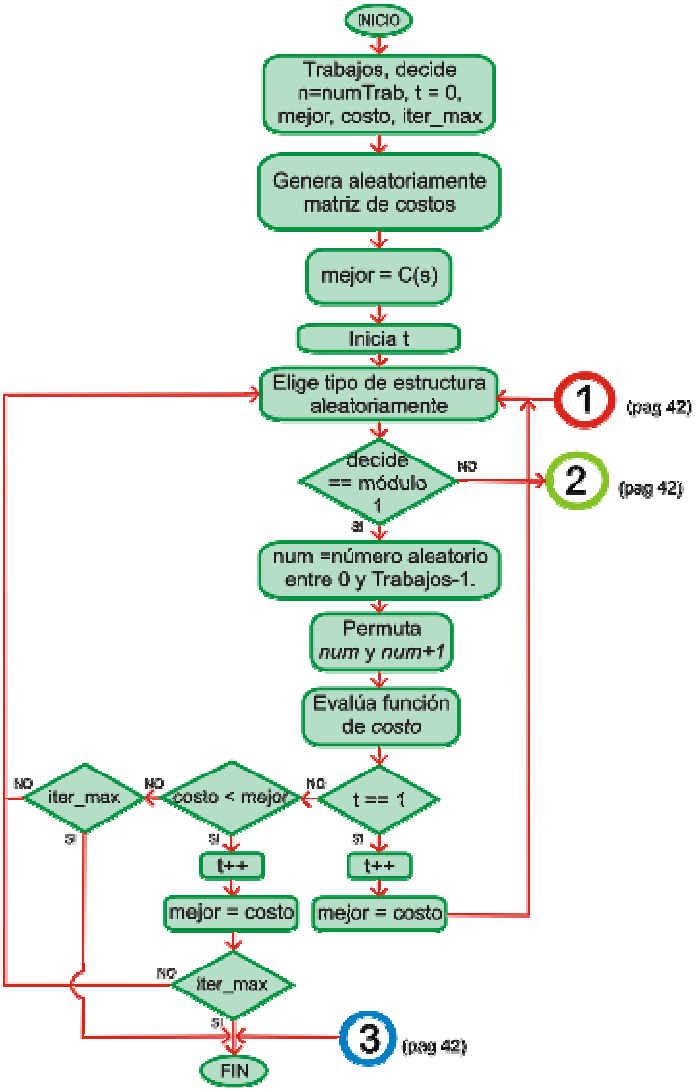


***Figura 3-10a**. Estructura híbrida. A) Módulo perteneciente a una estructura sencilla de una permutación adyacente aplicada a búsqueda local.*

En caso de que se elija el segundo módulo, se aplica a la búsqueda local una estructura de vecindad de un par aleatorio. La figura 3-10b muestra el diagrama de flujo correspondiente al funcionamiento del segundo módulo de la estructura híbrida.

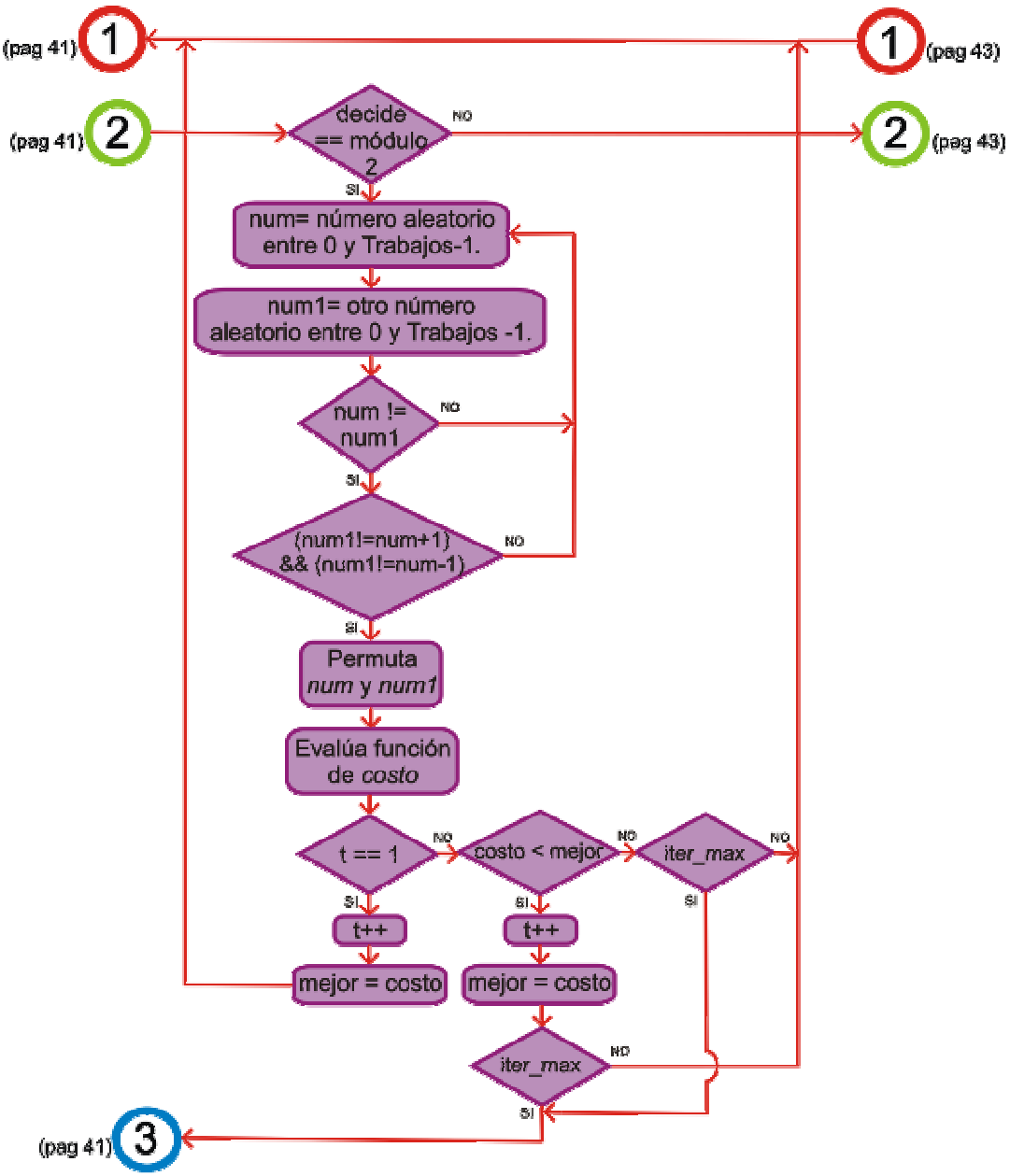


***Figura 3-10b**. Estructura híbrida de vecindad. B) Correspondiente a una estructura sencilla de una permutación aleatoria aplicada a búsqueda local.*

Para el caso de que el módulo elegido sea el tercero, el movimiento a realizar será el correspondiente a dos permutaciones de pares adyacentes. La figura 3-10c muestra el funcionamiento del tercer módulo.

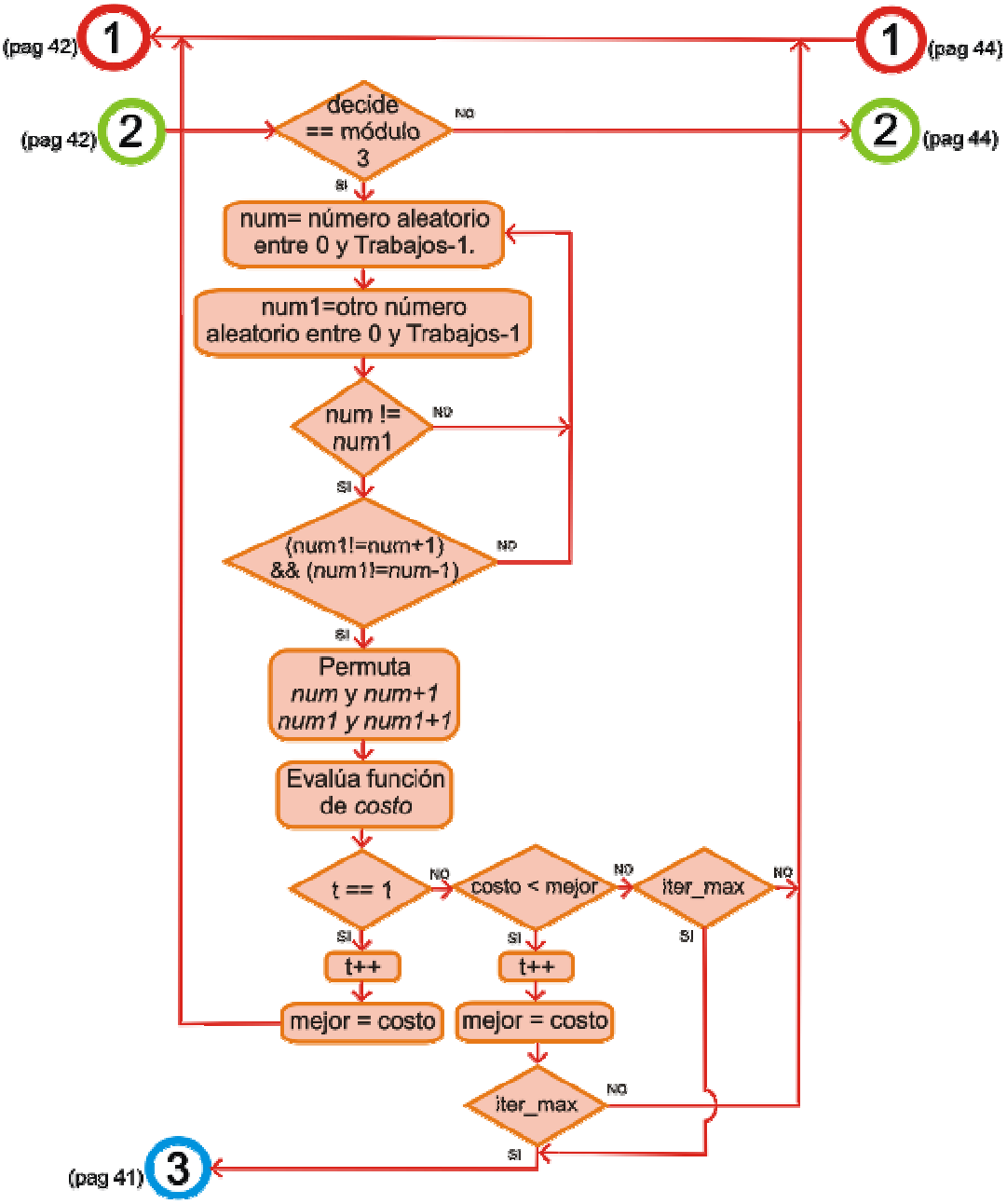


Figura 3-10c11. *Estructura híbrida de vecindad. C) Parte correspondiente a una estructura de dos permutaciones adyacentes aplicada a búsqueda local.*

El cuarto módulo se ejecuta cuando la opción elegida no corresponda a ninguna de las mencionadas anteriormente, y su funcionamiento correspondiente a dos permutaciones de pares aleatorios aplicados a búsqueda local (figura 3-10d).

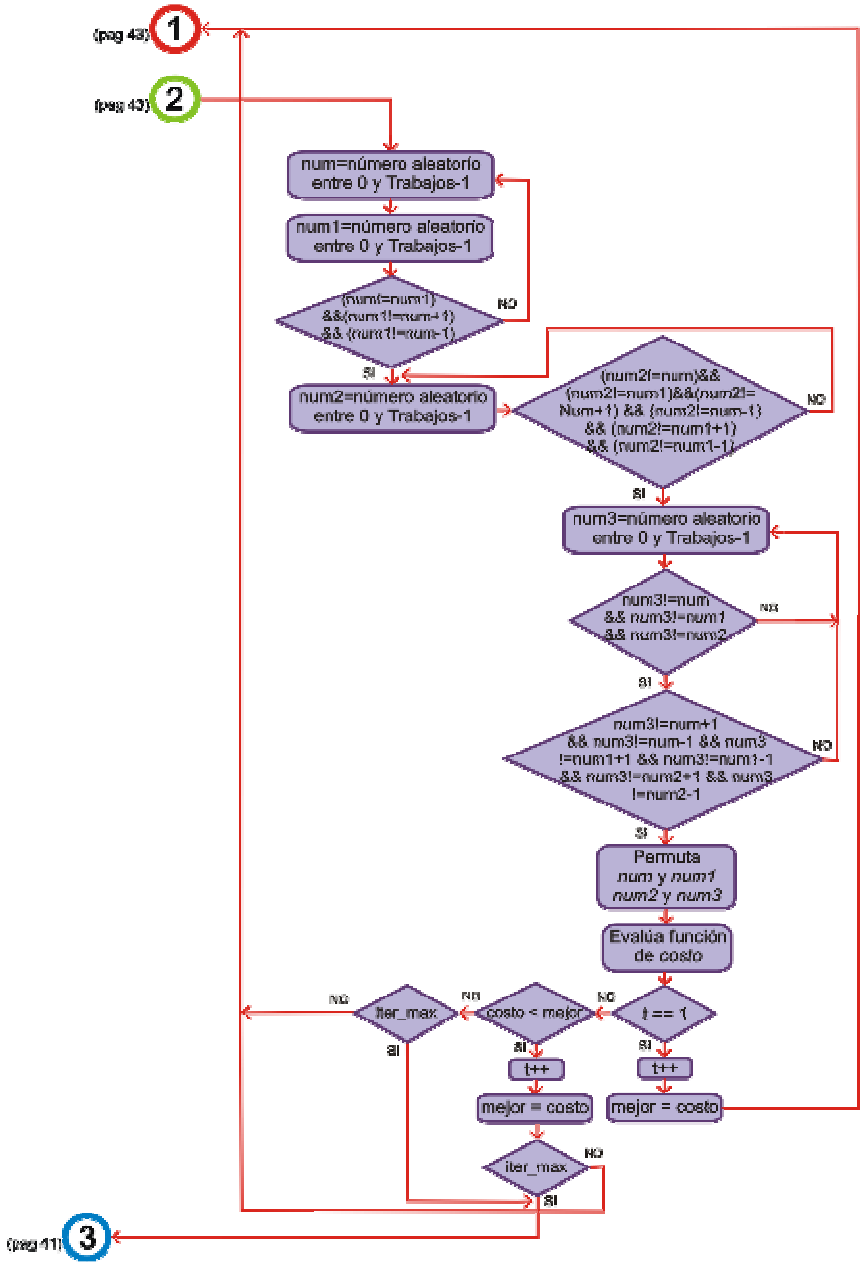


***Figura 3-10d**. Estructura híbrida de vecindad. D) Parte correspondiente a una estructura de dos permutaciones aleatorias aplicada a búsqueda local.*

De acuerdo a los excelentes resultados obtenidos en [Cruz et al., 2010b] para el problema del Agente Viajero, donde se vio claramente que una estructura híbrida aplicada a búsqueda local obtiene resultados de mejor calidad que una estructura sencilla, se decidió aplicar la estructura híbrida a búsqueda local para un algoritmo Colonia de Hormigas, esperando obtener buenos resultados al ser aplicado al UPMP.

Cabe mencionar que las estructuras de vecindad explicadas anteriormente fueron desarrolladas en Visual C++ 2008, sobre Windows Vista Home Premium.

3.4 Generación Aleatoria de Instancias de Prueba

Las instancias (matrices de costos), fueron generados de forma aleatoria, debido a que aunque en la literatura existen benchmarks para UPMP, éstas son instancias muy pequeñas, ya que la instancia más grande es de 120 trabajos por 12 máquinas y el objetivo de este trabajo de investigación es el realizar pruebas con instancias más grandes.

Para generar las instancias, se desarrolló un programa en Visual C++ 2008, el cuál permite generar las instancias de manera aleatoria. Cabe mencionar que de acuerdo a la función objetivo del problema tratado, además de la máquina, la posición también juega un papel importante en cuanto al costo de un trabajo en cada máquina, característica que fue tomada en cuenta al momento de generar las instancias. Otro punto importante es que el número de posiciones de cada máquina está en función del número de trabajos. A continuación en la tabla 3-1 se muestra un ejemplo de una instancia pequeña de 5 trabajos por 3 máquinas generada para UPMP.

Tabla 3-1. *Matriz de Costos para UPMP correspondiente a una instancia de 5 x 3 generada aleatoriamente.*

		Máquina 1					Máquina 2					Máquina 3				
Trabajos	*1*	**3**	6	9	12	15	**5**	10	15	20	25	**4**	8	12	16	20
	2	**5**	10	15	20	25	**3**	6	9	12	15	**7**	14	21	28	35
	3	**2**	4	6	8	10	**7**	14	21	28	35	**1**	2	3	4	5
	4	**7**	14	21	28	35	**4**	8	12	16	20	**8**	16	24	32	40
	5	**4**	8	12	16	20	**2**	4	6	8	10	**6**	12	18	24	30

Los costos generados de forma aleatoria para cada trabajo, se encuentran en un rango de 1 a 15 unidades de tiempo y son los correspondientes a la posición 1 de cada máquina (columna sombreada de matriz de costos), y a partir de estos se calculan los valores de las demás posiciones, multiplicando el valor generado, por el número de la posición para la cuál se está calculando el valor, de esta forma se tiene que el costo (tiempo de procesamiento de un trabajo) se irá incrementando gradualmente dependiendo de la posición en la que sea procesado.

3.5 Esquema Generalizado del Algoritmo de Colonia de Hormigas

En esta sección se presenta una propuesta para representar el UPMP para Colonia de Hormigas por medio de un grafo disyuntivo, además de explicar el método de desarrollo del algoritmo Colonia de Hormigas para UPMP. Finalmente se muestra el diagrama de flujo correspondiente al funcionamiento de dicho algoritmo.

3.5.1 Representación del UPMP para Colonia de Hormigas

Para llevar a cabo el modelado del problema de Máquinas en Paralelo no Relacionadas, se propone una representación del problema para Colonia de Hormigas mediante la utilización de un grafo disyuntivo, mismo que es mostrado en la figura 3-11.

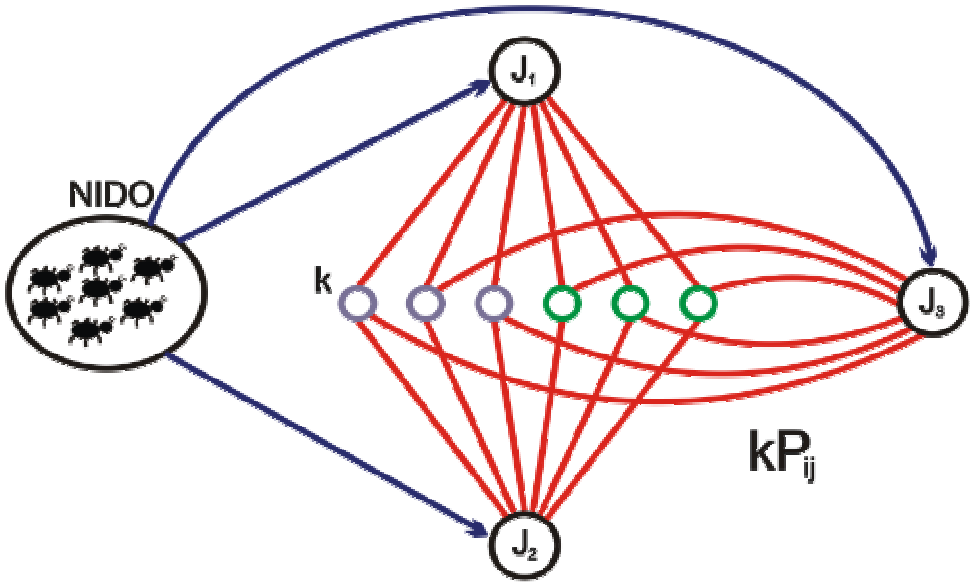


Figura 3-11*. Representación del UPMP para Colonia de Hormigas por medio de un grafo disyuntivo.*

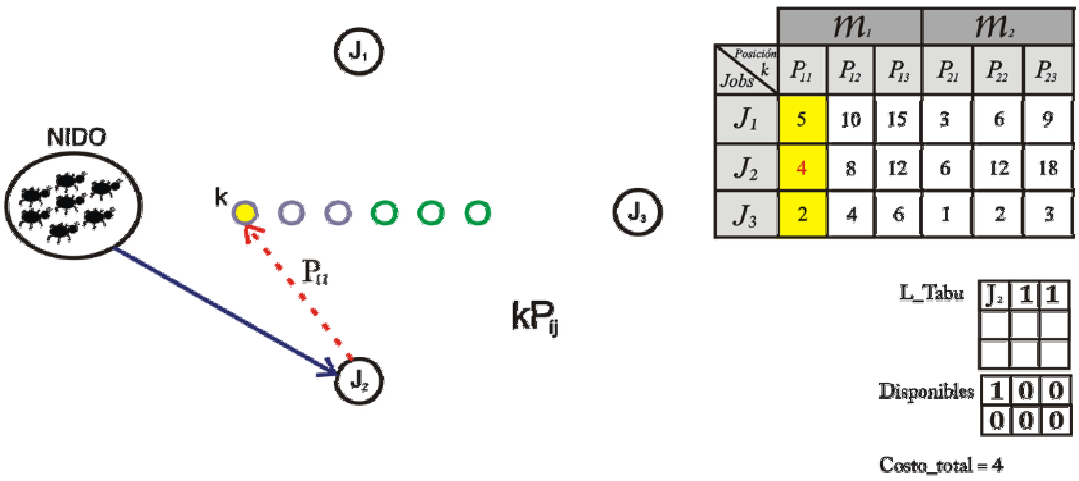
En esta representación se muestra que la elección de cada uno de los trabajos, se lleva a cabo de forma aleatoria sin ninguna ponderación, además de que para seleccionar una posición de una máquina, ésta tiene que ser evaluada de acuerdo a las restricciones propias del problema.

De acuerdo a lo anterior, cada hormiga cuenta con una estructura de datos que permite que los recorridos de las hormigas no violen ninguna de las restricciones especificadas en la formulación matemática (Lista Tabú, Matriz

de posiciones disponibles y Costo del recorrido), por lo que los pasos realizados por una hormiga para construir una solución a una instancia de 3 trabajos y 2 máquinas son ejemplificados en las figuras 3-12 a la 3-14. Cabe mencionar que para el caso de la matriz de costos, el número de posiciones en cada máquina, así como el número total de hormigas están en función del número de trabajos y los costos dependen directamente de la posición y máquina al que el trabajo sea asignado.

Paso 1:

Una hormiga elige un trabajo de forma aleatoria y lo agrega a su *lista tabú*, verifica la *matriz de posiciones disponibles* y de la *matriz de costos* elige aquella que conlleve menor costo, aunado a esto, marca la posición como ocupada para que no vuelva a ser utilizada por otro trabajo, lo cuál es una restricción del problema, además de que va almacenando el costo del recorrido en la variable definida *Costo_total*.

