UNIVERSIDAD AUTÓNOMA
DEL ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
CENTRO DE INVESTIGACIONES EN INGENIERÍA
Y CIENCIAS APLICADAS

ALGORITMO DE RECOCIDO SIMULADO PARALELIZADO APLICADO AL PROBLEMA DE ASIGNACION DE RECURSOS EN UN TALLER DE MANUFACTURA FLEXIBLE SUJETO A DISPOSICIONES DE TIEMPO

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE

DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

P R E S E N T A
M.C. MARTIN GERARDO MARTINEZ RANGEL

ASESOR INTERNO: Dr. MARCO ANTONIO CRUZ CHAVEZ
ASESOR EXTERNO: Dr. JOSE CRISPIN ZAVALA DÍAZ

CUERNAVACA MOR. DICIEMBRE 2008

UNIVERSIDAD AUTÓNOMA
DEL ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
CENTRO DE INVESTIGACIONES EN INGENIERÍA
Y CIENCIAS APLICADAS

"ALGORITMO DE RECOCIDO SIMULADO PARALELIZADO, APLICADO AL PROBLEMA DE ASIGNACIÓN DE RECURSOS EN UN TALLER DE MANUFACTURA FLEXIBLE SUJETO A DISPOSICIONES DE TIEMPO"

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE:

DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

P R E S E N T A

M.C. MARTÍN GERARDO MARTÍNEZ RANGEL

ASESOR INTERNO: Dr. MARCO ANTONIO CRUZ CHÁVEZ
ASESOR EXTERNO: Dr. JOSÉ CRISPÍN ZAVALA DÍAZ

CUERNAVACA MOR. DICIEMBRE 2008

RESUMEN

Este trabajo de tesis propone un algoritmo de recocido simulado basado en un modelo matemático que permite dar un acercamiento para resolver el problema de asignación de tareas en talleres de manufactura flexible sujeto a disposiciones de tiempo, el cual es una extensión del problema clásico de JSSP (por sus siglas en inglés, job shop scheduling problema). El algoritmo propuesto se presenta en dos versiones: secuencial y paralelo. El estudio de este tipo de problemas resulta de interés dada su relación con la administración de recursos en la industria.

Este trabajo de investigación presenta un panorama general acerca del problema y de los diferentes enfoques y estrategias combinatorias utilizadas para resolver el problema mediante el estudio del estado del arte de la literatura existente. Se logran plantear de manera general los métodos más utilizados, las funciones de vecindad propuestas para tal fin, así como las estrategias de evaluación de los movimientos que realizan a fin de mejorar las soluciones encontradas.

El algoritmo de recocido simulado implementa una estrategia eficaz y eficiente de calendarización parcial que permite explorar de manera más rápida un mayor espacio de soluciones para problemas que operan en ambientes de manufactura flexible. Recocido simulado también incluye un mecanismo que permite sintonizar su secuencia de enfriamiento. Para probar esta metaheurística se utilizaron instancias de prueba propuestas en la literatura y los resultados obtenidos fueron contrastados bajo dos ambientes de programación, secuencial y paralelo. Los resultados encontrados demuestran que el algoritmo propuesto resulta eficaz, eficiente y capaz de competir con los resultados de otros autores.

ABSTRACT

This thesis work proposes an algorithm of simulated annealing based on a mathematical model that allows to give an approach to solve the flexible job shop scheduling problem with setup time. The flexible job shop scheduling problem is an extension of the classical job shop scheduling problem. The proposed algorithm appears in two versions: sequential and parallel. The study of this kind of problems is of interest when they have relation with the resource management in the industry.

This research work presents a general overview about the problem and of the different approaches and used strategies to solve the problem by means of the study of the a state of the art, and they are managed to raise of general way the more used methods, the proposed neighbourhoods functions such aim and its strategies of evaluation of the movements that make in order to improve the found solutions. The proposed algorithm implements a strategy of partial rescheduling along with an effective and efficient strategy to explore of faster way a greater space of solutions for problems that operate in environments of flexible manufacture. This integral strategy to simulated annealing which also includes a mechanism that allows to tune its sequence of cooling. In order to prove this metaheurístic propose, benchmarks were used and the obtained results were resisted under two programming environment, sequential and parallel. The found results demonstrate that the proposed algorithm is effective and efficient and competes with the results of other authors.

AGRADECIMIENTOS

Al Dr. Marco A. Cruz Chávez, amigo y Director de este trabajo de investigación. Gracias por la orientación durante el tiempo de mis estudios y por todo el apoyo brindado para la realización y culminación de esta tesis doctoral.

Al Dr. J. Crispín Zavala Díaz, amigo y Co-Director de este trabajo de investigación. Gracias también por el apoyo prestado para la culminación de este proyecto.