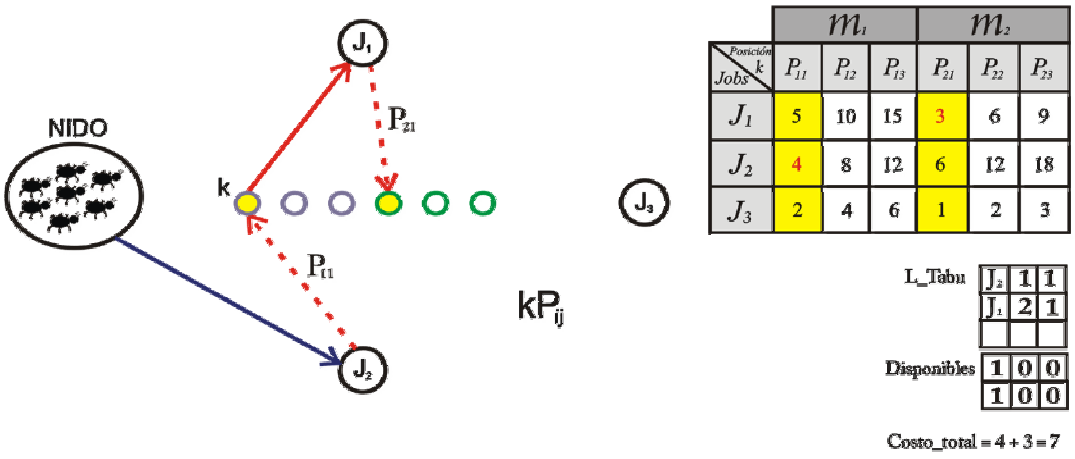


Jobs \ Posición k	m_1			m_2		
	P_{11}	P_{12}	P_{13}	P_{21}	P_{22}	P_{23}
J_1	5	10	15	3	6	9
J_2	4	8	12	6	12	18
J_3	2	4	6	1	2	3

Figura 3-12*. Paso 1. Recorrido de una hormiga para la construcción de una solución en un grafo disyuntivo de UPMP para Colonia de Hormigas.*

Paso 2:

La hormiga vuelve a elegir un trabajo de forma aleatoria y verifica en su *lista tabú* que dicho trabajo no se haya calendarizado previamente, de lo contrario elige otro trabajo y repite el procedimiento, si este trabajo no se ha calendarizado anteriormente, lo agrega a su *lista tabú* y revisa la *matriz de posiciones disponibles* y la *matriz de costos*, tomando aquella posición que conlleve el menor costo y que esté disponible, siendo así, marca la posición al igual que en el paso anterior e incrementa su *Costo_total* de acuerdo al costo de la posición seleccionada.



Posición k / Jobs	P_{11}	P_{12}	P_{13}	P_{21}	P_{22}	P_{23}
	m_1			m_2		
J_1	5	10	15	3	6	9
J_2	4	8	12	6	12	18
J_3	2	4	6	1	2	3

L_Tabu			
	J_3	1	1
	J_1	2	1

Disponibles			
	1	0	0
	1	0	0

Figura 3-13. *Paso 2. Recorrido de una hormiga para la construcción de una solución en un grafo disyuntivo de UPMP para Colonia de Hormigas.*

Paso 3:

En este paso, la hormiga vuelve a realizar el mismo procedimiento de selección llevado a cabo en el paso 2, de modo que una vez que la lista tabú se ha completado, la hormiga obtendrá el costo total correspondiente al recorrido realizado.

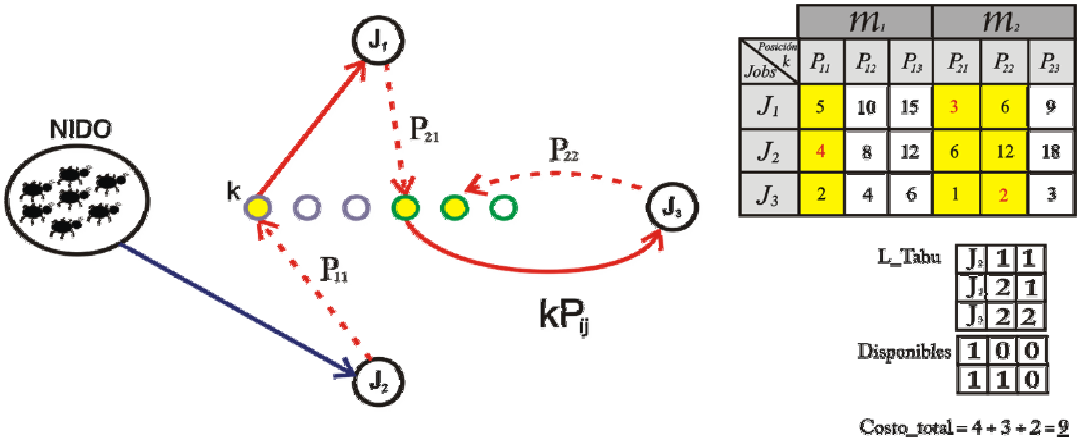


***Figura 3-14**. Paso 3. La hormiga completó un recorrido en el grafo disyuntivo de UPMP para Colonia de Hormigas y se tiene el Costo total del recorrido.*

3.6 Algoritmo Secuencial para UPMP

Para proponer una solución al problema de Máquinas en Paralelo no Relacionadas, se desarrolló un algoritmo Colonia de Hormigas secuencial, para el cual fue aplicada una estructura de vecindad híbrida a búsqueda local, misma que fue explicada anteriormente.

Dado un conjunto de trabajos y máquinas, el problema de Máquinas en Paralelo no Relacionadas debe encontrar la calendarización que conlleve el mejor valor de acuerdo a la función objetivo, tomando como base los *n* trabajos que deben ser calendarizados en *m* máquinas, mismo que es representado por medio de un grafo bipartita.

Para llevar a cabo el procedimiento de solución, inicialmente se genera una colonia de hormigas artificiales, las cuales están en función del número de

trabajos a calendarizar [Bullnheimer et al., 1997] y deben ser inicializadas. Cada una de las hormigas posee una *Lista Tabú,* la cuál indica a la hormiga los trabajos que ha calendarizado y que no pueden volver a ser procesados, lo que es una restricción del problema, la *Matriz de posiciones disponibles* muestra a la hormiga las posiciones utilizadas de cada una de las máquinas y las cuales no pueden volver a ser utilizadas, y *Costo del recorrido,* cuya función es ir almacenando el costo de cada uno de los trabajos calendarizados, de modo que al final de la calendarización, éste contendrá el costo total del recorrido en unidades de tiempo.

De la misma manera, la colonia de hormigas cuenta con una *matriz de feromona*, la cuál es inicializada al comenzar la ejecución del algoritmo. La matriz de feromona es de gran importancia ya que permite guardar los nuevos valores de la feromona para cada arco (relación que existe entre cada trabajo y cada una de las posiciones correspondientes a cada máquina). Estos nuevos valores se encuentran en función del incremento realizado por cada hormiga, así como de la actualización y evaporación de la feromona.

Inicialmente, de manera general, se tiene que la metaheurística es la interacción de tres acciones fundamentales:

- ***Construcción de Soluciones:*** Durante la construcción de soluciones cada una de las hormigas debe elegir de forma probabilística el siguiente nodo a visitar, para ello es necesario aplicar la ecuación correspondiente a la probabilidad de transición (*ecuación 1*). Cabe mencionar, que para el UPMP, el orden de los trabajos a calendarizar será elegido de forma aleatoria por cada una de las hormigas, lo cuál permite tener una mayor diversidad de soluciones y por ende una mejor exploración del espacio de soluciones.

$$P_{ij}^{A} = \frac{[\tau_{ij}(t)]^{\alpha}[1/d_{ij}]^{\beta}}{\sum_{j \in Np}[\tau_{ij}(t)]^{\alpha}[1/d_{ij}]^{\beta}} \quad (1)$$

Donde:

P_{ij}^{A} Probabilidad de transición de un nodo *i* a un nodo *j* para una hormiga *A*

τ_{ij} Monto de feromona en el arco (*i, j*).

d_{ij} Distancia heurística de un nodo *i* a un nodo *j* (costo en unidades de tiempo).

α Importancia relativa del monto de feromona.

β Importancia relativa de la distancia heurística.

Np Conjunto de trabajos permitidos

De acuerdo a la *ecuación 1*, es necesario verificar que el nodo elegido pertenezca al conjunto de nodos permitidos, así como evaluar cada camino con respecto a los demás, tomando en cuenta aquellos factores que juegan un papel determinante en la selección del siguiente nodo, estos factores son: la importancia relativa del monto de feromona (α) y la importancia relativa de la distancia heurística (β), los cuales son evaluados en base al monto de feromona y la distancia heurística (costo en unidades de tiempo) respectivamente.

Una vez que la hormiga ha elegido una posición de una máquina, es necesario evaluar si ésta no se ha utilizado previamente para la calendarización de otro trabajo, debido a que esto es una restricción del problema. Ya que la lista tabú de la hormiga se ha completado, se calcula el costo del recorrido realizado por la hormiga *A*.

- ***Actualización de la Feromona:*** Cada que una hormiga ha concluido su recorrido, se realiza un incremento local de feromona en los arcos incluidos en la lista tabú de la hormiga *A*. Este incremento se lleva a cabo de acuerdo a la calidad de la solución, aplicando las ecuaciones 2 y 3.

$$\tau_{ij}(t+N) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \tag{2}$$

Donde:

ρ Coeficiente de evaporación de la feromona

$\Delta\tau_{ij}$ Incremento de la feromona

$$\Delta\tau_{ij}^{A} = \begin{cases} Q/f_{solución} & \text{sí fue recorrido} \\ 0 & \text{en caso contrario} \end{cases} \tag{3}$$

Donde:

Q Cantidad de feromona por unidad de tiempo

$f_{solución}$ Calidad de la solución encontrada.

De esta forma se tiene que el monto total de feromona en un arco (τ_{ij}), estará dado por la cantidad de hormigas que transiten el mismo (Ecuación 4).

$$\Delta\tau_{ij} = \sum_{A=1}^{N} \Delta\tau_{ij}^{A} \tag{4}$$

Una vez que todas las hormigas han terminado de realizar un recorrido, se dice que ha finalizado una iteración (*t*). Por lo que se lleva a cabo una comparación de las soluciones encontradas por cada una de las hormigas, de modo que la mejor solución, de acuerdo a la función objetivo, será almacenada como el mejor valor hasta el momento, a esta nueva mejor solución obtenida, se aplica una estructura de vecindad utilizada en búsqueda local que permita mejorar el resultado de la función objetivo. De esta forma se llevan a cabo *t* número de iteraciones, comparando la mejor solución de cada iteración con el mejor valor de la función objetivo hasta el momento, de modo que si existe una solución que la mejore, el valor de la función objetivo es reemplazado.

Administrador de Acciones: Al término de cada iteración y una vez que se tiene la mejor solución de la función objetivo hasta el momento, se aplica un procedimiento llamado Administrador de Acciones, cuya función es desempeñar acciones globales que no pueden ser realizadas por las hormigas de forma individual. Algunas de estas acciones son: la aplicación de un procedimiento de búsqueda local con una estructura de vecindad a la mejor solución hasta el momento, la evaporación global de la feromona, así como la búsqueda

de información global que permita la toma de decisiones que afecten el resultado final del algoritmo.

- *Búsqueda Local:* A este procedimiento se aplica una estructura de vecindad que permita mejorar la calidad de la solución obtenida por el algoritmo de Colonia de Hormigas en cada iteración.

- *Evaporación de la Feromona:* Se aplica a todos los arcos correspondientes al grafo, al término de cada iteración, tomando en cuenta el valor del coeficiente de evaporación. La evaporación tiene la función de provocar una especie de olvido en las hormigas, lo que permite seguir explorando el espacio de soluciones y por ende evita la convergencia prematura del algoritmo.

- *Búsqueda de Información Global:* Toma la mejor solución encontrada durante cada iteración (t) y aplica nuevamente el procedimiento de búsqueda local con una estructura de vecindad, buscando la mejora de la solución. Una vez que se ha encontrado la mejor solución de una iteración, se aplica un monto extra de feromona a los arcos correspondientes al mejor valor de la función objetivo, ayudando a que estos arcos sean más atractivos a las demás hormigas.

Todo este proceso realizado para encontrar una buena solución cercana al óptimo, se lleva a cabo de forma iterativa hasta que se cumpla el número máximo de iteraciones (criterio de paro). De acuerdo a lo explicado anteriormente, en la figura 3-15 se muestra el diagrama de flujo correspondiente al funcionamiento del algoritmo Colonia de Hormigas propuesto.

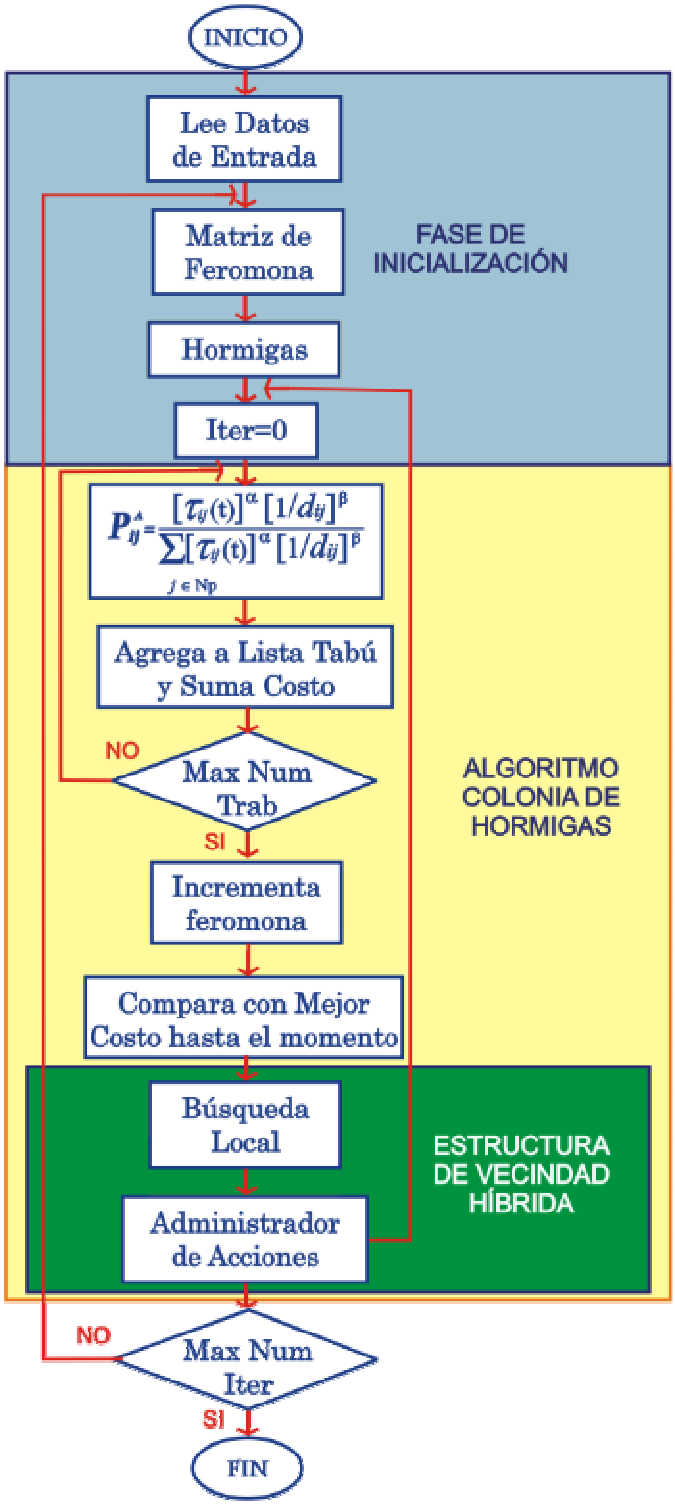


***Figura 3-15**. Diagrama de Flujo del Algoritmo Colonia de Hormigas aplicado al UPMP utilizando una estructura de vecindad híbrida.*

3.7 Metodología de Sintonización

Para obtener una adecuada sintonización de los parámetros de control de un algoritmo, es necesario llevar a cabo un análisis de sensibilidad.

El *análisis de sensibilidad* es una evaluación del comportamiento de las variables críticas de un problema (parámetros de control), con la finalidad de establecer un rango numérico, dentro del cuál la solución obtenida por el algoritmo sigue siendo buena, además de que permite conocer que tan sensible es el algoritmo a cambios en los valores de ciertas variables propias del problema.

El análisis de sensibilidad tiene su origen en la programación lineal (PL) debido a que facilita la toma de decisiones. Los cambios llevados a cabo durante dicha evaluación pueden ser en el entorno general del problema, en la empresa o bien en los datos característicos del problema.

El objetivo primordial del análisis de sensibilidad, es encontrar la adecuada sintonización de los parámetros de control del algoritmo, de modo que éste tenga una mejora en cuanto a la eficiencia y eficacia. Los valores más estudiados en este análisis de forma general, son los coeficientes de la función objetivo y los términos independientes de las restricciones.

A continuación se explica una metodología de sintonización propuesta para encontrar la adecuada proporción en los valores correspondientes a los parámetros de control, tomando en cuenta tanto el problema como el método de solución implementado.

1. *Selección de los Parámetros de Control.*

   Para determinar los parámetros de control del algoritmo, es necesario llevar a cabo una revisión de publicaciones que se relacionen con el algoritmo implementado, de esta manera es posible analizar los parámetros tomados en cuenta en investigaciones anteriores.

   Otra forma de identificar los parámetros de control, es realizar un análisis tanto del problema como del algoritmo, a manera de identificar los parámetros críticos que influyen de cierta manera en la calidad de las soluciones.

2. *Establecer Rangos de Evaluación*

   Una vez que los parámetros de control se han identificado, es necesario establecer los rangos que van a ser utilizados para realizar el análisis de sensibilidad a cada uno de los parámetros, para lo que es necesario llevar a cabo una revisión en la literatura, a manera de identificar y analizar los valores utilizados por diferentes autores para el algoritmo propuesto, de ésta manera es más sencillo especificar un rango de acción para cada uno de los parámetros.

   Cabe mencionar que es de vital importancia llevar a cabo una adecuada sintonización debido a que éste permite identificar la proporción adecuada entre los parámetros de control, dentro de la cuál, el algoritmo obtiene buenas soluciones.

3. *Pruebas a Rangos de Evaluación*

   Para llevar a cabo el análisis de sensibilidad a los parámetros de control mencionados anteriormente, se toman en cuenta los rangos establecidos, calculando series de muestras (la cantidad de muestras varía de acuerdo al tamaño del rango) que permitan evaluar el comportamiento del algoritmo cuando los parámetros de control toman valores distintos.

   Una vez que se tiene el conjunto de muestras, se procede a realizar las pruebas experimentales, para o cuál se recomienda realizar conjuntos de mínimo 30 pruebas para cada una de las muestras calculadas, para llevar a cabo una adecuada sintonización de los parámetros de control, es necesario realizar un barrido de los valores correspondientes a una de las variables, manteniendo fijos los demás, hasta identificar aquel valor que mejore la calidad de las soluciones. Una vez que se ha obtenido el mejor valor para esta variable, se fija dicho valor y se comienza con la variación de otro parámetro, llevando a cabo el mismo proceso, hasta que se obtengan el conjunto de valores que permitan que el algoritmo mejore en eficacia y eficiencia.

   Una vez que se han fijado los valores, se vuelven a calcular cierta cantidad de muestras para un rango más pequeño, tomando como punto medio, el valor fijado, de modo que se vuelve a realizar el proceso de sintonización hasta fijar nuevamente los valores.

4. *Sintonización de Parámetros*

   Los valores obtenidos para cada uno de los parámetros de control al término de la evaluación de todas y cada una de las muestras, son considerados como los *valores de sintonización*.

La sintonización de parámetros se lleva a cabo con la finalidad de identificar aquellos valores correspondientes a las variables de control, que influyen de manera positiva en la calidad de la solución, lo que permite obtener una mejora en el desempeño del algoritmo.

3.8 Análisis de la Complejidad del Algoritmo Colonia de Hormigas

Una vez que se tenga un algoritmo que funcione correctamente, es necesario realizar un estudio que permita conocer su comportamiento y medir su rendimiento, centrándose principalmente en su simplicidad y el uso eficiente de los recursos.

La complejidad de un algoritmo puede ser medida en función de los siguientes parámetros:

- ***Espacio:*** Cantidad de memoria utilizada durante la ejecución del algoritmo.
- ***Tiempo:*** Tiempo requerido para la ejecución del algoritmo.

Ambos parámetros representan el *costo* requerido por el algoritmo para encontrar una solución al problema tratado.

El tiempo de ejecución de un algoritmo o complejidad temporal (*T(n)*), se encuentra en función de los siguientes parámetros: Datos de entrada, calidad del código objeto compilado, naturaleza y rapidez del procesador así como la complejidad del algoritmo.

Ambos análisis representan los requerimientos de memoria y tiempo respectivamente, requeridos por un algoritmo para encontrar una solución a un problema dado. Este análisis de complejidad permite determinar la eficiencia de un algoritmo y por lo tanto, compararlo con otros algoritmos que resuelvan un mismo problema.

De acuerdo a esto, se tiene que *T(n)* representa el número de instrucciones simples (asignaciones, comparaciones, operaciones aritméticas, etc.) que son ejecutadas en el algoritmo.

Salvo que se indique lo contrario, siempre se considera la complejidad del algoritmo en el peor de los casos, aunque también pueden ser analizados el mejor caso y el caso promedio.

Para calcular la complejidad del algoritmo Colonia de Hormigas es necesario realizar un análisis del número de instrucciones requeridas por el algoritmo para encontrar una solución al problema tratado. De acuerdo a esto, se muestra la ecuación correspondiente a la función temporal del algoritmo (Ecuación 5).

$$T(n) = \sum_{t=0}^{iter\max} mn^3 + mn^3 + mn^2 + \sum_{t=0}^{iter\max} (3n^2 + n) + (n^2 t + n) + \sum_{t=0}^{iter\max} (3n + 1) + (3t + 1) \quad (5)$$

Donde *n* representa el número de trabajos de la instancia, *m* corresponde al número de máquinas y *t* representa al número de iteraciones definidas para el algoritmo.

De acuerdo a esto, se puede concluir que la complejidad del algoritmo secuencial de Colonia de Hormigas con una estructura de vecindad híbrida aplicada a búsqueda local para el UPMP, tiene una complejidad en el peor de los casos de $O(mn^3)$.

Capítulo 4

Resultados Experimentales del Algoritmo Colonia de Hormigas

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al Algoritmo Colonia de Hormigas, donde se integra la estructura de vecindad híbrida a búsqueda local. Estos resultados son comparados con los obtenidos por el mismo algoritmo cambiando la estructura de vecindad a una sencilla de un par aleatorio.

Finalmente, se llevó a cabo un análisis para mostrar la eficiencia y eficacia del algoritmo, comparando los resultados obtenidos por Colonia de Hormigas en sus dos versiones (con una estructura de vecindad híbrida y con una estructura de un par adyacente), con los obtenidos por un algoritmo Simplex Clásico utilizando el método de variables artificiales [Hillier y Lieberman, 2008], un Recocido Simulado [Cruz et al., 2009a], y con el algoritmo de Colonia de Hormigas Clásico [Dorigo et al., 1996].

4.1 Descripción del Equipo Utilizado

Para llevar a cabo las pruebas experimentales correspondientes a las diferentes estructuras de vecindad, se utilizó una computadora portátil con las siguientes características:

- ***Procesador:*** Centrino Core 2Duo a 2.0 GHz
- ***Memoria:*** 3.0 GB
- ***Sistema Operativo:*** Windows Vista Home Premium
- ***Compilador:*** Microsoft Visual C++ 2008

En el caso de las pruebas realizadas al Algoritmo Colonia de Hormigas, se utilizó equipo perteneciente al laboratorio de Optimización. A continuación se enlistan las características del equipo utilizado:

- ***Procesador:*** Pentium Dual Core a 2.80 GHz
- ***Memoria:*** 1.0 GB
- ***Sistema Operativo:*** Windows XP Profesional
- ***Compilador:*** Microsoft Visual C++ versión 6.0

4.2 Análisis de Sensibilidad

Para llevar a cabo el análisis de sensibilidad de los parámetros de control del algoritmo Colonia de Hormigas aplicado al problema de Máquinas en Paralelo no Relacionadas, se propuso una metodología, la cuál fue explicada anteriormente (Capítulo 3). A continuación se aplica la metodología al algoritmo Colonia de Hormigas.