A los Drs. David Juárez Romero, Juan Frausto Solís, J. Alfredo Hernández Pérez, Margarita Tecpoyotl Torres, Álvaro Zamudio Lara, a todos gracias por sus acertadas observaciones durante las evaluaciones de Comité Tutoral y por su apoyo para mejorar la presente tesis con sus valiosas y pertinentes observaciones y echas al documento final. Nuevamente gracias.

A los Drs. María Camino Rodríguez Vela y Ramiro Varela Arias del Centro de Inteligéncia Artificial de la Universidad de Oviedo, Asturias, España. Gracias por el apoyo recibido durante mi estancia en ese Centro.

Gracias a la Universidad Autónoma del Estado de Morelos por el apoyo otorgado durante todo el tiempo que duró este proyecto.

A los Directivos del Centro de Ingeniería y Ciencias Aplicadas, Facultad de Ciencias Químicas e Ingenierías, Facultad de Contaduría, Administración e Informática: Dr. Gustavo Urquiza Beltrán, MC. José Antonio Valerio Carvajal, L.A.P. Carlo Pastrana Gómez, respectivamente. Gracias por el apoyo brindado durante todo este tiempo.

Al Dr. Pedro A. Márquez Aguilar, Secretario de Investigación y Posgrado del Centro de Ingeniería y Ciencias Aplicadas, y a su personal. Gracias por todo el apoyo brindado durante mi estancia en este centro.

Al Consejo Nacional de Ciencia y Tecnología por la beca otorgada.

A la Fundación Telmex, por la beca otorgada.

DEDICATORIAS

Primeramente a Dios, quién no se cansa de darme oportunidades para cumplir su misión en este mundo.

A mi madre: María Luisa, ejemplo de tesón e ingenio en tiempos difíciles, gracias.

A mi padre Gabriel^{+}, quién sin quererlo señaló el camino que debería seguir mi vida.

A mi abuelo Ángel^{+}, hombre de carácter fuerte, quién me enseñó a una corta edad el valor del trabajo y del esfuerzo como único medio para triunfar en la vida.

A mi esposa Tere, por su paciencia y aguante durante estos años de matrimonio, y por ayudarme a consolidar un patrimonio para bien de nuestras hijas.

A Diana Sofía, la hija mayor y mi primer motor en los inicios de mi vida profesional, niña inteligente, hoy una adolescente con un prometedor futuro basado en el estudio y esfuerzo.

Alexa Mariana, la hija menor, niña inteligente, ejemplo de dulzura y amor, mi segundo motor a medio camino.

A mis hermanos, Juan Gabriel, María Aurora, Miguel Ángel, Pedro Antonio, José Enrique, María Luisa, Carlos Augusto, Rosario Gabriela: gracias por su cariño y amor.

En un apartado especial a todas aquellas personas que de manera desinteresada me han ayudado en los momentos difíciles , a quienes me han amado, a quienes me han enseñado, a todos gracias.

Y a todos los soñadores de este mundo que con coraje se arriesgan a vivir sus sueños.

"Algoritmo de Recocido Simulado Paralelizado, Aplicado al Problema de Asignación de Recursos en un Taller de Manufactura Flexible Sujeto a Disposiciones de Tiempo"

Contenido

Índice de Figuras

Índice de tablas

Capítulo 1

1 Introducción

El problema de asignación de tareas en un taller de manufactura JSSP (por sus siglas en inglés, Job Shop Scheduling Problem) es uno de los problemas de calendarización más conocidos y difíciles de resolver. JSSP es probablemente el modelo más estudiado y desarrollado de todos los problemas pertenecientes al problema general de calendarización [Pinedo, 2001]. Esto sirve como una referencia para otras técnicas que tratan de resolver problemas en el mismo campo, por ejemplo el problema del transporte y el problema de la mochila [Garey *y Johson,* 1976].

Para instancias grandes no existe un algoritmo determinístico que pueda resolver los problemas de calendarización. Por esta razón, se utilizan metaheurísticas para la búsqueda del óptimo global [Applegate *et al.,* 1991], a fin de generar algoritmos que puedan dar buenas soluciones en tiempo polinomial. En muchos ambientes de manufactura que tienen que ver con el control de la producción y/o planeación de la producción, el problema de calendarizar el trabajo en un taller de manufactura flexible FJSSP (por sus siglas en inglés, Flexible Job Shop Scheduling Problem) reviste gran importancia dentro de los procesos de fabricación. Hoy en día las máquinas utilizadas son multipropósito, y en consecuencia, pueden realizar múltiples operaciones, permitiendo que los procesos se flexibilicen.