1. *Selección de los Parámetros de Control.*

   De acuerdo al análisis realizado tanto al método aplicado como al problema tratado, las variables utilizadas para llevar a cabo el análisis

de sensibilidad del algoritmo Colonia de Hormigas aplicado al problema de Máquinas en Paralelo no Relacionadas son las siguientes:

- Importancia relativa del monto de feromona (α).
- Importancia relativa de la distancia heurística (β).
- Cantidad de feromona por unidad de tiempo (Q).
- Coeficiente de evaporación de la feromona (ρ).

2. Establecer Rangos de Evaluación

Una vez que los parámetros de control se han identificado, es necesario establecer los rangos que van a ser utilizados para realizar el análisis de sensibilidad a cada una de las variables antes mencionadas.

Haciendo un análisis de los valores de cada una de las variables de control, se tiene que sí α es muy grande en comparación con el valor de β, el algoritmo llega a quedar rápidamente atrapado en un óptimo local; este comportamiento es mejor conocido como *stagnation behavior*, de modo que no se encuentran buenas soluciones, produciéndose una convergencia prematura del algoritmo [Alonso et. al. 2004]. Lo mismo sucede con el coeficiente de evaporación de la feromona, ya que para valores de ρ cercanos a 0, la información global de la feromona se evapora muy rápidamente, por lo que las hormigas no obtienen el aprendizaje suficiente para encontrar buenos recorridos. Por el contrario, sí ρ es

muy cercano a la unidad, se llega propiciar la convergencia prematura del algoritmo.

Para el caso de Q (cantidad de feromona por unidad de tiempo), siempre que su valor no sea muy pequeño, este parámetro no es crucial para la convergencia del algoritmo [Dorigo et. al. 1996].

Por lo que se concluye que es de vital importancia encontrar la proporción adecuada entre los parámetros de control, dentro de la cuál el algoritmo obtiene buenas soluciones. De acuerdo a lo mencionado anteriormente y a los valores utilizados en algunas investigaciones reportadas en la literatura, se proponen los siguientes rangos a evaluar para cada una de las variables de control (Tabla 4-1).

Tabla 4-1. *Rangos utilizados para realizar el análisis de sensibilidad al algoritmo Colonia de Hormigas.*

Parámetro de Control	Límite Inferior	Límite Superior
α	0.5	2
β	1	5
ρ	0.2	0.8
Q	{1, 100, 10000}	

3. Pruebas a Rangos de Evaluación

Para realizar el análisis de sensibilidad a los parámetros de control mencionados anteriormente, se llevan a cabo lo siguiente:

- Inicialmente, para cada uno de los rangos mostrados en la tabla anterior se calcularon 10 muestras con los incrementos mostrados en la tabla 4-2, excepto para el caso de Q, donde los valores ya están definidos en la literatura, por lo que se tomaron únicamente tres muestras.

***Tabla 4-2**. Valores de Q e incrementos utilizados para los rangos de los parámetros de control utilizados para el análisis de sensibilidad.*

Parámetro de Control	*Incremento*
α	0.165
β	0.445
ρ	0.066
Q	{1, 100, 10000}

- El análisis de sensibilidad se llevó a cabo realizando un barrido de las 10 muestras de β, de acuerdo al incremento presentado en la Tabla 4-2, manteniendo constante el valor de los parámetros α, ρ y Q por cada serie de pruebas. Cabe mencionar que, por cada muestra evaluada se realizaron un total de 30 ejecuciones.

- Una vez que se ha realizado la evaluación de los resultados obtenidos para cada una de las muestras, se fija el valor de β que haya obtenido los mejores resultados de acuerdo a la función objetivo del problema tratado.

- Se vuelven a realizar las pruebas, esta vez realizando un barrido del valor de α y manteniendo constante los valores de los parámetros β, ρ y Q respectivamente.

- Al igual que en el caso de β, se realiza una serie de 30 ejecuciones por cada incremento evaluado, fijando el valor de α que obtenga las mejores soluciones.

- El mismo proceso se lleva a cabo para los valores de ρ y Q, de modo que prevalezcan sólo aquellos valores que permitan obtener los mejores resultados. De acuerdo a esto, se obtuvieron los siguientes valores (Tabla 4-3).

***Tabla 4-3**. Valores de los parámetros de control, fijados de acuerdo al análisis de sensibilidad.*

Parámetro de Control	*Valor Sintonizado*
α	0.665
β	4.115
ρ	0.8
Q	10000

- Una vez que se han fijado los valores para todos los parámetros de control, se vuelven a generar un total de 10 muestras tomando como punto medio los valores obtenidos y realizando incrementos más pequeños (Tabla 4-4).

Tabla 4-4. *Valores de Q e incrementos utilizados para obtener la sintonización final de los parámetros de control.*

Parámetro de Control	*Incremento*
α	0.03
β	0.07
ρ	0.006
Q	{1, 100, 10000}

- Se vuelven a realizar las pruebas tal y como se explicó anteriormente, hasta que todos los valores de los parámetros de control vuelvan a ser fijados.

4. Sintonización de Parámetros

Los valores obtenidos para cada uno de los parámetros de control evaluados, al término de la evaluación de todas y cada uno de los incrementos (Tabla 4-2), dan como resultado la *sintonización de los parámetros de control.*

La sintonización de parámetros de control se lleva a cabo con la finalidad de identificar aquellos valores correspondientes a las variables de control, que influyen de manera positiva en la calidad de la solución, lo que permite obtener una mejora en el desempeño del algoritmo.

Con base en el análisis de sensibilidad realizado, se obtuvieron los siguientes valores sintonizados (Tabla 4-5).

***Tabla 4-5**. Valores sintonizados correspondientes a los parámetros de control evaluados durante el análisis de sensibilidad.*

Parámetro de Control	Valor Sintonizado
α	0.68
β	4.16
ρ	0.8
Q	10000

4.3 Aplicación de la Estructura de Vecindad al Algoritmo Colonia de Hormigas

Para mejorar la calidad de las soluciones obtenidas por el algoritmo Colonia de Hormigas propuesto para el problema de Máquinas en Paralelo no Relacionadas y de acuerdo a los resultados obtenidos en el análisis realizado a las estructuras de vecindad evaluadas [Cruz et al. 2010b], se decidió llevar a cabo la implementación de una estructura de vecindad híbrida, que permita tener una mejor explotación del espacio de soluciones.

Para aplicar la estructura de vecindad fue necesario realizar una evaluación del algoritmo Colonia de Hormigas que permitió identificar el proceso que requería ser mejorado por dicha estructura, de esta forma, se llegó a la conclusión de que cada que se termina una iteración, es decir, cuando todas las hormigas han terminado un recorrido, se lleva a cabo la comparación entre las soluciones obtenidas por las hormigas para identificar la mejor solución hasta el momento, una vez que se tiene identificado este mejor valor

de la función objetivo hasta el momento, se le aplica la búsqueda local con la estructura de vecindad híbrida, de modo que en cada iteración se vaya mejorando el mejor valor de la función objetivo hasta el momento.

En el caso del Administrador de Acciones, una de sus funciones es la actualización de feromona, para lo cuál, antes de realizar una deposición de la sustancia, aplica nuevamente la estructura de vecindad propuesta a los arcos correspondientes a la mejor solución hasta el momento. Este procedimiento permite mejorar una solución ya mejorada, lo que conlleva a una mayor explotación del espacio de soluciones y por ende, la obtención de resultados de mejor calidad.

4.4 Resultados Experimentales

Para conocer el comportamiento de un algoritmo, es necesario recurrir al cálculo de parámetros estadísticos como son la media, desviación estándar y error relativo. La media permite obtener el promedio de las soluciones encontradas para cada instancia. La desviación estándar permite conocer que tan dispersas están las soluciones con respecto a la media aritmética. El error relativo permite conocer el porcentaje de error de la mejor solución encontrada con respecto a una cota establecida, que para este caso es la solución óptima correspondiente a cada instancia. De acuerdo a los resultados obtenidos por dichos parámetros, se puede determinar si un algoritmo es bueno o no para cierto problema.

Para probar el funcionamiento del algoritmo, se generaron de forma aleatoria cuatro problemas de prueba de 200, 250, 270 y 300 trabajos que requerían ser calendarizados en 12 máquinas en paralelo no relacionadas, los cuáles

fueron nombrados MC_12_200, MC_12_250, MC_12_270 y MC_12_300 respectivamente. Cabe mencionar que los problemas de prueba fueron generados de forma aleatoria debido a que los Benchmarks más grandes referenciados en la literatura para este problema [Arnaout, et al. 2008], son de 120 trabajos por 12 máquinas.

Recordando las características propias del problema, se tiene que el costo de un mismo trabajo requiere diferente cantidad de unidades de tiempo para ser procesado, dependiendo de la máquina y posición a la que sea asignado. Cabe resaltar que estas características del problema ya se encuentran incluidas al momento de generar los problemas de prueba.

Para realizar las pruebas experimentales, se ejecutaron 30 veces cada uno de los problemas de prueba antes mencionados, tomando como criterio de paro el número de iteraciones, las cuales varían dependiendo del tamaño de la entrada. Primeramente se llevaron a cabo pruebas por alrededor de 1 hora por prueba, de acuerdo al número de iteraciones programadas, pero la calidad de las soluciones no fue muy buena, por lo que se incrementó el tiempo de ejecución para cada prueba. El número de iteraciones programadas para cada instancia, el tiempo de ejecución promedio fue de 3 horas por instancia, con lo cuál mejoró considerablemente la calidad de las soluciones. Como punto de comparación, se tomó el valor del óptimo global reportado en [Cruz y Juárez, 2010] con el algoritmo Simplex Clásico con el método de variables artificiales [Cruz, Juarez, 2010]. Los resultados son presentados a continuación en la tabla 4-6.

Este algoritmo Simplex Clásico, en el peor de los casos obtiene el óptimo en aproximadamente 3 días por cada instancia, dependiendo del tamaño del tamaño de la misma, además de que requiere más de 18 GB de memoria

RAM para su ejecución, siendo su complejidad de tipo exponencial tanto en tiempo como en memoria.

***Tabla 4-6**. Resultados del algoritmo Colonia de Hormigas aplicando una estructura de vecindad Híbrida.*

Instancia	Mejor	Peor	Promedio	Óptimo	σ	Error Rel.
MC_12_300	4974	5982	5387.87	4870	280.48	2.14%
MC_12_270	4136	5092	4581.90	4056	449.04	1.97%
MC_12_250	3346	3777	3515.07	3283	134.43	1.92%
MC_12_200	2167	2754	2320.00	2127	139.32	1.88%

De acuerdo a esto, se tiene que *mejor* representa la mejor solución encontrada de acuerdo al total de ejecuciones realizadas para cada instancia, *peor* es la solución más lejana al óptimo encontrada, *óptimo* representa el resultado obtenido por el algoritmo Simplex Clásico para cada una de las instancias [Cruz, Juárez, 2010], σ simboliza el cálculo de la desviación estándar y Error Rel., representa el error relativo de la mejor solución encontrada con respecto al óptimo. Para el cálculo del error relativo se utilizó la ecuación 9.

$$ER = \frac{mejor\_encontrado - Óptimo}{Óptimo} \qquad (9)$$

Como se puede observar en los resultados obtenidos en la Tabla 4-4, al calcular el error relativo del mejor resultado obtenido para cada una de las instancias por el algoritmo de Colonia de Hormigas con la estructura de vecindad híbrida aplicada a búsqueda local, se obtiene un porcentaje de alrededor del 2% dependiendo de la instancia evaluada. En base a este porcentaje, el cuál aún es considerado un poco alto dentro del área de

optimización, se probaron otras opciones, tomando en cuenta los resultados obtenidos en las pruebas realizadas a las estructuras y tomando en cuenta que aunque ambos problemas son clasificados como NP, el problema Clásico del Agente Viajero es más duro [Papadimitriou y Steiglitz, 1998].

Se decidió realizar nuevamente las pruebas utilizando las mismas instancias, solo que esta vez cambiando la estructura de vecindad utilizada por una estructura sencilla (cada una de las utilizadas en la estructura híbrida), de modo que para el problema de Máquinas en Paralelo no Relacionadas, los mejores resultados fueron obtenidos por una estructura de un par aleatorio.

Dicha estructura obtuvo mejores resultados que la estructura híbrida propuesta para este trabajo de investigación, por lo que se realizaron pruebas con todas las instancias evaluadas anteriormente y bajo las mismas condiciones.

De acuerdo esto, se llevó a cabo una compilación de los resultados obtenidos en las pruebas experimentales realizadas a esta segunda versión del algoritmo de Colonia de Hormigas (Tabla 4-5).

***Tabla 4-7**. Resultados del algoritmo Colonia de Hormigas aplicando una estructura de vecindad de un par aleatorio.*

Instancia	Mejor	Peor	Promedio	Óptimo	Desv. Est	Error Rel.
MC_12_300	4924	5376	5139.00	4870	118.34	1.11%
MC_12_270	4096	4659	4279.00	4056	159.86	0.99%
MC_12_250	3315	3733	3460.17	3283	105.03	0.97%
MC_12_200	2147	2428	2316.60	2127	91.41	0.94%

Haciendo una comparación entre los resultados obtenidos por ambas versiones del algoritmo, es posible observar que al aplicar una estructura sencilla que sólo realiza una permutación de un par aleatorio, se obtienen mejores resultados que con la estructura híbrida.

Con la finalidad de realizar una comparación de los resultados obtenidos por el algoritmo mejorado por una estructura de vecindad, se ejecutaron los mismos problemas de prueba en un algoritmo de Recocido Simulado Secuencial, mismo que fue desarrollado dentro del grupo de trabajo y probado experimentalmente tanto en eficiencia como en eficacia [Cruz et. al., 2009a]. Las pruebas se realizaron bajo las mismas condiciones (equipo, tiempo de ejecución y cantidad de ejecuciones por instancia) para poder llevar a cabo un análisis confiable.

Los resultados obtenidos por el algoritmo de Recocido Simulado (Tabla 4-8) se obtuvieron bajo las siguientes condiciones: temperatura inicial de 40, factor de reducción de la temperatura de 0.986, punto de congelamiento de 0.0001 y tomando el número de ciclos como la cantidad de vecinos *1000.

***Tabla 4-8**. Resultados del algoritmo de Recocido Simulado.*

Instancia	Mejor	Peor	Promedio	Óptimo	Desv. Est	Error Rel.
MC_12_300	4874	4874	4874.00	4870	0.00	0.08%
MC_12_270	4067	4067	4067.00	4056	0.00	0.27%
MC_12_250	3290	3290	3290.00	3283	0.00	0.21%
MC_12_200	2133	2134	2133.20	2127	0.41	0.28%

4.5 Análisis de Eficacia y Eficiencia del Algoritmo

El análisis de eficacia y eficiencia se lleva a cabo para evaluar tanto la calidad de las soluciones obtenidas por un algoritmo, como los recursos y tiempo necesarios para su ejecución. Para llevar a cabo este análisis, se realizaron comparaciones del algoritmo propuesto con los algoritmos antes mencionados (Recocido Simulado y Simplex Clásico utilizando el método de variables artificiales).

En la figura 4-1, se muestra más claramente la calidad de las soluciones encontradas por cada uno de los algoritmos evaluados (tablas 4-4 a la 4-6) con respecto a la solución óptima de cada instancia, la cuál fue obtenida con un algoritmo Simplex Clásico [Cruz y Juárez. 2010].

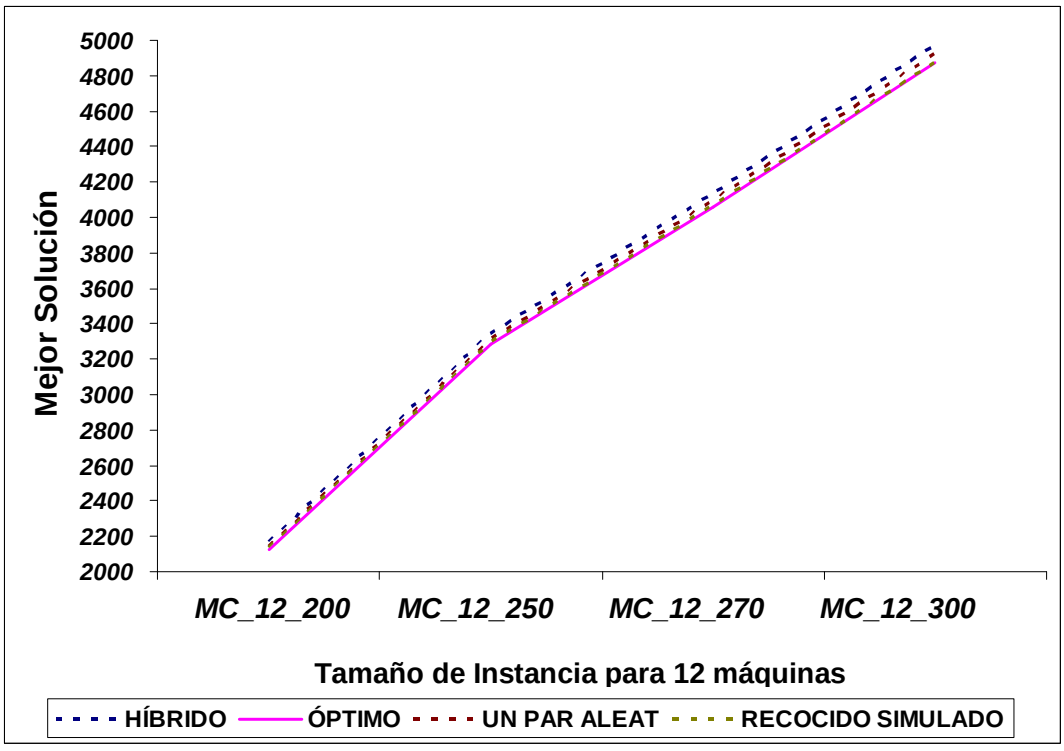


***Figura 4-1**. Gráfica comparativa del funcionamiento de los algoritmos evaluados con respecto a la solución óptima.*

Para tener un panorama más claro en cuanto a la calidad de las soluciones, se muestra en la figura 4-2 una gráfica comparativa del error relativo obtenido por cada uno de los tres algoritmos evaluados (Colonia de Hormigas con una estructura híbrida, Colonia de Hormigas con una estructura de par aleatorio y Recocido Simulado) para cada tamaño de instancia evaluada.

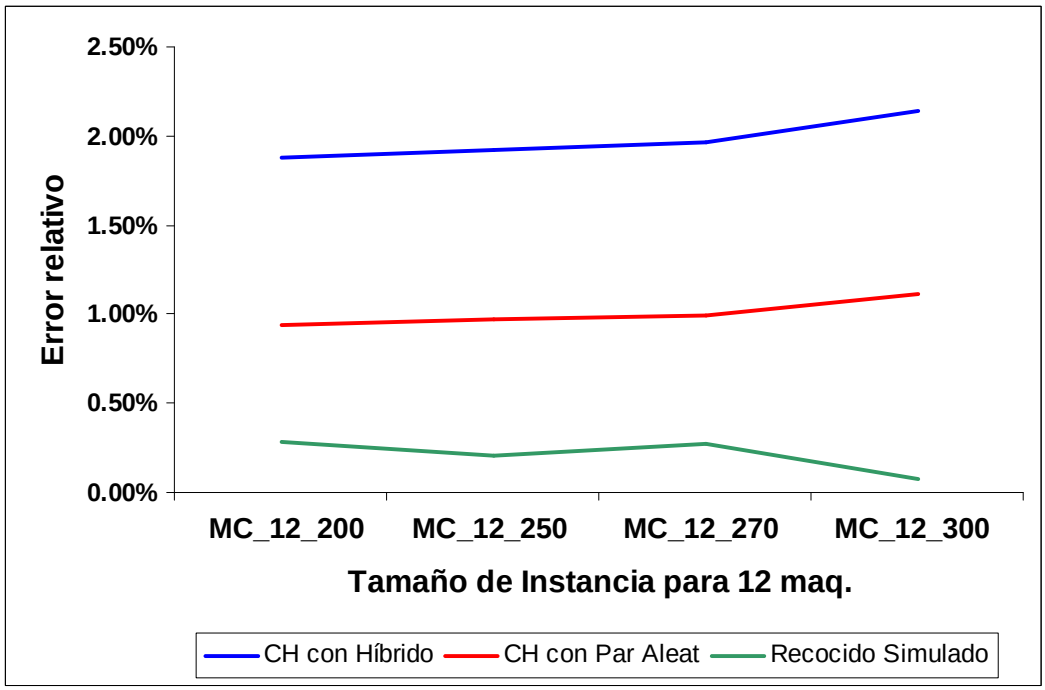


Figura 4-2. *Gráfica comparativa del error relativo obtenido por cada uno de los tres algoritmos evaluados.*

De acuerdo al comportamiento mostrado en las figuras 4-1 y 4-2, se puede observar claramente que por parte del algoritmo Colonia de Hormigas, la versión que obtiene mejores resultados es el algoritmo al que fue aplicada una estructura de vecindad de un par aleatorio a búsqueda local.

En cuanto al algoritmo más eficaz, es decir, el que obtiene mejores soluciones para este problema, tomando en cuenta todas las evaluaciones realizadas, es el algoritmo de Recocido Simulado; de acuerdo a esto, se puede decir que el algoritmo propuesto, aplicando una estructura de vecindad de un par aleatorio es competitivo, tomando en cuenta la eficacia y eficiencia ya conocida en la literatura por parte del algoritmo de Recocido Simulado, al ser aplicado a diversos problemas de optimización [Dowsland y Adenso, 2003].

Además de la calidad de las soluciones, otro factor a evaluar es el tiempo requerido por cada uno de los algoritmos para obtener una solución al problema tratado. En el caso del algoritmo Simplex Clásico, el tiempo requerido para encontrar la solución óptima a cada una de las instancias evaluadas se muestra en la tabla 4-9.

Tabla 4-9. *Tiempo requerido por Simplex Clásico para obtener la solución óptima a cada una de las instancias evaluadas [Cruz y Juárez, 2010].*

Instancia	Tiempo de Ejecución
MC_12_300	57.18 hrs.
MC_12_270	37.10 hrs.
MC_12_250	24.36 hrs.
MC_12_200	9.45 hrs.

Cabe mencionar que las pruebas realizadas al algoritmo Simplex Clásico, para obtener la solución óptima a cada una de las instancias evaluadas, se llevó a cabo en una máquina con características diferentes, debido a los requerimientos de memoria del algoritmo Simplex Clásico, dada su ya conocida complejidad. La

máquina utilizada fue una Workstation cuyas características se enlistan a continuación.

- **Sistema Operativo:** Windows Vista Ultimate de 64 bits. SP. 2
- **Procesador:** Intel Xeón Cuad-Core 3.16 Ghz
- **Memoria:** 20 GB
- **Compilador:** Microsoft Visual C++ 2008.

Parte importante a señalar es que para el caso de ambas versiones del algoritmo Colonia de Hormigas (con una estructura híbrida y con una estructura de un par aleatorio), así como para el caso del algoritmo de Recocido Simulado, se dieron cierta cantidad de iteraciones (para el caso de Colonia de Hormigas) y número de ciclos (Recocido Simulado), de modo que el tiempo de ejecución fuera el mismo en cualquier algoritmo para cada prueba realizada.

Por lo que se puede observar, que a pesar de que en el caso de Simplex Clásico se obtiene la solución óptima, el tiempo de ejecución es extremadamente largo en comparación con los otros dos algoritmos heurísticos evaluados, los cuales, aunque cuentan con cierto rango de error, el tiempo para obtener una buena solución cercana al óptimo, se reduce considerablemente.

4.6 Análisis de los Resultados

Para comprobar la mejora del algoritmo de Colonia de Hormigas de acuerdo a la aportación de la tesis, por medio de la implementación

de una estructura de vecindad aplicada a búsqueda local, se realizaron pruebas experimentales al algoritmo Colonia de Hormigas Clásico, tal y como se muestra en la literatura, es decir, quitando las funciones de la estructura de vecindad. Cabe mencionar que estas pruebas fueron realizadas bajo las mismas condiciones manejadas para los otros algoritmos y utilizando los mismos problemas de prueba. A continuación, en la tabla 4-10 se muestran los resultados obtenidos.

Tabla 4-10. *Resultados obtenidos de la ejecución del algoritmo Colonia de Hormigas Clásico.*

Instancia	Mejor	Peor	Promedio	Óptimo	Desv. Est	Error Rel.
MC_12_300	2460	2548	2514.63	4870	21.11	15.66%
MC_12_270	3833	3956	3878.10	4056	31.79	16.75%
MC_12_250	4790	4878	4847.37	3283	26.39	18.10%
MC_12_200	5786	5896	5843.83	2127	26.60	18.81%

Como se puede observar, los resultados obtenidos por este algoritmo tienen un error relativo muy alto con respecto a los obtenidos con la implementación de una estructura de vecindad, ya sea híbrida o de un par aleatorio. Para observar esta diferencia de forma más clara, se muestra en la figura 4-3 la comparación de las tres versiones del algoritmo Colonia de Hormigas (Colonia de Hormigas con una estructura de vecindad híbrida, Colonia de Hormigas con una estructura de un par aleatorio y Colonia de Hormigas Clásico) en cuanto al porcentaje de error relativo de las soluciones obtenidas.

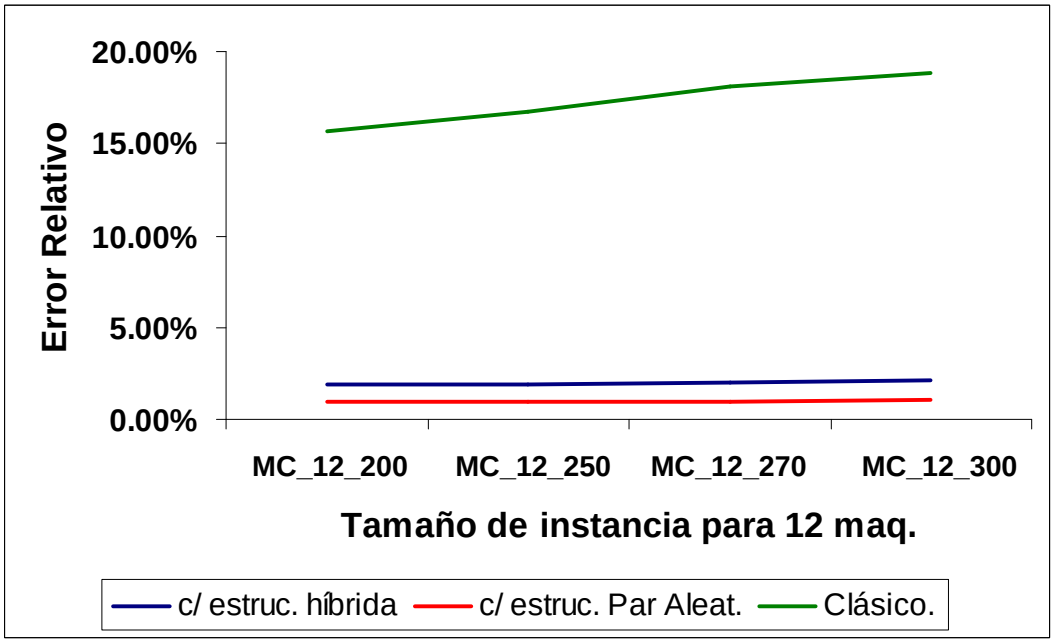


Figura 4-3. *Gráfica comparativa del error relativo del algoritmo Colonia de Hormigas con una estructura híbrida, con una estructura un par aleatorio y Colonia de Hormigas Clásico.*

De acuerdo a la figura 4-3, es posible observar claramente en base al error relativo, que las mejores soluciones al UPMP utilizando el algoritmo Colonia de Hormigas, se obtuvieron al implementar una estructura de vecindad sencilla de un par aleatorio, quedando en segundo lugar el algoritmo Colonia de Hormigas con una estructura híbrida. Por lo que se comprueba que la aplicación de una estructura de vecindad hace eficaz al algoritmo Colonia de Hormigas.

Con la finalidad de dar una explicación al comportamiento de la estructura híbrida aplicada al algoritmo Colonia de Hormigas para UPMP, la cuál, contrario a lo que se esperaba de acuerdo a las pruebas realizadas en Cruz et al., 2010b, no obtuvo mejores resultados para el UPMP, en comparación

con la aplicación de una estructura sencilla, por lo que se plantea la siguiente hipótesis:

"*Los buenos resultados obtenidos por una estructura de vecindad híbrida aplicada al problema Clásico Simétrico del Agente Viajero, se debe a la complejidad de su espacio de soluciones, ya que aunque tanto el problema del agente viajero como el de máquinas en paralelo no relacionadas son clasificados como NP, la dureza de éste último es menor, por lo que su espacio de soluciones no es tan complejo como el del Agente Viajero Clásico.*

De acuerdo a esto, se podría decir que una estructura de vecindad híbrida, sólo funciona adecuadamente al ser aplicada a problemas que cuenten con un espacio de soluciones de complejidad alta; esto, debido a la naturaleza de las estructuras que la componen, ya que los saltos que realiza dentro del espacio de soluciones varían considerablemente, pudiendo ser muy pequeños o bien, muy grandes, lo que beneficia la explotación del espacio de soluciones complejos, contrario a lo que sucede con espacios de soluciones más sencillos".

Capítulo 5

Conclusiones y Trabajos Futuros

5.1 Conclusiones

De acuerdo a las pruebas experimentales realizadas al algoritmo Colonia de Hormigas, utilizando tanto la estructura híbrida propuesta como la de un par aleatorio, se puede concluir que el algoritmo propuesto es eficiente, debido a que su rango de error relativo se encuentra alrededor del 1% al utilizar una estructura de vecindad sencilla de un par aleatorio, aplicada a búsqueda local.

Al comparar el error relativo obtenido por el algoritmo propuesto con ambas estructuras de vecindad, con el obtenido por un algoritmo de Recocido Simulado, se puede decir que es un algoritmo competitivo, debido a la ya conocida eficiencia y eficacia del Recocido Simulado, al ser aplicado a diversos problemas de optimización considerados difíciles de resolver, además, de que al comparar dichos resultados con los del algoritmo Clásico de Colonia de Hormigas, se puede observar claramente que la implementación de una estructura de vecindad hace al algoritmo eficaz, ya que sin ésta, los resultados son muy pobres, con lo que se prueba que Colonia de Hormigas Clásico es un algoritmo ineficiente para UPMP.

Se concluye que la estructura de vecindad híbrida propuesta en este trabajo de investigación, no es muy adecuada para el problema tratado, comparando los resultados obtenidos con los de una estructura sencilla de un par aleatorio.

De acuerdo a la hipótesis planteada en éste trabajo de investigación, se considera que el comportamiento de la estructura híbrida se debe a la complejidad del espacio de soluciones del problema tratado, ya que aunque es considerado como un problema NP, éste es menos complejo que el del problema del Agente Viajero donde fue probada inicialmente la estructura de vecindad híbrida aplicada a búsqueda local, dando excelentes resultados. De manera que, de acuerdo a la hipótesis planteada, espacios de soluciones más complejos se verían beneficiados con la aplicación de una estructura de este tipo, debido a que al incorporar características de diversas estructuras, los saltos que realiza dentro del espacio de soluciones varían en magnitud, debido a que la selección de las diferentes estructuras incorporadas a la estructura híbrida se lleva a cabo de forma aleatoria.

5.2 Trabajos Futuros

La propuesta de solución desarrollada durante este trabajo de investigación, servirá como base para las siguientes actividades a realizar como trabajo futuro:

1) Realizar la investigación requerida y pruebas necesarias para comprobar la hipótesis señalada al final del capítulo 4, en cuanto al funcionamiento de la estructura de vecindad híbrida para problemas de optimización con espacios de soluciones complejos, como es el

caso del problema del Agente Viajero, el cuál es conocido como uno de los problemas más difíciles de resolver y en el que fue probada dicha estructura en un trabajo anterior obteniendo excelentes resultados [Cruz et al., 2010b].

a. Probar la estructura de vecindad híbrida propuesta con problemas más sencillos como es el caso del problema de Emparejamiento (*Matching*) o el del Árbol de Expansión Mínima, a manera de observar el comportamiento de la estructura en problemas de menor dureza.

b. Probar la estructura de vecindad híbrida en problemas más duros como es el caso del JSSP, para poder realizar una comparación de los resultados y con ello comprobar la hipótesis planteada en este trabajo de investigación.

2) Realizar la paralelización del algoritmo de Colonia de Hormigas para mejorar su eficiencia en cuanto al tiempo de ejecución del algoritmo, probando instancias más grandes, para las cuales Simplex Clásico tardaría demasiado tiempo en encontrar una solución óptima.

Resumen de las Publicaciones

Se enlistan artículos realizados a partir de éste trabajo de investigación, señalando aquellos que ya se han publicado, así como de los que están por ser publicados.

Cruz-Chávez Marco Antonio, Juárez-Pérez Fredy, Ávila-Melgar Erika Yesenia, **Martínez-Oropeza Alina**. Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem. Presentado como ponencia en Cerma 2009 y publicado como memoria escrita. ISBN-13:978-0-7695-3799-3. pp. 94. 2009

Cruz-Chávez Marco Antonio, **Martínez-Oropeza Alina**, Zavala-Díaz José Crispín, Martínez-Rangel Martín G. Relajación del Problema de Calendarización de Trabajos en un Taller de Manufactura utilizando un Grafo Bipartita. Presentado en el 7mo. Congreso Internacional de Cómputo en Optimización y Software AGECOMP-CICos 2009 como ponencia y publicado como memoria electrónica y escrita. ISBN: 978-970.9750-26.3. 2009

Cruz-Chávez Marco Antonio, **Martínez Oropeza Alina**, Rivera López Rafael. Relaxation of Job Shop Scheduling Problem using a Bipartite Graph. Aceptado como ponencia en Cerma 2010. A publicar en IEEE. 2010.

Cruz-Chávez Marco Antonio, **Martínez-Oropeza Alina**, Serna-Barquera Sergio A. Neighboorhood Hybrid Structure for Discrete Optimization Problems. Aceptado como ponencia en Cerma 2010. A publicar en IEEE. 2010.

Referencias

[Agarwal y Grossman, 2004]. Agarwal A., Grossman I. E. *Modeling for Optimization of Hybrid Dynamic Networks*. Submitted to Computers Chemical Engineering, 2004.

[Aho, et al., 1974]. Aho Alfred V., Hopcroft E. John, Ullman Jeffrey D. *The Design and Analysis of Computer Algorithms.* Ed. Addison-Wesley Publishing Company. ISBN. 0-201-00029-6. pp. 2, 60. 1974.

[Alba, 2005]. AlbaEnrique. *Parallel Metaheuristics*. *A New Class of Algorithms*. ISBN-13 978-0-471-67806-9, ISBN-10 0-471-67806-6. Canada. 2005.

[Alonso et al. 2004]. Alonso Sergio, Cordón Oscar, Fernández-de Viana Iñaki, Herrera Francisco. *La Metaheurística de Optimización Basada en Colonia de Hormigas: Modelos y Nuevos Enfoques*. Optimización Inteligente: Técnicas de Inteligencia Computacional para Optimización. ISBN. 84-9747-034-6. pp. 261-314- 2004.

[Anagnostopoulos y Rabadi, 2002]. Anagnostopoulos Georgios C., Rabadi Ghaith. A Simulated Annealing Algorithm for the Unrelated Parallel Machine Scheduling Problem. School of

Electrical Engineering & Computer Science, Orlando Florida, 2002.

[Anastova y Dror, 1998]. Anastasova K., Dror M. Intelligent Scheduler for Processing Help Requests on Unrelated Parallel Machines in a Computer Support Administration System, Conference on Systems, Man, and Cybernetics, IEEE, pp. 372-377, ISBN: 0-7803-4778-1, 1998.

[Anil y Marathe, 2005]. Anil Kumar V.S., Marathe M. V., Approximation Algorithms for Scheduling on Multiple Machines, Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), pp. 254-263, ISBN: 0-7695-2468-0/05, 2005.

[Arnaout et al., 2008]. Arnaout Jean-Paul, Musa Rami, Rabadi Ghaith. Ant Colony Optimization Algorithm to Parallel Machine Scheduling Problem with Setups. 4th IEEE Conference on Automation Science and Engineering. ISBN: 978-1-4244-2023-0/08. 2008.

[Bisbal, 2009]. Bisbal Riera Jesús. "Manual de Algorítmica". Recursividad, Complejidad y Diseño de Algoritmos. Ed. UOC. Primera Edición en Castellano. ISBN: 978-84-9788-027-5. pp. 54-59. Barcelona. 2009.

[Bullnheimer et al., 1997]. Bullnheimer Bernd, Hartl Richard F., Strauss Christine. Applying the Ant System to the Vehicle Routing Problem. 2nd International Conference on Metaheuristics. MIC 97. Francia. 1997.

[Chun y Chuen, 2006]. Chun-Lung Chen, Chuen-Lung Chen. A Heuristic Method for a Flexible Flow Line with Unrelated Parallel Machines Problem. IEEE Conference on Robotics, Automation and Mechatronics. Digital Object Identifier: 10.1109/RAMECH.2006.252673. pp. 1 – 4. 2006.

[Chun, 2008]. Chun-Lung Chen, An Iterated Local Search for Unrelated Parallel Machines Problem with Unequal Ready Times, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 2044-2047, Qingdao, China September, ISBN: 978-1-4244-2502-0. 2008.

[Cruz et al., 2009a]. Cruz-Chávez Marco Antonio, Juárez-Pérez Fredy, Ávila-Melgar Erika Yesenia, Martínez-Oropeza Alina. Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem. Cerma 2009. ISBN-13:978-0-7695-3799-3. pp. 94. 2009

[Cruz et al., 2009b]. Cruz-Chávez Marco Antonio, Martínez-Oropeza Alina, Zavala-Díaz José Crispín, Martínez-Rangel Martín G. Relajación del Problema de Calendarización de Trabajos en un Taller de Manufactura utilizando un Grafo Bipartita. 7mo. Congreso Internacional de Cómputo en Optimización y Software AGECOMP-CICos 2009. ISBN: 978-970.9750-26.3. 2009

[Cruz et al., 2010a]. Cruz-Chávez Marco Antonio, Martínez Oropeza Alina, Rivera López Rafael. Relaxation of Job Shop

Scheduling Problem using a Bipartite Graph. Accepted as paper in Cerma 2010. To publish in IEEE. 2010.

[Cruz et al., 2010b]. Cruz-Chávez Marco Antonio, Martínez-Oropeza Alina, Serna-Barquera Sergio A. Neighborhood Hybrid Structure for Discrete Optimization Problems. Accepted as paper in Cerma 2010. To publish in IEEE. 2010.

[Cruz y Juárez, 2010]. Cruz-Chávez Marco Antonio, Juárez Pérez Fredy. Algoritmo de Recocido Simulado con Paso de Mensajes en Ambiente Grid para el Problema de Máquinas en Paralelo no Relacionadas. A enviar a Journal of Grid Computing. ISSN (e): 1570-7873. Springer. 2010.

[Dorigo et al., 1991a]. Dorigo M., Maniezzo V., Colorni A. Positive Feedback as a Search Strategy. Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1991a.

[Dorigo et al., 1991b]. Dorigo M., Maniezzo V., Colorni A. The Ant System: An Autocatalytic Optimizing Process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1991b.

[Dorigo, 1992]. Dorigo M. Optimization, Learning and Natural Algorithms [in Italian]. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1992.

[Dorigo et al., 1996]. Dorigo M., Maniezzo V., Colorni A. Ant System: Optimization by a Colony of Cooperating Agents. IEEE

Transactions on Systems, Man. and Cybernetics–Part B, 26(1):29–41, 1996.

[Dorigo y Stützle, 2003]. Dorigo M., Stützle T. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Adavances. Handbook of Metaheuristics, Springer. 2003.

[Dowsland y Adenso, 2003]. Dowsland Kathryn A., Adenso-Díaz Belarmino. Diseño de Heurística y Fundamentos del Recocido Simulado. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. Asociación Española para la Inteligencia Artificial. Año/Vol. 7, No. 019. ISSN(i): 1137-3601, ISSN(e): 1988-3064. Valencia, España. 2003.

[Feo y Resende, 1989]. Feo T. A., Resende M. G. Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. Operation Research Letters. Vol. 8. Issue 2. pp. 67 – 71. 1989.

[Garey et al. 1976]. Garey M. R., Johnson, D.S., Shethi, R. The Complexity of Flow Shop and Job Shop Scheduling, in Mathematics of Operation Research, vol. 1; No.2. pp. 117-129. 1976.

[Garey y Johnson, 1979]. Garey M. R., D. Johnson. “Computers and Intractability” a Guide to the Theory of NP-Completeness. Freeman. New York NY. ISBN 0-7167-1044-7. 1979.

[Glover, 1989]. Glover F. Tabu Search. Part I. ORSA Journal of Computing. Vol. 1. No. 3. pp. 190 – 206. 1989.

[Gómez et al., 2007]. Gómez Ravetti, Geraldo R. Mateus, Pedro L. Rocha. A Scheduling Problem with Unrelated Parallel Machines and Sequence Dependent Setups. 380 Int. J. Operational Research, Vol. 2, No. 4. Copyright © 2007 Inderscience Enterprises Ltd. 2007.

[Graham et al., 1979]. Graham R.L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Ann. Discrete Math., pp. 287–326, 1979.

[Grassé, 1959]. Grassé P. P. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. La théorie de la Stigmergie: essai d inter-pretation du Comportement des Termites Constructeurs, pages 41-81, 1959.

[Gu, Huang, 1994]. Gu Jun, Huang Xiaofei. Efficient Local Search with Search Space Smoothing : A Case Study of the Traveling Salesman Problem (TSP). IEEE Transactions on Systems, Man, and Cybernetics. Vol.24. No. 5. 1994.

[Guo et al., 2007]. Guo Y., Lim. A., Rodrigues B., Yang L. Minimizing the Makespan for Unrelated Parallel Machines. International Journal on Artificial Intelligence Tools (IJAIT). Vol. 16. Issue 3. pp. 399 – 415 Junio, 2007.

[Hillier, Lieberman, 2008]. Hillier S., Lieberman G.J., Introduction to Operation Research, 8th ed.,Mc Graw Hill , ISBN: 0073017795, 2008.

[Hong et al., 2007]. Hong Zhou, Zhengdao Li, Xuejing Wu. Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization. Proceeding of the IEEE International Conference on Automation and Logistic. Jinan, China. 2007

[Klee y Minty, 1972]. Klee V., Minty G.J.. "How Good is the Simplex Algorithm?" In O. Shisha, editor, Inequalities, III, pages 159–175. Academic Press, New York, NY, 1972.

[Koranne, 2002] Koranne S. Formulating SoC Test Scheduling as a Network Transportation Problem, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, december 2002.

[Rosen, 2007]. Rosen Kenneth H. Discrete Mathematics and its Applications. Fifth Edition. Ed. Mc. Graw Hill. International Edition 2003. pp. 549. 2003.

[Mönch, 2008]. Mönch L. Heuristics to Minimize Total Weighted Tardiness of Jobs on Unrelated Parallel Machines, 4th IEEE Conference on Automation Science and Engineering Key Bridge Marriott, Washington DC, USA, pp. 572-577, August 23-26, ISBN: 978-1-4244-2022-3, 2008.

[Martínez, 2006]. Martínez Morales A. A. Algoritmo Basado en Tabu Search para el Cálculo del Índice de Transmisión de un Grafo. Departamento de Computación Facultad de Ciencias y Tecnología. FARAUTE de Ciencias y Tecnología, Vol. 1, No. 1,

pp. 31-39. Universidad de Carabobo, Valencia, Estado Carabobo, Venezuela. 2006.

[Martínez, Martín, 2003]. Martínez Gil Francisco A., Martín Quetglás Gregorio. Introducción a la Programación Estructura en C. ISBN. 84-370-5666-7. Universidad de Valencia. pp. 209-212. Ed. Maite Simon.2003.

[Mateo, 2009]. Mateo Manuel, Ribas Imma, Ramón Companys. A GRASP Procedure for Scheduling Orders on Parallel Machines with Setup Times. 3rd. International Conference on Industrial Engineering and Industrial Management. XIII Congreso de Ingeniería de Organización. Barcelona-Terrassa, 2009.

[Michalewicz y Fogel, 2002]. Michalewicz Zbigniew, Fogel David B. *How to Solve It: Modern Heuristics*. Springer-Verlag Berlin Heidelberg New York. ISBN. 3-540-66061-5. pp. 55 – 90. 2002

[Muñoz y Moraga, 2006]. Muñoz Valdez, R. J. Moraga Cuarzo. Heurística Constructiva Visionaria para el Problema de Máquinas Paralelas no Relacionadas con tiempos de Setup Dependientes de la Secuencia (Look-Ahead Constructive Heuristic for the Unrelated Parallel Machine Problem with Sequence Dependent Setup Times). Revista Ingeniería Industrial – Año 5, Num. 1, Segundo Semestre 2006. ISSN 0717-9103. 2006.

[Osman y Kelly, 1996]. Osman, I.H., Kelly, J.P. Meta-Heuristics: Theory and Applications, 39 Kluwer Academic Publishers. ISBN: 0792397002. USA, 1996.

[Papadimitriou y Steiglitz, 1998]. Papadimitriou C. H., Steiglitz Kenneth. Combinatorial Optmization. Algorithms and Complexity. ISBN. 0-486-40258-4. USA. 1998

[Pei y Shih, 2007]. Pei-Chann Chang, Shih-Hsin Chen. Integrating Dominance Properties with Genetic Algorithms for Parallel Machine Scheduling Problems with Setup Times. Applied Soft Computing. ScienceDirect, 2007.

[Pessan et al. 2007]. Pessan C., Bouquard J. L., Néron E., An Unrelated Parallel Machines Model for Production Resetting Optimization, International Conference on Service Systems and Service Management IEEE, Vol 2. ISBN: 1-4244-0450-9. pp. 1178-1182. 2007.

[Pinedo, 2001]. Pinedo M. Scheduling Theory, Algorithms, and Systems. ISBN: 0130281387, 2 ed. Prentice Hall. USA., Aug. 2001.

[Pinedo, 2008]. Pinedo Michael L. Scheduling Theory, Algorithms, and Systems. Third Edition. New York University. ISBN: 978-0-387-78934-7, e-ISBN: 978-0-387-78935-4. Ed. Prentice Hall. Springer. pp. 14. 2008.

[Riojas, 2005]. Riojas Cañari, Alicia Cirila. Conceptos, Algoritmo y Aplicación al Problema de las N-Reinas. Capítulo 2. Heurística y Metaheurística. Monografía para optar el Título de Licenciada de Investigación Operativa. Lima – Perú, 2005.

[Stattennberger et al., 2007]. Stattenberger Günther, Dankesreiter Markus, Baumgartner Florian, Scheider Johannes J. On the Neighborhood Structure of the Traveling Salesman Problem Generated by Local Search Moves. J Stat Phys129:623-648. DOI 10.1007/s10955-007-9382-1. Springer Science + Business Media, LLC 2007.