El FJSSP es una generalización del JSSP clásico y en consecuencia, al ser una extensión de éste, también es NP-duro ya que incorpora todas las dificultades y complejidades de su antecesor JSSP [Garey *y Johson*, 1976; Applegate *y Cook*, 1991; Laarhoven *et al.,* 1992; Amico, 1993; Nowicky *y Smutniki*, 1996; González *et al.*, 2005; Burke *et*

al., 1997; Mastrolilli *et al.*, 1998; Ho *et al.*, 2004; Kacem *et al.*, 2002], tanto JSSP como FJSSP, por su naturaleza misma, son uno de los problemas más difíciles de resolver dentro del grupo de problemas encontrados en el área de optimización discreta [Jain, *et al.*, 1998; Mastrolilli *et al.*, 1998; Kacem *et al.*, 2002; Ho y Tay, 2004; Xia *y Wu*, 2005]. El FJSSP es abordado a través de diversos métodos, a fin de dar un acercamiento para resolverlo [Mastrolilli *et al.*, 1998; Kacem *et al.*, 2002; Ho *et al.*, 2004; Kacem, 2003; Zribi *et al.*, 2004; Ho y Tay, 2004; Xia *y Wu.*, 2005]. El FJSSP puede ser agrupado de acuerdo a las características de los problemas. Un grupo integra a los problemas con flexibilidad total, en el cual, cualquier operación puede disponer del total de máquinas que forman parte del proceso y debe seleccionarse una de ellas que la pueda procesar. Por otra parte está el grupo de problemas de flexibilidad parcial, donde no todas las máquinas están disponibles para todas las operaciones, este último tipo de problemas es el más duro de tratar dentro de los FJSSP [Kacem *et al.*, 2002; Ho *y Tay.*, 2004; Kacem 2003; Zribi *et al.*, 2004; Hong *et al.*,2001, Rossi y Dini, 2001].

En este trabajo de investigación se abordan problemas del tipo parcial FJSSP por la gran similitud que existe en ambientes de sistemas de manufactura reales conocidos como Sistemas de Manufactura Flexible (SMF), y se le adiciona el manejo de disposiciones de tiempo para acercarlo más aún a estos tipos de ambientes. Un SMF incorpora el proceso distribuido de información y el flujo automatizado de materiales a través de máquinaria controlada por computadora, celdas de ensamble, robots industriales, máquinas de inspección, sistemas de manejo y almacenamiento de materiales e integrados por sistemas inteligentes [Ranky, 1983; Klahorst, 1981]. Al incrementarse el volumen de producción se recomienda un SMF, en este sistema también se realizan varias operaciones sobre el material antes de salir del mismo. Sin embargo, aquí no todas las operaciones se realizan en un solo centro de máquinado, por lo que se requiere que la pieza o material se transfiera en forma automática de un centro de máquinado a otro.

El sistema puede contener máquinas que automáticamente realizan las operaciones requeridas de acuerdo a un conjunto detallado de instrucciones codificadas para la máquina y su operación en lotes proporciona flexibilidad en volumen y variedad, precisión y consistencia, estas máquinas son conocidas como máquinas de control numérico computarizado (CNC). Una máquina CNC normalmente se diseña alrededor de una familia de productos o partes, cuyo volumen justifica la inversión. En un sistema de producción, es posible conformar centros de máquinado compuestos por un conjunto de máquinas CNC y por máquinas CNC individuales para producir volúmenes mayores de artículos. El problema de un SMF o FJSSP, tiene las mismas restricciones que un JSSP, pero además se le agregan otras restricciones como son múltiples periodos de tiempo (disposiciones de tiempo) para cada una de las máquinas. Estos periodos de tiempo pueden corresponder a los tiempos requeridos para la preparación de cada una de las máquina a fin de que puedan recibir al inicio del proceso una primer operación. Pero también, estos tiempos pueden corresponder al tiempo necesario para preparar a una máquina para que pueda iniciar otra operación, terminada una anterior en un mismo trabajo. Y por último, también es posible agregar en todos los casos tiempos que corresponden a los tiempos requeridos para la limpieza de las máquinas al finalizar los trabajos y dejarlas listas para el siguiente proceso. En cuanto a los tiempos de procesamiento para cada una de las operaciones en un trabajo, estos tiempos están caracterizados por un tiempo de inicio y un tiempo final más el tiempo correspondiente a disposiciones de tiempo. Por lo anterior, para cada Operación O_{kL} de un trabajo K en una máquina L, es dado un intervalo [Tinferior, Tsuperior]. Una vez que la operación O_{kL} es finalizada, el límite inferior de este intervalo es el mínimo (respecto del máximo) tiempo que la operación $O_{k,L+1}$ debe esperar antes de comenzar.