[Téllez, 2007]. Téllez Enríquez Emanuel. Uso de una Colonia de Hormigas para Resolver Problemas de Programación de Horarios. Tesis para obtener el grado de Maestro en Ciencias de la Computación. Laboratorio Nacional de Informática Avanzada (LANIA). Veracruz, 2007.

[Vallada y Ruiz, 2009]. Vallada Eva, Ruiz Rubén. Metaheurísticas para el problema de Secuenciación en Máquinas Paralelas no Relacionadas con Tiempos de Cambio. 3rd. Internacional Conference on Industrial Engineering and Industrial Management. XIII Congreso de Ingeniería de Organización. Barcelona-Terrassa, 2009.

[Wetzel, 1983]. Wetzel. A. Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization. University of Pittsburgh, Pittsburgh (unpublished). 1983.

[Yamada y Nakano, 1997]. Yamada Takeshi, Nakano Ryohei. Genetic Algorithms for Job-Shop Scheduling Problems. Proceedings of Modern Heuristic for Decisión Support, pp. 67-81, UNICOM Seminar, London 1997.

[Zalzala y Fleming, 1997]. Zalzala A. M. S., Flemming P. J: Genetic Algorithms in Engineering Systems. Published by: Institute of Electrical Engineers. London, United Kingdom. ISBN. 0 85296 902 3.1997.

[Zhou et al., 2007]. Zhou H., Li Z., Wu X. Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization. Proceedings of the IEEE International Conference on Automation and Logistics. pp. 132-136, August 18 - 21, Jinan, China, 2007.

APÉNDICE A

Estructuras de Datos Utilizadas en el Algoritmo Colonia de Hormigas Aplicado a UPMP

Para dar una solución al UPMP por medio de un algoritmo Colonia de Hormigas, fue necesaria la utilización de estructuras de datos que permitieran el uso ordenado y eficiente de la información, ésto, debido a que una estructura es una colección de uno o más tipos de elementos, donde cada uno puede ser de diferente tipo de dato.

A continuación se explican cada una de las estructuras utilizadas en el algoritmo propuesto.

```
struct lista_tabu  {
                        int trab;
                        int m;
                        int k;
                   };
```

La estructura `lista_tabu`, permite almacenar el recorrido de las hormigas, ésta estructura no es utilizada directamente, sino que sus características son utilizadas por las demás estructuras de datos utilizadas por el algoritmo.

```
struct Hormigas {
                      int k_Disp[MACHINES][Kn];
                      int Costo_total;
                      struct lista_tabu result[JOBS];
                }Ant[ANTS];
```

La estructura `Hormigas`, es la estructura principal, ya que ésta es la estructura utilizada por cada una de las hormigas para llevar a cabo el proceso de construcción de soluciones, además de que ayuda a que las soluciones construidas cumplan con las restricciones del problema, debido a que incorpora una matriz que muestra las posiciones disponibles de cada una de las máquinas, además de que almacena el recorrido de cada hormiga con su correspondiente Costo.

```
struct minimiza {
                      int mejor_costo;
                      struct lista_tabu mejor_rec[JOBS];
                }mejor;
```

La estructura `mejor`, almacena el mejor recorrido y valor de la función objetivo hasta el momento, de modo que la estructura sirve como punto de comparación para las soluciones encontradas en cada iteración, además de que al término de la ejecución del algoritmo ésta estructura contendrá el mejor valor de la función objetivo encontrado por el algoritmo.

```
struct busca {
                    int b_costo;
                    struct lista_tabu b_rec[JOBS];
              }busqueda;
```

La estructura `búsqueda` hace una copia del recorrido correspondiente a la mejor solución de la función objetivo hasta el momento, con la finalidad de aplicarle la estructura de vecindad híbrida, de modo que el costo de ésta nueva solución vecina, es comparado con el de el mejor hasta el momento, si el costo de la nueva solución es mejor de acuerdo a la función objetivo, la mejor solución hasta el momento es reemplazada, con lo que se logra una mejora en la calidad de la solución.

APÉNDICE B

Cálculo de Complejidad Temporal por Pasos

Parte importante de un algoritmo es conocer su eficiencia, para lo cuál se lleva a cabo el cálculo de su complejidad. Considerando que cualquiera de las instrucciones de un algoritmo puede ser considerada un paso, sin tomar en cuenta ciclos, condiciones y funciones, cuyos pasos están en función de las operaciones realizadas, [Martínez y Martín, 2003] dan una explicación correspondiente a la complejidad (coste) en pasos para algunas instrucciones comúnmente utilizadas.

- La complejidad de declaraciones de variables, constantes y comentarios es de 0 pasos.

- Para el caso de expresiones, el número de pasos es 1 solo sí no se incluyen llamadas a funciones, de lo contrario, el coste será el correspondiente a la función. En caso de asignaciones simples, es decir, correspondiente a una variable, el coste es de 1 paso.

- Para el caso de sentencias condicionales, por ejemplo

```
If(<expresión1>) <sentencia 1> else <sentencia 2>
```

Donde, para calcular el número total de pasos, se suman los pasos correspondientes a la `expresión 1` con `sentencia 1`, o bien, con `sentencia 2`, según sea el caso.

- La complejidad correspondiente a sentencias iterativas se encuentra en función del número de pasos requeridos para resolver las expresiones de control de las sentencias. Por ejemplo, para:

```
for(<asignación>;<expresión 1>; <expresión 2> )
```

La complejidad será el número total de pasos necesarios para resolver

```
<asignación>, <expresión 1>,  <expresión 2>
```

Lo mismo sucede para el caso de

```
while(<expresión 3) y  do… while(<expresión 4>)
```

mismo que tendrá el número de pasos correspondientes a `expresión 3` y `expresión 4` respectivamente, cuyos pasos serán contados cada vez que el flujo del programa incluya dichas sentencias.

- Un caso especial es para una sentencia condicional múltiple `switch`, donde el número de pasos será el número de comparaciones necesarias para buscar sentencia por sentencia aquella requerida. Por ejemplo:

```
switch(n) {
   n=1: <sentencia>si n vale 1, el coste del switch es 1
   n=2: <sentencias>si n vale 2, el coste del switch es 2
   n=3: <sentencias>
```

```
    default: <sentencias> si n no es 1, ni 2, ni 3, el
    coste del switch es 3
}
```

- Llamadas a funciones: Constarán de un paso, salvo que dependan de algún parámetro que indique el tamaño del problema. En este caso el coste será el de la expresión correspondiente a la variable.

- En caso de una función recursiva, se debe tomar en cuenta el coste de las variables locales de la función.

A continuación se muestra un ejemplo del cálculo de la complejidad.

```
1    long mult2(int i) {
2      long aux = 0;
3         int j, k;
4         for(j = 0; j< i; j++)
5             for(k = 0; k < i; k++)
6                  aux = aux + 1;
7         return(aux);
8    }
```

Para este caso, se tiene que:

Líneas 1, 2 y 3:	0 pasos
Línea 4:	3 pasos (1 – asignación, 1- comparación y 1 - incremento), realizados *i* veces
Línea 5:	3 pasos (1 – asignación, 1 – comparación y 1 - incremento), realizados i^2 veces

Línea 6: 1 paso, ejecutado i^2 veces

Línea 7: 1 paso

De acuerdo a ésto, se tiene que la función temporal de ésta función es:

$$T(i): 3i + 3i^2 + i^2 + 1 = 4i^2 + 3i+1 \text{ pasos}$$

Teniendo como resultado un polinomio de segundo orden respecto a la entrada, por lo que la complejidad de la función es de $O(i^2)$.

APÉNDICE C

Código Fuente de la Estructura de Vecindad Híbrida aplicada en Búsqueda Local

A continuación se presenta el código fuente correspondiente a la búsqueda por vecindad con estructura híbrida, implementada al algoritmo Colonia de Hormigas aplicado al problema de Máquinas en Paralelo no Relacionadas.

```
// estructura Hibrida.cpp. Estructura compuesta de cuatro
diferentes estructuras de vecindad sencillas, mismas que
interactúan de forma aleatoria durante la ejecución del
algoritmo.


#include "stdafx.h"

#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>

#define N 4000       /* Numero de ciudades */

void leer_archivo(int *);
void crea_matriz();
void crea_vector();
void imprime_vect();
void distancias();
void permuta();
void compara();
```

```
void libera();

int mejor,dist,iter=0,bandera;
int ** matriz, * Vect_ini, * Vect_final;

FILE *ff;

int _tmain(int argc, _TCHAR* argv[])
{
   int m,i,j,cont=0;
   time_t timer1, timer2;

   system("cls");
   crea_matriz();
   crea_vector();

   for(i=0;i<N;i++)
   {
      Vect_ini[i]=i;
   }

   leer_archivo(&m);
   timer1=time(NULL);

   do{
      srand(time(NULL));
      permuta();
      distancias();
      compara();

      if(bandera==2)
      {
          mejor=dist;
          iter=1;
          for(j=0;j<N;j++)
            Vect_final[j]=Vect_ini[j];
      }
      else if(bandera==0)
      {
          iter++;
      }

      timer2=time(NULL)-timer1;
      cont++;
   }while(timer2<300);
```

```
   for(j=0;j<N;j++)
        printf("%d\t",Vect_final[j]);

   printf("\n\nLa mejor distancia es: %d \n",mejor);
   printf("\n Numero de repeticiones: %d \n", iter);
   printf("\n Numero de iteraciones totales: %d",cont);
   system("pause");
   libera();

   system("pause");
   return 0;
}

void crea_matriz()
{
   int x;

   matriz = (int**)malloc(sizeof(int*)*N);

   for(x=0; x<N; x++)
     matriz[x] = (int*)malloc(sizeof(int)*N);
}

void crea_vector()
{
   Vect_ini=(int*)malloc(sizeof(int)*N);
   Vect_final=(int*)malloc(sizeof(int)*N);
}

void libera()
{
  free(matriz);
  free(Vect_ini);
  free(Vect_final);
  printf("\n MEMORIA LIBERADA ");
}

void leer_archivo(int *m)
{
   int i,j;
   int dato;

   fflush(stdin);
```

```
   if((ff=fopen("matrizad.txt","r"))==NULL)
   {
     printf("\nNO SE PUDO ABRIR EL ARCHIVO DE LECTURA ");

      system("pause");
      exit(0);
   }

   fscanf(ff,"%d",m);

   for(i=0;i<N;i++)
      for(j=0;j<N;j++)
      {
        fscanf(ff,"%d",&dato);
        matriz[i][j]=dato;
      }
   fclose(ff);
}

void distancias()
{
   int i, aux, aux1;
   int Vect_res[N];

   dist=0;

   for(i=0; i<N; i++)
   {
      aux=Vect_ini[i];

      if((i>=0)&&(i<(N-1)))
      {
        aux1=Vect_ini[i+1];
        Vect_res[i]=matriz[aux][aux1];
      }
      else if(i==(N-1))
      {
        aux1=Vect_ini[0];
        Vect_res[i]=matriz[aux][aux1];
      }
   }

   for(i=0; i<N; i++)
      dist+=Vect_res[i];
```

```
}


void permuta()
{
   int aux=0,decide=0;
   int num=0, num1=0, num2=0, num3=0;
   int aux1=0, i, t=0;

   decide = (unsigned)rand()%4;

   switch(decide)
   {
     case 0:
          num = (rand()%(N-1))+1;

          if(num!=(N-1))
          {
               aux=Vect_ini[num];
               Vect_ini[num]=Vect_ini[num+1];
               Vect_ini[num+1]=aux;
          }
          else
          {
               aux=Vect_ini[num];
               Vect_ini[num]=Vect_ini[0];
               Vect_ini[0]=aux;
          }
     break;

     case 1:
          num = (rand()%(N-1))+1;
          num1 = (rand()%(N-1))+1;

          while((num1==num)||(num1==(num+1))||(num1==
          (num-1)))
          {
               num1=(rand()%(N-1))+1;
          }

          aux=Vect_ini[num];
          Vect_ini[num]=Vect_ini[num1];
          Vect_ini[num1]=aux;
     break;
```

```
case 2:
     num = (rand()%(N-1))+1;

     if(num!=(N-1))
     {
          aux=Vect_ini[num];
          Vect_ini[num]=Vect_ini[num+1];
          Vect_ini[num+1]=aux;
     }
     else
     {
          aux=Vect_ini[num];
          Vect_ini[num]=Vect_ini[0];
          Vect_ini[0]=aux;
     }

     num1 = (rand()%(N-1))+1;

     while((num1==num)||(num1==(num+1))||(num1==
     (num-1)))
     {
          num1 = (rand()%(N-1))+1;
     }

     if(num1!=(N-1))
     {
          aux=Vect_ini[num1];
          Vect_ini[num1] = Vect_ini[num1+1];
          Vect_ini[num1+1]= aux;
     }
     else
     {
          aux = Vect_ini[num1];
          Vect_ini[num1] = Vect_ini[0];
          Vect_ini[0] = aux;
     }
break;

case 3:
     num = (rand()%(N-1))+1;
     num1 = (rand()%(N-1))+1;
```

```
          while((num1==num)||(num1==(num+1))||(num1==
          (num-1)))
          {
               num1 = (rand()%(N-1))+1;
          }

          aux=Vect_ini[num];
          Vect_ini[num]=Vect_ini[num1];
          Vect_ini[num1]=aux;
          num2 = (rand()%(N-1))+1;

          while((num2==num)||(num2==(num+1))||(num2==
          (num1))||(num2==num1)||(num2==(num1+1))||
          (num2==(num1-1)))
          {
               num2 = (rand()%(N-1))+1;
          }

          num3 = (rand()%(N-1))+1;

          while((num3==num)||(num3==(num+1))||(num3==
          (num1))||(num3==num1)||(num3==(num1+1))||
          (num3==(num11))||(num3==num2)||(num3==(num2+1))
          || (num3==(num2-1)))
          {
               num3 = (rand()%(N-1))+1;
          }

          aux=Vect_ini[num2];
          Vect_ini[num2]=Vect_ini[num3];
          Vect_ini[num3]=aux;
     break;
     }
}

void compara()
{

   if(iter==0)
   {
      mejor=dist;
      bandera=2;
   }
   else
   {
```

```
      if(dist<mejor)
     {
         bandera=2;
     }
      else if(dist>mejor)
     {
          bandera=1;
     }
      else if(dist==mejor)
     {
          bandera=0;
     }
   }
}
```

APÉNDICE D

Código Fuente de la Estructura de Vecindad de un Par Aleatorio Aplicada a Búsqueda Local.

A continuación se presenta el código fuente correspondiente a la búsqueda por vecindad con la estructura sencilla de un par aleatorio aplicada al algoritmo Colonia de Hormigas propuesto para resolver el problema de Máquinas en Paralelo no Relacionadas.

```
// estructura Par_Aleat.cpp. Estructura sencilla que
realiza permutaciones de un par aleatorio

#include "stdafx.h"

#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>

#define N 4000       /* Numero de ciudades */

void leer_archivo(int *);
void crea_matriz();
void crea_vector();
void imprime_vect();
void distancias();
void permuta();
void compara();
void libera();

int mejor,dist,iter=0,bandera;
```

```
int ** matriz, * Vect_ini, * Vect_final;

FILE *ff;

int _tmain(int argc, _TCHAR* argv[])
{
   int m,i,j,cont=0;
   time_t timer1, timer2;

   system("cls");
   crea_matriz();
   crea_vector();

   for(i=0;i<N;i++)
   {
      Vect_ini[i]=i;
   }

   leer_archivo(&m);
   timer1=time(NULL);

   do{
      srand(time(NULL));
      permuta();
      distancias();
      compara();

      if(bandera==2)
      {
          mejor=dist;
          iter=1;
          for(j=0;j<N;j++)
           Vect_final[j]=Vect_ini[j];
      }
      else if(bandera==0)
      {
          iter++;
      }

      timer2=time(NULL)-timer1;
      cont++;
   }while(timer2<300);

   for(j=0;j<N;j++)
        printf("%d\t",Vect_final[j]);
```

```
   printf("\n\nLa mejor distancia es: %d \n",mejor);
   printf("\n Numero de repeticiones: %d \n", iter);
   printf("\n Numero de iteraciones totales: %d",cont);
   system("pause");
   libera();

   system("pause");

   return 0;
}

void crea_matriz()
{
   int x;

   matriz = (int**)malloc(sizeof(int*)*N);

   for(x=0; x<N; x++)
     matriz[x] = (int*)malloc(sizeof(int)*N);
}

void crea_vector()
{
   Vect_ini=(int*)malloc(sizeof(int)*N);
   Vect_final=(int*)malloc(sizeof(int)*N);
}

void libera()
{
  free(matriz);
  free(Vect_ini);
  free(Vect_final);
  printf("\n MEMORIA LIBERADA ");
}

void leer_archivo(int *m)
{
   int i,j;
   int dato;

   fflush(stdin);

   if((ff=fopen("matrizad.txt","r"))==NULL)
   {
```

```
     printf("\nNO SE PUDO ABRIR EL ARCHIVO DE LECTURA ");

      system("pause");
      exit(0);
   }

   fscanf(ff,"%d",m);

   for(i=0;i<N;i++)
      for(j=0;j<N;j++)
      {
        fscanf(ff,"%d",&dato);
        matriz[i][j]=dato;
      }

   fclose(ff);
}

void distancias()
{
   int i, aux, aux1;
   int Vect_res[N];

   dist=0;

   for(i=0; i<N; i++)
   {
      aux=Vect_ini[i];

      if((i>=0)&&(i<(N-1)))
      {
        aux1=Vect_ini[i+1];
        Vect_res[i]=matriz[aux][aux1];
      }
      else if(i==(N-1))
      {
        aux1=Vect_ini[0];
        Vect_res[i]=matriz[aux][aux1];
      }
   }

   for(i=0; i<N; i++)
      dist+=Vect_res[i];
}
```

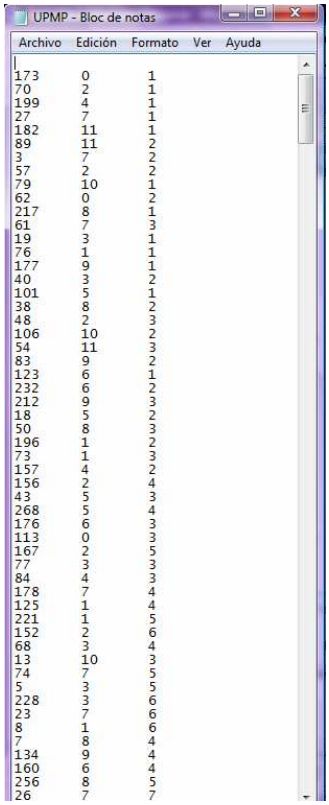
```
void permuta()
{
   int aux=0,decide=0;
   int num=0, num1=0, num2=0, num3=0;
   int aux1=0, i, t=0;

   num = (rand()%(N-1))+1;
   num1 = (rand()%(N-1))+1;

   while((num1==num)||(num1==(num+1))||(num1==(num-1)))
   {
       num1=(rand()%(N-1))+1;
   }

   aux=Vect_ini[num];
   Vect_ini[num]=Vect_ini[num1];
   Vect_ini[num1]=aux;

}

void compara()
{

   if(iter==0)
   {
      mejor=dist;
      bandera=2;
   }
   else
   {
      if(dist<mejor)
      {
         bandera=2;
      }
      else if(dist>mejor)
      {
          bandera=1;
      }
      else if(dist==mejor)
      {
          bandera=0;
      }
   }
}
```

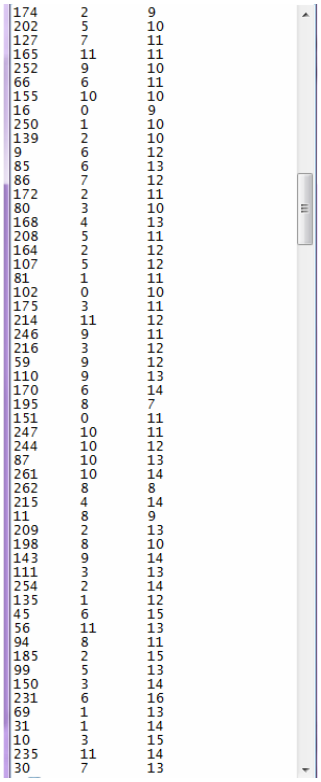
APÉNDICE E

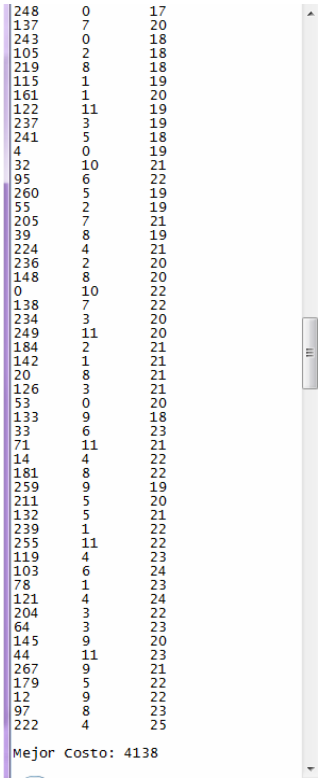
Ejecución del Algoritmo Colonia de Hormigas con una Estructura de Vecindad Híbrida Aplicada a Búsqueda Local

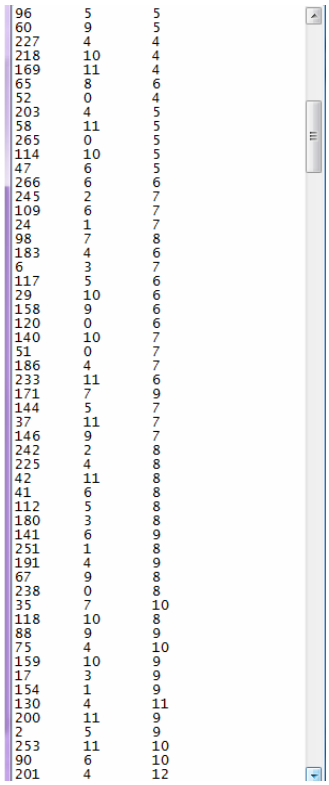
A continuación se presenta el resultado de la ejecución del algoritmo Colonia de Hormigas con una estructura de vecindad híbrida (versión 1) aplicado al problema de Máquinas en Paralelo no Relacionadas para una instancia de 270 trabajos por 12 máquinas.

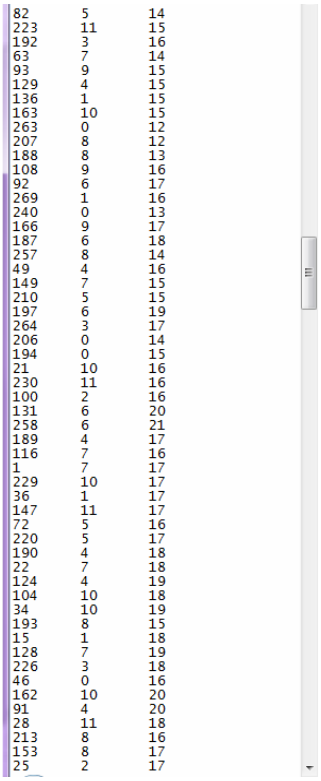
La calendarización y valor de la función objetivo obtenido por el algoritmo se almacenan directamente en un archivo de texto, tal y como se muestra a continuación


UPMP - Bloc de notas
Archivo Edición Formato Ver Ayuda
173 0 1
70 2 1
199 4 1
27 7 1
182 11 1
89 11 2
3 7 2
57 2 2
79 10 1
62 0 2
217 8 1
61 7 3
19 3 1
76 1 1
177 9 1
40 3 2
101 5 1
38 8 2
48 2 3
106 10 2
54 11 3
83 9 2
123 6 1
232 6 2
212 9 3
18 5 2
50 8 3
196 1 2
73 1 3
157 4 2
156 2 4
43 5 3
268 5 4
176 6 3
113 0 3
167 2 5
77 3 3
84 4 3
178 7 4
125 1 4
221 1 5
152 2 6
68 3 4
13 10 3
74 7 5
5 3 5
228 3 6
23 7 6
8 1 6
7 8 4
134 9 4
160 6 4
256 8 5
26 7 7

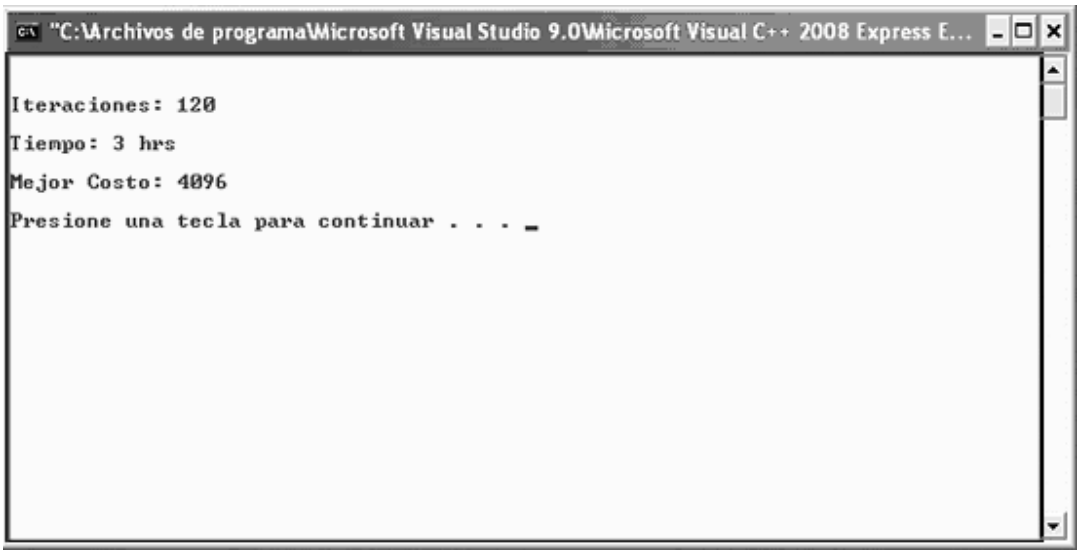

174 2 9
202 5 10
127 7 11
165 11 11
252 9 10
66 6 11
155 10 10
16 0 9
250 1 10
139 2 10
9 6 12
85 6 13
86 7 12
172 2 11
80 3 10
168 4 13
208 5 11
164 2 12
107 5 12
81 1 11
102 0 10
175 3 11
214 11 12
246 9 11
216 3 12
59 9 12
110 9 13
170 6 14
195 8 7
151 0 11
247 10 11
244 10 12
87 10 13
261 10 14
262 8 8
215 4 14
11 8 9
209 2 13
198 8 10
143 9 14
111 3 13
254 2 14
135 1 12
45 6 15
56 11 13
94 8 11
185 2 15
99 5 13
150 3 14
231 6 16
69 1 13
31 1 14
10 3 15
235 11 14
30 7 13


248 0 17
137 7 20
243 0 18
105 2 18
219 8 18
115 1 19
161 1 20
122 11 19
237 3 19
241 5 18
4 0 19
32 10 21
95 6 22
260 5 19
55 2 19
205 7 21
39 8 19
224 4 21
236 2 20
148 8 20
0 10 22
138 7 22
234 3 20
249 11 20
184 2 21
142 1 21
20 8 21
126 3 21
53 0 20
133 9 18
33 6 23
71 11 21
14 4 22
181 8 22
259 9 19
211 5 20
132 5 21
239 1 22
255 11 22
119 4 23
103 6 24
78 1 23
121 4 24
204 3 22
64 3 23
145 9 20
44 11 23
267 9 21
179 5 22
12 9 22
97 8 23
222 4 25
Mejor Costo: 4138

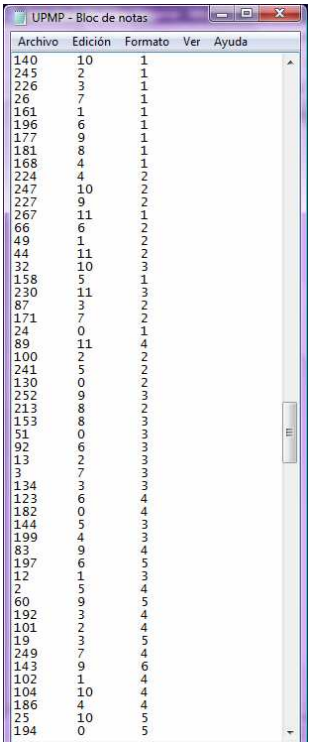

96 5 5
60 9 5
227 4 4
218 10 4
169 11 4
65 8 6
52 0 4
203 4 5
58 11 5
265 0 5
114 10 5
47 6 5
266 6 6
245 2 7
109 6 7
24 1 7
98 7 8
183 4 6
6 3 7
117 5 6
29 10 6
158 9 6
120 0 6
140 10 7
51 0 7
186 4 7
233 11 6
171 7 9
144 5 7
37 11 7
146 9 7
242 2 8
225 4 8
42 11 8
41 6 8
112 5 8
180 3 8
141 6 9
251 1 8
191 4 9
67 9 8
238 0 8
35 7 10
118 10 8
88 9 9
75 4 10
159 10 9
17 3 9
154 1 9
130 4 11
200 11 9
2 5 9
253 11 10
90 6 10
201 4 12

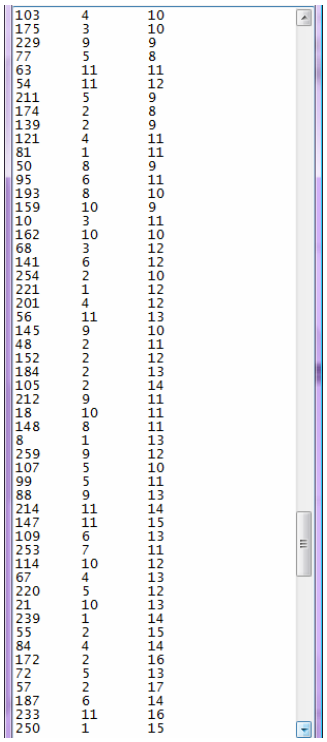

82 5 14
223 11 15
192 3 16
63 7 14
93 9 15
129 4 15
136 1 15
163 10 15
263 0 12
207 8 12
188 8 13
108 9 16
92 6 17
269 1 16
240 0 13
166 9 17
187 6 18
257 8 14
49 4 16
149 7 15
210 5 15
197 6 19
264 3 17
206 0 14
194 0 15
21 10 16
230 11 16
100 2 16
131 6 20
258 6 21
189 4 17
116 7 16
1 7 17
229 10 17
36 1 17
147 11 17
72 5 16
220 5 17
190 4 18
22 7 18
124 4 19
104 10 18
34 10 19
193 8 15
15 1 18
128 7 19
226 3 18
46 0 16
162 10 20
91 4 20
28 11 18
213 8 16
153 8 17
25 2 17

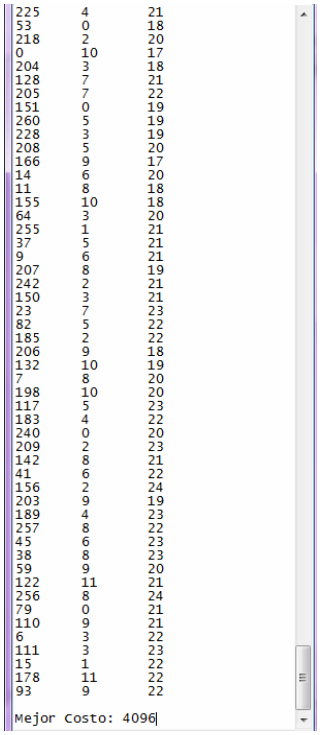
Las siguientes pantallas corresponden al resultado obtenido de la ejecución del algoritmo Colonia de Hormigas con una estructura de vecindad sencilla de un par aleatorio (versión 2) para el problema de Máquinas en Paralelo no Relacionadas para una instancia de 270 trabajos por 12 máquinas.

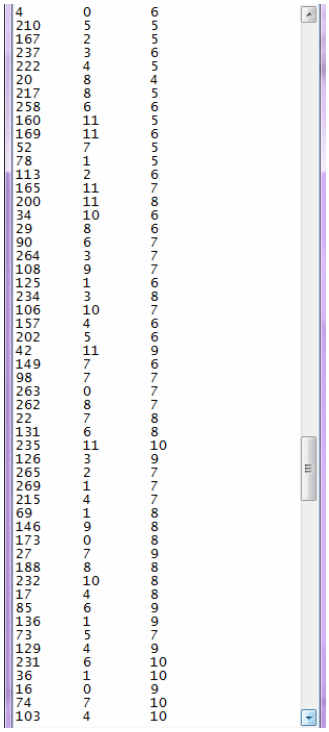


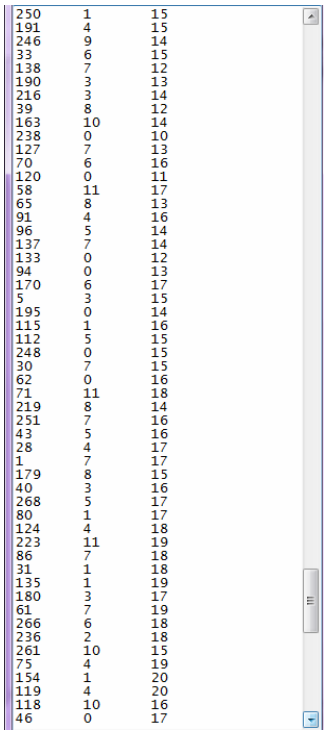
Al igual que en los resultados de la versión 1 del algoritmo, presentados anteriormente, la calendarización y valor de la función objetivo obtenido por el algoritmo se almacenan directamente en un archivo de texto, tal y como se muestra a continuación.


UPMP - Bloc de notas
Archivo Edición Formato Ver Ayuda
140 10 1
245 2 1
226 3 1
26 7 1
161 1 1
196 6 1
177 9 1
181 8 1
168 4 1
224 4 2
247 10 2
227 9 2
267 11 1
66 6 2
49 1 2
44 11 2
32 10 3
158 5 1
230 11 3
87 3 2
171 7 2
24 0 1
89 11 4
100 2 2
241 5 2
130 0 2
252 9 3
213 8 2
153 8 3
51 0 3
92 6 3
13 2 3
3 7 3
134 3 3
123 6 4
182 0 4
144 5 3
199 4 3
83 9 4
197 6 5
12 1 3
2 5 4
60 9 5
192 3 4
101 2 4
19 3 5
249 7 4
143 9 6
102 1 4
104 10 4
186 4 4
25 10 5
194 0 5


103 4 10
175 3 10
229 9 9
77 5 8
63 11 11
54 11 12
211 5 9
174 2 8
139 2 9
121 4 11
81 1 11
50 8 9
95 6 11
193 8 10
159 10 9
10 3 11
162 10 10
68 3 12
141 6 12
254 2 10
221 1 12
201 4 12
56 11 13
145 9 10
48 2 11
152 2 12
184 2 13
105 2 14
212 9 11
18 10 11
148 8 11
8 1 13
259 9 12
107 5 10
99 5 11
88 9 13
214 11 14
147 11 15
109 6 13
253 7 11
114 10 12
67 4 13
220 5 12
21 10 13
239 1 14
55 2 15
84 4 14
172 2 16
72 5 13
57 2 17
187 6 14
233 11 16
250 1 15


225 4 21
53 0 18
218 2 20
0 10 17
204 3 18
128 7 21
205 7 22
151 0 19
260 5 19
228 3 19
208 5 20
166 9 17
14 6 20
11 8 18
155 10 18
64 3 20
255 1 21
37 5 21
9 6 21
207 8 19
242 2 21
150 3 21
23 7 23
82 5 22
185 2 22
206 9 18
132 10 19
7 8 20
198 10 20
117 5 23
183 4 22
240 0 20
209 2 23
142 8 21
41 6 22
156 2 24
203 9 19
189 4 23
257 8 22
45 6 23
38 8 23
59 9 20
122 11 21
256 8 24
79 0 21
110 9 21
6 3 22
111 3 23
15 1 22
178 11 22
93 9 22
Mejor Costo: 4096


4 0 6
210 5 5
167 2 5
237 3 6
222 4 5
20 8 4
217 8 5
258 6 6
160 11 5
169 11 6
52 7 5
78 1 5
113 2 6
165 11 7
200 11 8
34 10 6
29 8 6
90 6 7
264 3 7
108 9 7
125 1 6
234 3 8
106 10 7
157 4 6
202 5 6
42 11 9
149 7 6
98 7 7
263 0 7
262 8 7
22 7 8
131 6 8
235 11 10
126 3 9
265 2 7
269 1 7
215 4 7
69 1 8
146 9 8
173 0 8
27 7 9
188 8 8
232 10 8
17 4 8
85 6 9
136 1 9
73 5 7
129 4 9
231 6 10
36 1 10
16 0 9
74 7 10
103 4 10


250 1 15
191 4 15
246 9 14
33 6 15
138 7 12
190 3 13
216 3 14
39 8 12
163 10 14
238 0 10
127 7 13
70 6 16
120 0 11
58 11 17
65 8 13
91 4 16
96 5 14
137 7 14
133 0 12
94 0 13
170 6 17
5 3 15
195 0 14
115 1 16
112 5 15
248 0 15
30 7 15
62 0 16
71 11 18
219 8 14
251 7 16
43 5 16
28 4 17
1 7 17
179 8 15
40 3 16
268 5 17
80 1 17
124 4 18
223 11 19
86 7 18
31 1 18
135 1 19
180 3 17
61 7 19
266 6 18
236 2 18
261 10 15
75 4 19
154 1 20
119 4 20
118 10 16
46 0 17

APÉNDICE F

Instancias de Prueba Generadas Aleatoriamente

A continuación se presentan las instancias de prueba generadas de forma aleatoria para realizar las pruebas experimentales al algoritmo Colonia de Hormigas con Estructura de Vecindad aplicado para resolver el UPMP. Las instancias generadas fueron 4, las cuáles fueron nombrada MC_12_200, MC_12_250, MC_12_270 y MC_12_300, de acuerdo a los primeros apellidos de los autores, así como al tamaño de cada instancia, correspondiente a 200, 250, 270 y 300 trabajos respectivamente, los cuales requieren ser calendarizados en 12 máquinas. Cabe mencionar que solo se muestra el costo de cada trabajo para la primera posición de cada máquina, debido a al espacio que requiere el mostrar las instancias completas.

INSTANCIA MC_12_200

	MÁQUINAS											
TRAB.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	1	13	8	9	9	12	15	15	14	2	13	5
2	2	13	3	3	11	3	12	2	3	12	6	4
3	12	5	13	7	8	6	2	14	10	14	14	7
4	14	15	15	9	1	12	13	5	14	13	2	5
5	9	6	5	12	4	5	15	6	4	15	14	1
6	6	1	3	13	2	14	6	9	14	11	7	11
7	6	13	4	5	4	13	9	6	10	3	6	6
8	7	13	15	1	1	10	9	1	1	12	9	7
9	8	9	12	13	13	13	5	8	12	14	15	11
10	9	15	1	10	3	5	13	11	9	11	6	5
11	8	4	9	10	9	10	2	4	8	2	5	7
12	15	4	3	9	4	13	7	14	3	14	6	6
13	9	10	11	3	12	2	3	5	4	6	13	2
14	1	11	7	12	3	8	6	3	6	13	1	13
15	12	8	10	6	13	9	3	3	14	10	13	2
16	15	9	7	7	14	2	1	8	15	4	5	6
17	9	4	3	1	15	6	7	12	11	5	6	2
18	5	9	14	6	14	11	5	4	3	15	11	11
19	4	13	2	9	9	6	15	6	9	6	10	6
20	4	4	8	6	12	12	13	10	14	7	1	13
21	13	9	6	5	3	14	9	15	12	3	12	14
22	4	9	10	4	7	7	9	2	5	14	11	15
23	10	7	4	8	4	1	9	5	12	13	4	10
24	12	1	3	1	15	1	1	8	7	2	11	2
25	15	12	6	14	14	8	11	13	6	2	12	14
26	9	3	10	10	6	5	3	9	6	2	1	4
27	2	8	2	7	3	8	13	4	8	3	14	9
28	2	8	11	13	12	6	12	6	1	12	4	5

29	4	11	14	15	3	12	5	14	1	1	9	2
30	7	5	15	6	5	14	1	12	12	5	11	3
31	4	2	15	1	11	3	15	5	6	2	12	12
32	13	9	10	10	13	11	11	6	8	4	7	8
33	15	14	10	11	11	1	6	7	14	11	3	14
34	11	2	14	7	6	13	8	7	13	4	4	2
35	1	12	15	11	15	14	8	14	6	13	5	13
36	3	2	7	6	15	4	3	14	5	2	3	8
37	5	1	1	14	11	9	3	8	1	15	4	5
38	15	4	9	7	9	11	12	7	5	7	2	9
39	5	3	5	14	12	6	6	5	10	13	8	12
40	15	7	11	13	10	3	9	9	7	14	15	4
41	3	3	11	4	15	10	7	8	14	4	14	13
42	12	3	9	1	4	14	13	1	3	11	1	10
43	1	4	6	1	11	9	3	3	1	10	14	14
44	11	3	3	12	13	1	6	15	13	9	8	13
45	11	3	15	13	2	15	13	14	13	5	10	15
46	1	10	8	3	15	11	15	10	11	14	3	3
47	14	13	7	3	7	4	6	7	6	14	10	4
48	7	3	5	8	6	13	13	1	9	5	15	1
49	10	8	4	9	10	6	1	12	9	11	3	9
50	7	13	12	13	9	9	13	2	5	11	5	13
51	12	8	10	12	6	3	8	2	3	14	7	5
52	9	1	1	5	14	14	12	13	14	15	15	1
53	13	11	7	13	4	2	9	11	1	10	13	3
54	11	1	8	11	9	9	8	12	7	11	1	3
55	8	2	13	11	9	5	9	1	6	14	12	7
56	2	13	5	8	15	14	2	12	8	12	14	14
57	10	9	8	8	9	1	3	1	13	12	2	3
58	7	1	7	13	7	15	14	10	11	3	6	6
59	15	14	8	7	7	12	5	15	3	10	15	5
60	5	14	14	3	12	10	4	12	12	2	4	5
61	8	8	11	3	4	12	1	9	2	14	3	11
62	1	8	11	1	5	9	1	11	2	10	8	12
63	8	1	8	1	8	6	4	12	11	7	11	6
64	3	7	3	11	9	9	9	3	6	9	8	1
65	9	13	12	15	8	12	4	14	4	5	2	13
66	4	6	12	1	11	11	5	5	1	2	10	13
67	4	1	7	14	4	10	6	10	15	5	6	14
68	1	4	1	10	7	4	4	3	15	8	9	13
69	7	5	9	4	11	4	4	1	15	8	13	14
70	13	15	5	12	3	9	12	10	5	5	4	1
71	10	7	10	5	12	11	15	6	15	12	7	10
72	5	12	5	6	7	3	14	10	7	12	13	6
73	8	4	5	9	3	2	11	9	15	7	4	11
74	14	7	4	13	5	12	8	10	9	8	5	4
75	7	1	4	8	1	2	8	13	8	1	9	2
76	1	8	6	3	12	15	10	13	10	10	12	12
77	5	9	12	3	15	14	3	8	12	7	10	12
78	1	12	1	14	5	11	11	1	8	4	10	5
79	1	2	5	11	9	9	6	5	8	15	14	4
80	6	1	4	3	2	15	4	7	4	7	15	15
81	9	8	15	5	6	13	13	3	6	13	15	4
82	11	7	15	15	5	6	5	2	3	15	8	7
83	7	2	4	8	6	2	7	4	12	8	9	15
84	5	4	1	10	15	7	8	4	15	15	15	10
85	2	4	10	6	8	13	7	2	12	11	5	8
86	1	10	2	7	15	3	9	6	9	10	11	14
87	12	6	10	9	10	1	11	5	3	9	9	1
88	15	1	14	7	14	6	12	12	4	10	9	11
89	4	2	15	1	10	10	4	4	12	11	7	1
90	14	3	2	6	8	9	10	11	15	11	7	1
91	1	9	5	4	13	1	9	2	7	15	13	5
92	7	1	3	4	4	2	10	5	6	5	2	2
93	3	6	9	4	15	8	9	9	6	2	5	4
94	3	13	13	11	9	7	8	8	4	10	15	1
95	5	11	4	9	15	2	3	12	2	13	7	2
96	2	11	13	3	5	7	13	15	12	1	7	15
97	4	15	10	6	12	6	4	11	8	14	1	8
98	2	2	4	10	10	5	6	11	2	9	3	4
99	13	12	6	5	12	9	3	15	10	5	13	10
100	3	3	7	4	15	14	3	1	7	12	13	7
101	6	10	11	4	8	9	11	7	10	11	2	12
102	10	13	13	9	11	1	1	7	12	4	10	10
103	13	11	5	15	4	4	6	4	15	15	1	10
104	10	10	7	1	15	6	1	10	2	14	1	9
105	6	2	7	7	1	15	7	11	2	12	2	1
106	6	2	7	11	12	12	2	1	15	13	15	1
107	5	3	5	14	1	12	11	7	14	5	10	11
108	11	14	6	14	4	13	7	13	14	4	9	12
109	13	5	8	11	9	5	14	12	4	7	7	11
110	15	2	8	4	2	11	9	13	8	1	1	1
111	4	14	7	11	9	4	4	2	11	4	5	9

112	2	12	7	8	1	9	4	7	13	11	12	7
113	13	10	9	3	13	8	3	15	2	5	14	9
114	1	4	1	8	10	1	15	15	8	8	15	1
115	7	5	2	3	11	5	5	8	3	1	11	13
116	12	9	12	5	11	6	9	9	8	8	1	7
117	9	15	14	13	11	7	5	9	8	1	13	3
118	4	5	15	4	7	5	3	9	15	14	15	4
119	9	8	5	2	6	5	13	12	10	2	6	10
120	14	13	2	11	7	9	9	14	10	4	3	2
121	15	15	9	12	13	15	1	7	7	2	13	5
122	1	2	8	5	13	8	11	5	13	8	7	10
123	14	3	9	9	15	8	2	4	4	9	10	12
124	6	2	10	1	3	6	8	7	4	4	11	5
125	11	12	11	13	1	5	2	6	15	3	14	2
126	11	15	4	1	7	6	9	13	1	15	6	1
127	4	5	3	8	3	5	9	8	9	3	4	1
128	14	9	12	7	10	8	7	9	3	9	1	8
129	6	2	10	1	11	1	8	15	5	4	4	3
130	6	14	11	1	8	14	9	4	4	15	1	14
131	1	10	12	12	10	12	6	15	6	2	15	10
132	12	2	13	4	11	6	14	5	11	12	12	1
133	14	8	4	9	13	14	13	1	10	11	12	3
134	11	13	9	7	1	5	1	12	5	12	14	4
135	4	9	12	9	3	2	8	10	2	12	1	15
136	5	12	12	5	6	13	10	15	14	4	2	1
137	10	10	13	15	14	8	4	5	15	7	8	4
138	12	13	10	14	2	12	5	5	2	9	3	9
139	6	5	3	9	4	8	8	10	1	11	6	12
140	1	9	14	13	4	5	4	13	11	6	13	10
141	9	11	9	8	11	5	12	15	6	14	9	3
142	13	4	4	7	10	12	12	10	15	10	11	1
143	4	14	11	8	9	11	4	7	13	11	4	15
144	12	12	7	15	13	8	14	15	1	8	11	1
145	10	5	8	14	7	10	1	2	1	14	1	12
146	9	6	12	10	13	8	10	5	8	9	6	9
147	5	2	3	8	5	4	8	2	12	14	4	9
148	9	5	11	5	10	15	9	3	10	10	1	2
149	3	15	8	6	1	1	8	8	7	13	1	9
150	4	9	9	12	8	12	1	6	15	11	8	10
151	15	2	13	1	5	2	6	9	2	1	15	4
152	5	10	2	6	2	10	1	15	9	1	8	7
153	6	14	2	11	14	13	12	11	6	2	4	14
154	12	5	10	6	15	4	14	3	2	4	9	10
155	4	14	6	14	11	2	12	2	1	13	6	3
156	3	14	6	1	4	8	8	15	14	14	15	11
157	6	9	7	10	4	14	5	7	8	6	11	10
158	11	5	1	5	12	4	10	10	2	1	12	3
159	15	6	13	3	2	9	14	2	13	5	15	4
160	11	5	10	10	2	5	8	5	9	4	13	4
161	4	12	8	1	10	2	5	9	5	14	10	10
162	15	11	1	14	11	1	12	15	11	8	1	10
163	8	12	11	2	5	4	12	8	14	11	4	2
164	11	6	13	15	3	4	10	4	12	2	15	7
165	15	3	8	5	1	1	5	14	2	4	13	6
166	3	14	14	2	2	6	6	6	5	13	13	5
167	4	13	2	1	15	4	14	5	11	2	11	8
168	15	15	1	3	4	5	10	6	14	14	15	8
169	3	4	5	13	9	15	12	6	3	7	8	10
170	15	3	12	1	8	4	2	9	12	7	1	3
171	9	2	5	13	9	14	15	12	12	5	2	13
172	3	15	2	6	8	9	2	2	2	1	15	1
173	6	4	9	12	8	5	6	4	11	1	2	6
174	11	2	4	9	14	3	2	7	2	6	12	9
175	9	12	5	1	15	7	15	9	4	2	7	15
176	2	12	8	15	3	12	4	12	3	1	5	9
177	15	15	14	7	13	1	5	9	10	2	14	12
178	11	5	10	2	11	2	11	2	15	7	14	8
179	11	5	10	6	11	15	6	12	7	7	15	11
180	2	12	3	9	15	7	8	7	11	12	15	5
181	3	8	10	2	5	10	10	15	3	10	13	15
182	2	1	11	4	7	7	8	7	10	15	7	7
183	10	13	4	11	15	1	13	1	15	11	5	7
184	5	1	10	13	2	4	12	12	11	11	1	12
185	10	13	5	6	1	9	10	13	2	5	6	9
186	2	13	13	6	11	9	15	4	15	4	9	2
187	10	14	12	7	3	13	1	15	1	3	11	11
188	12	12	4	4	3	12	1	11	6	15	7	8
189	1	10	11	1	12	9	6	11	2	11	2	4
190	10	7	1	3	8	7	8	1	11	1	6	7
191	3	10	13	15	10	15	9	15	5	9	6	13
192	7	10	4	2	7	1	3	1	15	9	3	9
193	13	5	9	15	3	13	11	14	9	10	4	6
194	15	2	5	14	2	15	10	8	11	9	10	2

195	4	5	4	2	3	15	6	2	12	7	13	1
196	11	3	10	7	10	5	5	5	2	4	11	4
197	14	10	12	14	4	2	5	15	3	14	5	6
198	14	4	13	13	12	10	1	3	14	3	1	6
199	2	15	6	10	12	4	13	8	1	14	8	7
200	14	8	13	1	11	5	4	11	7	5	15	10

INSTANCIA MC_12_250

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

59	15	12	10	4	3	9	11	3	10	11	6	7
60	5	3	11	5	9	13	12	3	11	5	7	2
61	15	8	10	11	7	7	5	8	12	3	3	4
62	14	2	13	3	7	8	7	12	6	1	4	7
63	3	12	1	4	5	7	5	7	11	6	7	6
64	15	5	1	10	6	7	13	11	5	14	3	6
65	14	13	13	15	5	13	12	7	11	6	3	4
66	13	13	13	4	8	3	15	5	3	15	10	15
67	9	11	3	13	2	15	4	14	7	6	15	5
68	8	2	14	1	6	7	7	3	11	5	5	3
69	15	3	13	1	7	15	3	2	9	5	13	8
70	5	1	8	7	15	15	11	13	4	13	13	9
71	3	7	2	12	8	10	10	12	8	10	5	15
72	10	11	1	6	11	4	13	12	15	11	14	8
73	8	6	6	10	11	1	14	9	3	14	8	15
74	3	1	8	1	15	13	2	6	6	15	3	7
75	15	14	3	12	4	3	11	14	1	5	12	8
76	11	13	13	6	14	12	13	2	7	6	3	5
77	15	1	10	10	12	6	12	11	6	7	12	6
78	10	9	13	8	13	2	3	5	11	11	6	9
79	8	7	4	7	8	14	13	4	15	9	9	2
80	15	4	5	2	6	8	8	15	2	4	8	7
81	1	1	12	10	2	5	8	13	13	1	13	3
82	11	8	1	9	11	12	8	9	4	8	13	6
83	13	8	11	3	15	11	1	13	15	8	10	3
84	8	2	8	1	10	8	4	12	13	9	15	11
85	11	7	10	11	6	4	11	12	4	11	4	4
86	8	6	7	1	9	13	14	12	1	6	4	3
87	13	14	7	5	8	5	9	1	15	7	9	6
88	7	14	15	4	11	5	12	8	14	13	9	8
89	4	13	2	13	14	9	13	13	7	4	6	5
90	10	12	14	15	4	9	3	2	12	15	12	13
91	6	2	9	1	3	15	13	6	8	6	5	2
92	5	8	11	6	15	10	8	13	6	4	13	1
93	10	13	2	11	8	8	2	6	14	2	12	8
94	14	7	6	13	2	6	6	11	11	4	12	11
95	9	11	4	4	6	11	13	15	1	2	11	14
96	12	1	2	9	4	15	8	2	14	11	10	13
97	8	12	5	1	2	15	5	4	8	7	14	3
98	8	8	3	10	11	11	14	3	6	9	1	1
99	2	9	13	14	15	7	2	14	5	14	7	12
100	5	6	4	13	15	8	5	6	1	3	4	5
101	2	2	1	3	1	13	8	12	1	13	3	13
102	8	1	2	4	9	5	9	15	10	10	4	3
103	8	1	3	4	1	2	3	14	11	12	13	9
104	4	11	3	11	9	2	2	4	15	12	10	9
105	14	14	7	14	3	15	2	6	6	3	10	13
106	13	5	12	15	4	3	9	12	6	14	2	10
107	13	5	15	7	13	1	13	14	4	9	6	10
108	10	3	15	7	12	1	14	6	9	11	1	8
109	4	4	8	7	8	15	13	4	7	6	12	6
110	13	11	9	13	11	6	7	4	14	7	14	11
111	3	13	9	1	8	7	6	8	1	1	4	8
112	12	4	5	2	5	11	4	12	3	3	15	10
113	7	5	5	14	14	9	1	1	14	4	11	14
114	5	6	1	3	15	7	13	14	13	9	15	4
115	9	12	10	6	9	5	13	2	11	6	1	8
116	2	5	15	15	1	8	5	9	12	11	13	12
117	7	2	15	3	15	6	10	14	14	8	12	2
118	15	9	10	14	5	2	4	10	9	1	1	2
119	2	13	13	11	5	12	15	14	6	9	9	11
120	2	10	13	13	11	7	4	8	4	13	2	15
121	11	9	13	14	7	7	5	15	6	7	3	5
122	15	6	1	15	7	12	2	6	10	4	9	13
123	1	1	5	5	13	11	1	12	5	4	5	10
124	15	15	8	4	1	2	4	4	4	6	12	8
125	10	6	9	15	9	12	13	4	9	12	7	13
126	15	7	10	10	11	5	1	5	7	13	11	5
127	15	1	13	13	4	1	11	1	14	14	11	2
128	15	8	11	10	1	3	4	9	5	3	12	5
129	1	13	5	14	10	9	14	11	8	9	4	5
130	11	6	14	13	12	7	14	13	9	10	5	4

131	5	13	4	4	1	12	4	5	8	14	15	3
132	11	6	11	13	3	15	10	1	8	11	4	1
133	6	4	8	8	9	15	12	12	4	13	5	7
134	15	14	15	3	6	9	5	8	9	12	15	12
135	15	2	12	4	14	13	6	7	15	5	10	15
136	5	12	11	11	5	3	15	10	10	7	5	11
137	2	2	8	7	13	2	2	4	10	5	3	2
138	13	4	8	13	4	12	8	14	2	5	1	4
139	2	15	15	15	14	9	12	5	15	2	11	8
140	4	13	1	6	5	13	2	15	7	7	1	1
141	10	14	12	15	10	14	12	10	11	6	3	15
142	14	4	5	1	4	4	1	10	1	13	1	5
143	14	3	14	6	3	3	5	10	1	13	1	1
144	12	6	4	10	6	7	6	2	4	2	7	14
145	5	11	2	10	2	10	6	14	9	9	4	7
146	5	2	1	1	1	4	5	3	5	1	14	7
147	5	4	4	5	2	2	12	15	8	4	13	10
148	12	10	11	2	8	8	14	8	11	9	6	4
149	3	5	10	6	5	6	1	9	9	9	15	9
150	14	10	6	15	15	15	5	6	3	6	12	2
151	1	13	15	6	7	3	1	15	3	13	6	15
152	2	8	15	7	6	1	11	5	14	7	5	3
153	2	10	13	1	12	5	3	1	15	1	9	11
154	10	11	10	14	4	1	9	6	3	9	6	15
155	13	5	15	13	2	1	15	3	13	1	6	12
156	13	1	1	8	13	9	3	4	3	7	12	1
157	11	2	7	11	13	12	6	4	2	10	5	11
158	1	5	12	7	10	15	4	11	1	15	5	12
159	15	5	1	5	12	12	14	14	15	6	6	9
160	12	3	4	7	5	2	11	1	12	1	3	14
161	9	10	11	8	3	11	8	4	11	11	4	13
162	14	5	9	12	9	2	9	7	7	1	7	2
163	3	2	3	3	2	13	14	6	6	14	1	3
164	4	5	1	2	7	6	9	9	14	1	9	15
165	1	2	12	2	8	8	8	15	10	4	8	2
166	3	1	4	8	10	8	7	3	2	7	14	6
167	3	4	3	10	15	10	14	8	9	12	11	6
168	6	13	1	8	8	6	10	13	14	3	8	5
169	12	5	1	12	6	1	4	2	15	5	7	2
170	3	10	5	14	12	13	7	13	13	12	5	12
171	14	5	5	3	3	3	11	4	2	11	6	13
172	3	6	9	10	9	10	9	7	12	12	3	10
173	1	11	8	9	12	8	6	5	3	3	4	15
174	9	9	11	12	3	15	1	11	4	15	15	3
175	10	13	7	6	15	15	9	14	11	3	15	8
176	3	4	8	10	4	7	10	13	2	6	14	15
177	14	15	9	1	4	1	12	7	15	13	9	2
178	12	8	7	15	5	11	13	6	3	7	10	9
179	9	13	4	4	6	9	7	2	9	3	6	11
180	11	9	1	7	1	12	12	13	15	9	3	5
181	12	11	1	7	6	3	12	13	3	4	2	9
182	3	4	12	2	3	15	6	14	15	7	14	2
183	11	5	14	1	5	4	8	8	5	15	15	11
184	8	7	13	9	4	15	12	13	6	1	6	6
185	4	11	15	6	9	4	7	11	10	8	11	1
186	1	12	7	6	15	8	7	5	15	7	6	4
187	15	15	5	15	5	5	9	12	6	14	15	2
188	9	11	2	1	12	7	8	3	3	6	1	1
189	3	12	11	14	8	1	10	5	10	9	10	13
190	15	5	6	7	3	1	9	1	4	8	15	2
191	2	13	15	9	12	3	10	12	3	4	9	15
192	11	8	7	8	9	3	11	11	13	10	13	8
193	9	7	2	1	12	14	13	15	9	5	4	12
194	6	9	4	2	15	13	4	15	15	14	10	9
195	13	11	14	2	2	6	1	6	3	10	11	13
196	9	3	3	15	4	9	14	11	10	11	10	6
197	14	9	13	11	10	7	14	9	13	9	1	2
198	7	3	14	15	3	11	7	9	7	2	5	15
199	1	5	1	12	4	14	12	14	13	7	7	12
200	11	10	14	7	12	4	13	2	11	4	5	15
201	5	10	15	10	6	6	6	8	15	3	9	10
202	3	13	5	14	10	15	11	4	2	12	5	10

203	3	3	2	15	11	1	11	9	13	7	4	1
204	9	13	5	10	1	6	2	15	3	9	8	8
205	12	12	9	11	9	15	2	12	7	6	1	12
206	5	1	12	5	5	10	6	15	9	5	10	7
207	1	5	15	4	5	1	15	2	6	12	1	6
208	2	15	8	2	5	5	4	3	3	4	8	9
209	1	5	11	11	2	11	10	2	10	15	8	13
210	1	15	14	2	7	10	7	10	10	13	5	1
211	5	15	13	9	12	15	14	4	3	1	3	3
212	15	3	12	7	4	14	2	2	10	14	13	4
213	7	6	4	8	14	8	2	2	5	15	5	15
214	13	5	14	7	13	6	2	15	2	5	3	1
215	9	11	7	4	11	6	14	12	14	9	11	1
216	15	2	11	9	7	12	12	14	1	4	5	13
217	12	9	12	14	13	12	4	13	14	11	14	3
218	9	3	14	9	14	8	3	10	9	14	1	9
219	12	7	1	10	6	5	8	13	3	9	1	11
220	14	10	2	13	1	4	8	7	2	13	8	4
221	12	3	13	4	2	3	12	5	11	1	10	3
222	2	14	14	14	1	9	8	14	4	9	15	5
223	12	10	15	14	5	9	14	7	11	11	6	6
224	15	4	5	4	6	4	6	1	12	4	8	3
225	10	5	13	11	2	12	5	6	11	13	13	14
226	2	15	1	8	9	5	14	4	1	6	4	3
227	5	8	1	13	7	11	4	15	4	13	4	13
228	5	8	13	2	4	8	4	6	10	13	11	8
229	12	11	2	4	1	6	13	7	5	9	7	8
230	4	10	9	12	4	7	3	12	3	12	8	8
231	3	3	2	8	9	5	3	7	7	8	14	13
232	7	2	3	3	8	9	12	7	14	5	5	8
233	7	12	11	9	2	1	5	14	1	11	5	10
234	11	10	9	9	9	7	7	3	9	1	5	6
235	15	9	11	11	11	1	5	7	1	15	13	6
236	9	5	2	1	4	8	5	9	6	8	2	6
237	7	5	8	9	11	12	11	1	15	12	13	11
238	9	6	11	14	14	15	15	2	11	1	6	8
239	9	15	1	15	15	5	14	6	11	11	9	13
240	10	15	7	8	11	6	13	11	6	9	14	7
241	7	13	7	11	11	6	5	6	15	8	3	8
242	15	10	1	11	4	2	8	2	7	5	13	14
243	1	2	4	12	10	15	5	2	3	14	10	13
244	12	2	12	2	8	10	8	2	7	15	15	13
245	6	9	2	10	1	6	14	14	7	11	6	1
246	2	10	12	5	2	7	12	4	4	15	15	6
247	11	6	13	10	3	6	12	15	12	7	10	10
248	2	7	13	1	14	11	13	6	14	15	5	4
249	8	12	11	13	14	1	10	8	4	9	4	13
250	10	14	11	14	5	9	5	11	5	2	2	9

INSTANCIA MC_12_270

	MÁQUINAS											
TRAB.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	5	5	15	9	10	9	4	14	4	3	1	13
2	10	11	8	4	9	10	7	2	4	6	7	10
3	12	7	7	12	6	2	4	11	4	11	4	13
4	2	3	6	14	5	9	5	1	4	13	6	5
5	1	11	3	12	9	9	9	10	9	7	2	2
6	6	7	4	1	7	13	11	14	10	15	14	12
7	15	11	7	2	12	15	2	2	6	3	14	4
8	15	11	13	9	11	7	3	7	2	9	15	2
9	5	3	5	9	14	7	14	15	13	7	9	6
10	9	10	5	2	13	2	1	4	12	13	2	5
11	12	13	9	1	3	5	12	6	7	8	3	5
12	15	4	10	3	13	8	14	13	2	7	12	6
13	15	4	14	12	9	13	9	14	14	3	15	10
14	6	15	1	4	8	2	8	8	1	11	1	10
15	12	9	8	14	1	11	1	7	13	7	3	10

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

88	7	14	3	1	15	15	15	2	13	5	1	14
89	8	11	10	11	11	13	15	11	4	2	6	2
90	10	13	12	14	7	10	7	8	12	12	12	3
91	11	5	6	11	6	9	2	8	12	6	11	14
92	11	9	15	10	1	14	12	6	3	5	2	11
93	4	12	13	5	4	9	3	11	7	12	14	11
94	4	2	5	7	7	2	13	10	4	1	14	1
95	1	11	3	5	13	2	1	9	1	15	14	3
96	9	14	3	7	11	8	1	3	9	5	11	3
97	9	3	2	13	5	2	15	5	7	6	12	15
98	9	6	13	1	15	9	4	13	1	14	11	11
99	8	14	15	6	3	9	15	2	6	8	10	14
100	3	2	10	7	15	2	3	5	4	7	6	3
101	2	11	1	8	14	10	7	13	7	1	5	4
102	12	3	3	6	7	3	6	13	12	8	4	7
103	2	2	13	2	11	7	11	5	10	4	14	8
104	6	12	7	13	1	2	1	9	11	2	9	8
105	13	1	6	6	1	3	10	1	2	15	1	6
106	6	2	1	2	4	5	13	12	13	11	2	5
107	7	3	12	14	3	5	14	4	5	12	2	9
108	6	9	8	6	12	5	10	9	12	13	11	15
109	15	2	8	12	8	15	15	5	3	1	2	5
110	3	12	15	6	7	5	1	5	11	14	14	5
111	15	10	10	5	10	15	4	14	8	3	13	8
112	12	14	11	1	7	5	14	7	12	14	4	12
113	15	14	7	6	8	1	12	2	11	7	11	12
114	3	4	3	13	10	8	12	3	14	7	8	14
115	6	13	2	15	1	9	1	4	2	1	1	10
116	11	1	7	13	3	3	9	4	11	12	12	6
117	9	10	8	15	14	13	7	3	9	13	6	3
118	2	9	14	8	9	1	6	11	13	11	11	7
119	15	5	5	9	12	5	15	4	15	9	4	4
120	7	7	13	12	3	4	7	8	8	9	15	7
121	2	8	7	11	6	10	4	10	14	2	12	3
122	4	7	6	10	2	10	7	5	9	7	9	4
123	8	5	11	7	10	15	5	13	6	4	10	3
124	14	14	14	8	14	14	5	7	13	12	8	15
125	11	11	2	9	1	15	13	4	9	15	9	13
126	6	3	15	6	11	10	10	13	5	15	11	6
127	8	14	5	1	8	12	6	1	9	10	9	2
128	14	3	6	14	5	12	6	1	8	10	6	15
129	7	14	11	3	4	8	15	1	3	7	10	2
130	5	7	5	7	1	10	1	9	7	12	8	1
131	4	10	14	9	3	9	11	7	11	5	11	15
132	4	11	13	8	5	9	1	6	4	10	15	3
133	5	7	9	2	4	1	12	7	11	7	1	4
134	3	5	3	13	8	8	4	10	7	3	15	10
135	8	7	3	3	8	5	8	15	12	4	11	8
136	11	4	12	6	7	15	15	14	9	10	6	10
137	9	4	7	13	5	8	11	10	11	10	9	13
138	2	4	15	6	11	10	13	1	5	3	6	10
139	12	10	13	8	12	9	12	1	11	8	11	11
140	2	4	1	1	1	11	4	5	12	12	11	7
141	15	3	4	11	15	4	15	2	7	10	1	8
142	13	7	3	10	5	7	1	5	6	15	11	11
143	7	1	5	3	15	8	8	12	1	14	1	7
144	6	12	10	7	11	11	9	15	11	3	13	7
145	5	7	14	10	12	1	4	14	9	2	11	1
146	8	6	10	8	3	13	13	13	6	3	10	13
147	1	14	11	10	7	7	13	2	8	1	10	6
148	6	7	13	8	8	13	14	2	6	12	2	1
149	4	13	15	8	10	9	4	11	2	14	9	3
150	2	10	5	11	6	11	11	1	15	6	13	14
151	4	8	15	2	13	7	5	11	12	13	15	5
152	2	13	9	12	7	6	12	14	5	5	9	4
153	12	10	1	11	11	6	9	13	8	15	11	8
154	3	6	4	7	3	3	15	14	1	14	2	4
155	12	2	11	4	7	3	13	5	11	12	6	14
156	5	15	4	11	13	7	7	2	8	10	1	11
157	5	3	2	12	4	3	12	6	15	11	4	9
158	12	12	8	1	1	14	1	13	7	9	7	15
159	15	9	5	8	15	1	6	4	3	1	12	6

160	1	6	6	8	9	2	11	2	6	13	1	13
161	5	10	13	11	14	12	3	10	15	8	15	4
162	15	2	5	9	13	14	9	4	15	5	9	11
163	14	3	8	5	5	4	6	14	8	4	2	6
164	6	11	14	12	9	7	10	8	3	11	2	8
165	3	10	2	8	10	6	7	13	14	11	14	15
166	4	7	7	3	9	5	14	5	4	14	4	2
167	14	6	13	8	6	10	5	14	3	2	12	5
168	4	8	1	10	9	15	2	6	14	6	7	10
169	8	15	11	7	2	7	14	7	12	12	12	9
170	4	10	7	3	7	8	6	15	15	7	9	2
171	12	12	9	5	12	15	1	8	15	3	15	14
172	14	12	8	9	13	3	8	2	13	8	10	11
173	15	12	3	6	7	11	7	11	9	4	6	13
174	4	14	12	4	5	15	6	11	6	14	14	6
175	13	7	2	10	3	12	13	11	13	14	4	7
176	14	8	10	2	7	8	10	13	6	10	10	4
177	5	4	11	3	15	2	1	8	1	7	14	11
178	6	3	15	10	10	13	9	3	15	2	5	14
179	6	10	12	8	15	4	12	3	14	8	7	3
180	13	6	8	6	10	1	11	6	1	4	6	15
181	6	6	9	2	8	2	4	14	6	5	4	8
182	11	1	8	8	3	15	10	6	1	5	9	11
183	1	15	12	7	3	2	15	12	4	15	8	1
184	10	13	14	12	1	14	4	1	13	15	7	2
185	5	9	1	7	5	13	2	4	7	7	15	10
186	9	9	1	2	4	12	9	2	4	10	14	10
187	14	14	14	8	2	13	15	3	14	14	8	6
188	10	9	8	5	4	4	1	13	5	5	5	4
189	2	2	12	3	7	6	5	4	1	12	11	3
190	6	2	2	11	1	5	8	3	12	11	3	11
191	4	6	2	1	1	15	4	14	8	9	2	2
192	3	12	11	5	1	5	4	3	5	3	8	14
193	15	7	4	2	5	12	11	9	4	7	9	3
194	14	2	7	8	15	14	13	15	2	8	6	5
195	1	12	9	8	13	9	13	10	12	12	14	15
196	2	14	11	11	3	8	14	15	3	6	10	8
197	4	2	13	4	3	5	2	3	9	9	3	13
198	12	4	11	3	11	8	1	9	3	9	9	8
199	13	2	12	15	15	9	12	6	1	4	1	2
200	10	11	9	15	5	13	6	6	6	15	8	7
201	14	3	2	13	6	5	11	7	11	10	15	1
202	9	7	11	3	2	12	10	12	15	6	5	4
203	7	11	14	6	8	2	2	9	10	10	6	13
204	5	15	13	15	2	9	8	8	11	2	11	2
205	9	12	6	1	6	11	12	12	10	14	15	12
206	3	13	3	8	3	14	11	1	6	7	2	5
207	1	2	12	3	3	3	15	8	4	1	4	12
208	6	12	11	12	8	7	7	8	2	7	13	15
209	3	5	1	11	4	1	1	1	7	3	2	3
210	10	12	1	2	5	14	10	6	4	15	6	3
211	13	13	13	2	12	1	2	10	6	12	8	1
212	4	3	8	15	11	1	6	8	8	5	13	13
213	10	12	14	7	8	11	15	4	4	3	12	11
214	9	13	6	8	4	15	6	5	1	6	5	14
215	5	5	7	13	7	5	4	12	8	6	6	2
216	15	4	5	10	1	11	4	7	8	13	7	13
217	13	11	2	1	11	15	11	6	12	4	9	14
218	7	4	7	13	10	13	14	14	4	8	14	6
219	14	15	2	10	6	10	7	11	3	10	3	9
220	12	7	11	4	10	15	7	8	2	7	9	13
221	7	15	5	12	3	1	9	3	3	13	14	2
222	14	1	8	9	7	6	13	8	6	2	6	4
223	13	7	9	8	1	11	4	12	10	5	11	7
224	12	11	14	11	2	8	14	4	4	3	15	2
225	8	2	6	8	1	12	8	6	15	5	7	7
226	14	6	3	11	2	14	15	14	7	9	7	13
227	7	4	8	1	11	5	12	15	5	14	12	4
228	8	3	5	15	2	7	11	11	11	2	5	12
229	6	14	12	2	7	5	11	6	8	12	3	11
230	15	13	11	2	10	10	4	15	14	1	1	4
231	11	14	5	9	3	4	3	10	10	10	4	2

232	7	5	8	15	10	12	3	4	7	15	12	4
233	14	6	6	3	4	15	3	11	6	15	3	10
234	7	4	8	2	13	10	3	10	13	4	9	1
235	3	5	3	1	6	2	8	6	15	14	10	3
236	6	10	10	4	12	15	7	3	12	5	11	1
237	7	11	1	5	14	8	8	5	7	15	3	12
238	7	12	14	1	14	12	2	8	14	9	5	8
239	1	12	5	3	11	15	8	14	13	12	15	8
240	6	1	6	6	6	12	11	5	10	4	3	6
241	1	10	7	12	9	5	5	14	3	8	11	6
242	10	11	13	10	15	2	3	11	11	3	8	9
243	5	11	2	12	4	15	8	7	4	5	13	4
244	1	11	12	13	7	15	3	14	2	1	12	8
245	4	15	9	11	11	12	12	12	10	2	2	2
246	10	4	1	3	2	8	7	1	9	13	11	15
247	14	13	7	5	8	5	5	8	5	1	4	2
248	12	8	5	15	13	15	11	8	10	6	4	12
249	2	8	4	10	14	15	7	5	13	10	4	4
250	6	6	13	15	14	7	11	2	10	3	3	2
251	4	1	10	2	3	15	7	3	11	6	14	5
252	14	1	6	10	6	14	1	1	4	10	11	4
253	5	12	13	5	15	15	15	5	13	2	10	11
254	2	5	6	6	13	9	8	1	11	11	8	1
255	11	10	1	6	4	12	2	3	13	10	3	9
256	6	1	6	5	3	6	2	10	15	6	12	1
257	3	5	14	9	3	4	7	2	1	7	6	12
258	11	8	2	12	9	9	3	11	2	13	11	5
259	10	10	10	7	8	13	1	11	4	15	13	13
260	10	5	4	8	11	9	15	10	7	1	11	11
261	2	7	11	9	13	2	3	2	11	15	14	8
262	13	3	3	4	9	9	9	12	6	11	1	7
263	13	7	13	2	5	9	13	14	1	12	15	13
264	3	10	3	4	7	15	12	8	12	15	7	5
265	11	14	2	1	5	4	4	12	13	14	3	3
266	2	9	2	13	6	6	10	9	8	8	4	8
267	14	11	11	10	8	7	3	13	4	14	14	8
268	4	11	6	7	5	11	12	4	13	2	12	2
269	12	9	12	5	8	1	10	4	9	10	14	2
270	13	4	5	12	5	14	4	10	15	7	6	14

INSTANCIA MC_12_300

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

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

99	10	4	2	2	7	2	9	8	12	2	11	8
100	4	3	6	8	4	12	14	12	9	15	7	11
101	13	4	3	9	12	15	1	15	13	8	7	12
102	14	13	13	12	3	1	8	1	8	4	3	5
103	10	14	1	15	15	1	13	11	6	8	3	11
104	4	7	9	7	11	8	14	5	4	14	1	8
105	11	10	15	4	2	10	11	12	9	12	7	14
106	12	7	8	8	2	5	6	6	4	9	7	15
107	8	2	6	14	10	4	5	14	4	9	14	8
108	9	7	9	13	7	9	4	7	2	3	12	2
109	12	1	15	4	5	15	10	15	15	6	10	10
110	15	14	13	15	8	7	6	12	10	1	12	8
111	9	5	8	5	9	12	8	15	7	4	10	7
112	6	15	9	11	2	3	10	1	12	14	6	6
113	8	14	8	4	2	15	15	1	5	14	7	14
114	2	15	6	9	5	8	2	1	4	14	11	8
115	14	6	2	9	11	15	8	14	6	3	3	14
116	8	7	11	8	4	14	10	5	1	14	5	3
117	4	9	15	15	9	10	8	7	4	7	2	10
118	8	14	9	6	7	4	6	4	15	2	15	2
119	7	6	11	12	10	6	2	3	9	9	9	6
120	7	5	15	13	12	13	14	11	2	8	4	12
121	15	14	7	5	13	5	10	11	6	2	6	7
122	9	5	5	3	1	7	2	15	6	6	1	4
123	11	5	8	3	11	11	6	5	2	11	10	14
124	14	14	8	6	12	12	12	12	4	2	10	1
125	11	2	12	6	4	1	13	12	1	6	13	4
126	5	14	9	9	14	12	11	7	12	5	9	15
127	8	13	10	15	2	4	8	6	2	1	6	3
128	11	9	12	6	4	14	13	3	7	5	5	8
129	6	11	14	10	4	9	3	8	1	1	10	6
130	15	8	7	13	3	4	10	13	1	11	9	3
131	15	3	1	12	7	9	3	9	12	4	7	6
132	15	13	5	3	15	9	9	8	9	15	12	15
133	2	1	7	8	2	15	2	15	2	11	13	10
134	9	4	11	4	14	12	4	13	10	12	14	12
135	9	14	2	4	6	6	15	13	10	15	1	15
136	8	11	9	4	1	6	5	6	8	5	8	8
137	6	8	2	9	3	10	5	9	4	13	1	7
138	8	10	9	12	6	7	12	11	14	1	15	12
139	14	7	14	10	12	2	3	15	10	11	12	6
140	15	11	11	14	1	15	11	12	2	10	6	14
141	10	6	15	1	10	2	1	14	14	13	11	13
142	13	1	6	9	2	10	4	12	7	1	7	10
143	10	11	12	10	2	14	10	15	14	10	1	5
144	12	2	4	5	15	15	4	8	6	11	2	14
145	10	2	11	3	1	15	9	2	7	13	3	4
146	8	15	2	7	5	5	13	10	4	4	14	5
147	3	8	2	2	5	11	2	6	5	4	11	11
148	15	11	9	11	11	15	10	10	9	9	9	7
149	9	8	11	7	6	15	1	7	1	15	14	15
150	9	12	15	14	13	14	1	11	8	12	9	2
151	5	9	1	8	14	6	13	14	9	15	2	9
152	3	5	6	6	8	4	2	14	12	10	1	4
153	9	12	10	10	4	1	1	2	12	14	11	13
154	10	3	2	6	14	7	14	9	15	1	2	9
155	6	7	15	4	3	2	5	12	5	9	8	4
156	4	4	3	4	3	8	9	4	10	15	14	9
157	4	5	2	9	10	15	14	12	6	9	8	12
158	10	11	10	4	15	1	15	15	12	12	13	11
159	6	4	15	10	13	9	14	8	5	8	9	14
160	14	4	8	3	9	15	15	2	12	14	10	7
161	9	15	2	14	2	2	13	1	3	14	13	7
162	8	11	1	7	3	14	13	6	3	8	10	15
163	15	5	14	8	5	5	10	12	3	6	4	13
164	13	5	3	13	13	2	8	8	1	6	10	3
165	4	12	14	1	10	11	9	12	15	9	9	6
166	6	1	11	10	10	11	8	6	2	15	2	1
167	12	4	15	12	4	13	2	13	8	3	11	9
168	8	11	1	2	13	1	10	7	1	13	6	6
169	13	9	13	10	13	12	2	15	9	7	4	7
170	5	13	13	11	4	7	5	1	2	1	13	1

171	12	13	12	4	15	9	12	11	7	10	12	11
172	13	1	8	2	2	6	4	11	3	5	13	4
173	14	12	11	1	1	11	6	9	4	9	11	1
174	12	10	12	13	1	3	6	9	5	2	13	6
175	15	13	5	5	7	1	6	10	10	11	15	15
176	6	6	4	10	11	13	11	14	15	8	2	7
177	2	12	3	11	13	7	15	5	1	9	3	6
178	4	3	1	14	13	2	2	14	15	13	8	9
179	2	14	9	11	7	4	3	5	9	3	3	1
180	15	8	15	14	2	10	3	8	7	5	4	8
181	13	2	12	2	12	12	12	13	9	2	7	3
182	15	7	12	5	9	1	1	14	1	6	3	4
183	4	5	14	4	1	9	7	13	14	13	15	7
184	13	10	2	9	10	13	4	10	13	5	6	2
185	6	4	14	10	13	13	11	7	3	7	6	2
186	4	12	10	14	8	15	14	4	6	13	11	10
187	8	10	11	13	6	3	11	1	14	9	12	9
188	10	8	14	10	11	7	10	9	10	3	7	6
189	4	5	11	12	13	1	10	2	7	15	2	7
190	13	14	5	4	4	7	6	6	5	12	14	15
191	13	3	6	14	11	3	12	3	6	11	6	11
192	2	3	11	6	4	10	12	9	13	7	7	7
193	1	8	6	4	15	15	5	11	2	13	15	10
194	2	5	4	15	5	3	4	4	7	9	6	13
195	1	3	2	5	5	12	11	9	10	13	5	12
196	5	11	2	9	5	7	4	1	13	2	12	1
197	2	11	5	15	15	15	8	10	1	1	8	4
198	11	9	13	13	10	2	14	10	11	14	12	12
199	12	12	11	4	14	12	10	6	12	7	6	15
200	5	11	10	11	2	14	12	11	11	7	8	3
201	6	1	7	13	10	6	9	6	10	2	7	10
202	2	15	12	5	6	8	9	4	13	10	11	10
203	3	15	13	13	9	11	15	3	7	10	1	7
204	4	3	15	14	7	12	11	6	11	7	14	11
205	11	10	11	12	7	7	11	11	6	1	1	8
206	5	3	7	15	8	10	7	4	5	9	11	15
207	10	4	10	15	6	3	1	3	14	4	15	8
208	13	1	9	15	13	5	14	15	13	12	5	3
209	12	3	7	4	5	2	12	2	8	9	6	9
210	1	1	14	9	3	11	2	12	2	10	7	2
211	10	8	6	2	8	7	9	14	4	12	12	6
212	3	9	14	9	6	4	3	11	3	10	9	8
213	7	1	13	14	12	1	4	8	4	14	12	9
214	10	5	4	3	14	10	10	5	10	3	4	13
215	3	10	11	14	14	9	13	10	3	12	8	4
216	2	7	10	11	5	4	3	4	10	8	1	14
217	9	14	13	13	3	13	8	3	8	5	11	1
218	3	15	11	15	10	3	6	8	3	6	1	2
219	2	1	2	3	5	12	12	8	9	1	4	11
220	9	12	5	1	13	11	8	8	15	3	8	9
221	7	5	4	14	7	11	11	12	12	4	14	15
222	11	7	13	8	13	11	3	8	2	13	7	7
223	12	3	11	2	4	15	14	14	11	2	15	1
224	8	1	8	4	4	8	5	12	7	12	1	9
225	8	11	6	8	15	6	10	12	10	12	10	12
226	4	12	11	8	14	14	13	8	5	8	12	2
227	5	5	10	1	15	7	10	9	8	12	5	4
228	8	3	9	1	14	1	10	10	6	3	1	9
229	12	11	5	13	4	5	15	15	13	10	10	12
230	7	12	4	2	8	4	4	2	6	13	15	6
231	9	10	1	8	3	2	12	2	10	15	11	10
232	1	3	6	8	14	9	4	6	5	8	1	2
233	5	11	10	11	11	7	5	14	6	4	9	6
234	13	5	10	11	11	6	14	12	13	2	7	14
235	9	10	8	5	8	5	9	7	6	1	5	7
236	5	9	2	9	3	4	3	15	1	10	7	5
237	6	10	13	10	1	14	9	5	11	4	11	13
238	2	14	3	15	12	7	4	4	3	8	14	13
239	1	7	3	14	13	11	14	1	8	8	13	11
240	5	3	5	12	4	15	4	13	14	12	6	8
241	5	10	6	6	6	2	14	10	8	13	13	1
242	11	7	15	10	8	13	15	11	2	14	7	5

243	5	7	13	7	5	11	1	12	11	13	9	10
244	11	10	15	13	7	14	10	7	15	7	7	15
245	9	2	9	14	15	15	2	12	8	13	3	6
246	5	5	11	8	9	13	6	5	15	10	13	12
247	10	15	15	1	7	2	11	13	11	4	11	2
248	4	14	2	4	7	6	14	5	11	10	5	13
249	1	10	3	2	13	2	2	2	5	7	12	11
250	8	11	1	2	7	6	13	14	8	12	2	15
251	12	2	10	13	10	12	15	7	7	12	8	10
252	5	12	15	3	9	4	7	4	8	13	14	6
253	9	3	14	14	15	12	9	14	7	11	1	8
254	10	10	1	4	2	4	6	8	12	11	7	2
255	5	11	7	14	2	5	11	6	14	6	14	4
256	10	7	8	3	1	15	14	13	14	14	4	6
257	4	5	13	13	4	13	13	9	6	12	11	8
258	4	7	5	10	9	5	1	8	7	15	14	11
259	3	4	15	5	1	1	14	12	15	2	14	2
260	10	14	11	9	10	14	4	2	15	12	11	13
261	9	14	2	9	9	14	15	15	11	13	6	5
262	1	8	13	14	3	14	2	11	14	13	6	5
263	9	12	8	9	1	4	6	7	6	2	14	8
264	5	15	3	12	2	15	7	3	5	6	14	2
265	4	5	8	3	11	10	13	9	13	7	2	4
266	11	7	10	8	13	7	6	9	10	15	3	10
267	14	8	6	5	10	11	6	3	15	1	11	13
268	11	1	9	7	5	12	8	5	1	2	3	12
269	1	10	9	11	1	5	8	10	14	11	1	14
270	4	13	12	14	1	14	6	7	15	1	3	12
271	4	14	11	4	3	15	12	14	3	6	4	10
272	5	6	15	3	8	5	10	4	9	15	4	11
273	14	12	8	14	13	15	15	7	14	15	13	7
274	12	12	10	8	10	13	7	11	5	13	7	6
275	10	9	2	15	12	15	6	7	2	9	5	6
276	3	3	3	10	14	11	14	7	5	9	2	15
277	7	8	4	1	9	11	4	15	8	14	11	12
278	8	13	7	1	14	6	15	8	10	8	10	12
279	3	6	9	15	13	11	14	1	3	13	14	13
280	10	11	15	14	7	13	12	10	11	8	9	15
281	7	5	1	7	14	7	2	7	2	13	13	6
282	8	5	15	7	3	13	1	12	7	2	15	7
283	6	4	10	10	5	11	8	12	7	8	4	6
284	13	10	7	14	7	3	7	8	11	9	15	3
285	14	2	6	5	11	12	4	7	6	1	13	13
286	8	8	4	15	13	4	12	4	7	2	7	11
287	13	5	15	8	5	9	14	9	5	13	1	15
288	4	2	3	15	3	2	1	2	1	11	13	9
289	1	15	6	10	6	2	1	15	6	6	9	8
290	8	12	7	15	11	11	5	13	9	2	13	14
291	3	7	6	7	15	3	4	14	11	15	4	14
292	5	15	8	2	7	3	13	4	1	7	10	1
293	12	10	7	11	1	13	3	3	13	5	2	8
294	11	4	13	7	11	14	10	15	4	8	5	11
295	14	13	2	4	5	9	11	9	11	9	10	1
296	12	1	12	15	4	4	11	12	8	8	13	5
297	15	2	13	11	12	12	4	10	7	13	14	15
298	12	3	4	4	9	7	2	3	5	14	1	11
299	12	1	3	8	12	15	3	7	1	8	7	15
300	15	10	6	13	4	14	10	15	10	14	7	10

Glosario de Términos

Algoritmo Colonia de Hormigas: Algoritmo bioinspirado en el comportamiento de las hormigas, que puede ser aplicado a problemas de optimización, basándose en el comportamiento estructurado y comunicación indirecta llevada a cabo por estos animales.

Algoritmo Determinístico: Es un algoritmo en el que una misma entrada obtiene siempre el mismo resultado.

Algoritmo no Determinístico: Algoritmo en el que a una misma entrada se obtienen diferentes salidas, de modo que no es posible saber de manera previa, el resultado que de la ejecución de un algoritmo de este tipo.

Algoritmo Secuencial: Algoritmo programado de manera estructurada que al ser ejecutado utiliza un solo procesador.

Análisis de Sensibilidad: Evaluación del comportamiento de las variables críticas de un problema con la finalidad de establecer un rango numérico, dentro del cuál, la solución obtenida por el algoritmo sigue siendo buena, además de que permite conocer que tan sensible es el algoritmo a cambios en los valores de ciertas variables propias del problema.

Benchmark: Palabra del inglés utilizada para nombrar a los problemas de prueba utilizados por diversos investigadores, para medir el rendimiento de un sistema o algoritmo para un problema dado.

Para el caso del Problema de Máquinas en Paralelo no Relacionadas existen en la literatura un conjunto de benchmarks utilizados por diversos autores, los cuales involucran problemas de prueba de 20, 40 , 60, 80, 100 y 120 trabajos, a calendarizar en 2, 4, 6, 8, 10 y 12 máquinas respectivamente.

Búsqueda Local: Técnica iterativa de que permite explorar el espacio de soluciones de un problema dado, a partir de una solución inicial, por medio de movimientos, de modo que vaya mejorando la solución obtenida de acuerdo a la función objetivo. Este tipo de técnicas son utilizadas para mejorar la calidad de las soluciones obtenidas por un algoritmo, así como para reducir el tiempo en que se obtienen dichas soluciones.

Búsqueda Tabú (TS): Técnica basada en la inteligencia artificial, empleando el concepto de memoria e implementándolo mediante estructuras simples, con el objetivo de dirigir la búsqueda de la solución final en función de los resultados alcanzados [Glover, 1989]. TS considera dos tipos de memoria que interaccionan entre sí, a corto y largo plazo respectivamente. Este tipo de datos darán lugar a estrategias de intensificación y/o diversificación dentro del ámbito local o global.

Complejidad Espacial: Es la cantidad de memoria requerida por el algoritmo durante la ejecución.

Complejidad Temporal: Es el número de pasos necesarios (tiempo) para obtener una solución a una instancia de un problema dado.

Costo: En el caso del problema de UPMP, son las unidades de tiempo requeridas para el procesamiento de uno o varios trabajos.

Estructura de Vecindad: Tipo de movimiento utilizado para explorar el espacio de soluciones de un problema dado.

GRASP: (Greedy Randomized Adaptative Search Procedures). Es una procedimiento iterativo que consiste en una fase de construcción y una de mejora [Feo y Resende, 1989]. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial, considerando una lista restringida de los mejores candidatos y seleccionando un elemento de dicha lista, de forma aleatoria. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local.

Heurística: Procedimiento intuitivo bien definido que proporciona buenas soluciones aproximadas a problemas difíciles de resolver sin garantizar la optimalidad, en un tiempo computacional razonable.

Lista Tabú: Lista utilizada en el algoritmo Colonia de Hormigas, la cuál contiene los nodos (dependiendo del problema, en el caso de UPMP, son los trabajos) visitados por cada hormiga, misma que tiene la función de evitar que un nodo sea visitado más de una vez. Ya que la lista tabú se ha completado, ésta contendrá la solución construida por la hormiga a un problema dado.

Metaheurísticas: Métodos aproximados que mejoran procedimientos heurísticos, los cuales son diseñados para ser aplicados a problemas considerados difíciles de resolver, donde las heurísticas no son eficientes.

Modelo de Programación entera binaria: El formato del modelo de programación entera binaria requiere tener una función objetivo lineal, ya sea a minimizar o maximizar. La característica de este modelo, es que su conjunto de variables solo puede tomar valores binarios, es decir, 0s y 1s.

Modelo de Grafos disyuntivos: Modelo que permite representar problemas de decisión mediante el uso de nodos y arcos. Donde los arcos conjuntivos representar el orden lógico de procesamiento de las operación, y los arcos disyuntivos representan el orden de procesamiento de las operaciones asignadas a cada máquina.

Óptimo Global: Es la mejor solución de un espacio de soluciones f(x).

Óptimo Local: Representa la mejor solución de f(x) en un entorno x

Preemptions: Interrupción en el procesamiento de un trabajo, debido a una mayor prioridad para ejecutar otro trabajo.

Recocido Simulado (SA): Algoritmo que realiza una búsqueda aleatoria orientada, el cuál hace una analogía del proceso simulado de fundición de metal, de modo que la temperatura va descendiendo gradualmente hasta que su energía mínima se alcanza. De modo que permite escapar de óptimos locales al aceptar algunas soluciones consideradas malas, obteniendo muy buenos resultados en diversos problemas de optimización a los que se ha aplicado.

Setup times: Tiempos de preparación requeridos por algunas máquinas para iniciar o intercambiar una operación, o bien, al finalizar un proceso.

Sintonización: Es la proporción adecuada en cuanto a los valores obtenidos mediante el análisis de sensibilidad aplicado a los parámetros de control, tomando en cuenta el problema y el método de optimización utilizado, de modo que el algoritmo muestre una mejora tanto en eficiencia como en eficacia.

Tiempo polinomial: En las ciencias computacionales, el tiempo viene expresado generalmente en función del tamaño de la entrada. Se considera que un algoritmo puede ser resuelto en tiempo polinomial sí su función de complejidad es de orden *O(p(n))*, es decir, que puede ser representado por un polinomio.

Vecindad: Es el conjunto de soluciones que pueden ser alcanzadas desde una solución dada por medio de un movimiento (inserción, eliminación, permutación).