1.1 Contribución de la tesis

Este trabajo de tesis propone un algorimo tipo recocido simulado eficiente y eficaz, al cual se le integra un mecanismo de calendarización denominado Parcial-S para FJSSP derivado del mecanismo propuesto para JSSP que permite evaluar más rápidamente el espacio de soluciones en el proceso de búsqueda local mediante la aplicación de una función de vecindad [Cruz-Chávez *et al.*, 2007]. Una función de vecindad determina el conjunto de las soluciones de vecindad que se pueden alcanzar a partir de la solución actual realizando un movimiento o una transición. La función de vecindad utilizada en esta investigación es N1 [Laarhove *et al.*, 1992]. Esta función de vecindad es utilizada en el Mecanismo de Generación de Vecindad denominada NGM (por sus siglas en inglés, Neighborhod Generator Mechanism) propuesto en [Cruz-Chávez *et al.*, 2006].

La combinación de N1 junto con NGM es muy simple, selecciona sólo soluciones factibles sin necesidad de calcular la ruta crítica de la solución generada como lo hace normalmente N1. Esto permite tener un alto rendimiento del algoritmo de calendarización propuesto en esta investigación doctoral, al generar solamente soluciones factibles al permutar operaciones sin tiempo de ocio en una máquina en particular. Uno de los principales problemas que se tienen al utilizar el algoritmo de recocido simulao es la sintonización de sus parámetros de control. En este trabajo de investigación se hace una propuesta que permite la sintonización de uno de sus principales parámetros de control que es la temperatura, la cual fue aplicada con éxito para JSSP en [Martínez-Rangel *et al.*, 2007] mediante la sintonización de la temperatura a través de la desviación estándar obtenida al generar una población de 65,000 soluciones para cada uno de las instancias utilizadas como prueba para problemas de FJSSP. Al sintonizar la temperatura de SA con la desviación estándar es posible converger a mejores soluciones en un menor tiempo. Aplicando el algoritmo propuesto se aborda el problema

de FJSSP que involucra disposiciones de tiempo, y se demuestra que este algoritmo es eficiente y eficaz tanto para FJSSP con disposiciones de tiempo como sin disposiciones de tiempo.

1.2 Trabajos relacionados

Las metaheurísticas utilizadas para problemas de calendarización son caracterizadas por búsquedas a través de vecindades. Por esta razón, el desarrollo de mecanismos más eficientes y efectivos que aceleren las búsquedas es importante para mejorar estas metaheurísticas. Un gran número de metaheurísticas han sido propuestas para encontrar buenas soluciones para el FJSSP en tiempo polinomial. En la literatura se encuentran diversos enfoques que han sido propuestos y que permiten encontrar buenas soluciones para el FJSSP. Mastrolilli y sus colaboradores [Mastrolilli *et al.*, 1998] proponen algunas funciones de vecindad que pueden ser utilizadas en metaheurística para resolver el FJSSP. Un estudio basado en el modelado de un algoritmo genético para resolver el problema se encuentra en [Kacem *et al.* 2002a] y los mismos autores [kacem *et al.*, 2002b] proponen un acercamiento basado en una hibridación de un algoritmo genético y lógica difusa. Ong y sus colaboradores aplican el principio de selección clonal del sistema inmune humano para resolver el FJSSP con re circulación [Ong *et al.*, 2006]. Mientras que Ho y otros [Ho *et al.,* 2004] proponen un algoritmo Genético eficiente cultural para resolver el FJSP mediante el uso de una metodología denominada GENACE. Todos estos enfoques requieren evaluar la calidad de soluciones para encontrar el makespan (valor objetivo de la función del problema) durante cada paso del algoritmo de la metaheurística. Para lograr esto, el algoritmo de calendarización generalmente utilizado es el propuesto por Zalzala y Fleming [Zalzala y Fleming, 1997]. Cada vez que la metaheurística obtiene una nueva solución (calendarización), este algoritmo es aplicado para la solución a fin de asignar un tiempo de inicio para cada una de las operaciones que son parte del FJSSP y obtener el valor de makespan. El makespan es

definido como el tiempo en que la última operación O_i es completada. Para el problema de FJSSP con disposiciones de tiempo Allahverdi y sus colaboradores [Allahverdi *et al.,* 1999] presentan una revisión exhaustiva de los problemas de asignación de tareas en talleres de manufactura que involucran disposiciones de tiempo. Se encontró que la mayoría de las investigaciones que abordan este tipo de problemas asumen que la disposición de tiempo es insignificante o que forma parte del tiempo de procesamiento de cada uno de los trabajos a ser procesados. Existen otros autores que describen el uso de tiempo de intercambio entre operaciones (disposiciones de tiempo) aplicado al JSSP clásico [Choi *et al.,* 1997; Cheung *et al.,* 2001; Choi *et al.,* 2002; Souissi *et al.*, 2004, Kim y Bobrowski, 1992], y en [Xia, *et al.* 2005; Freitas y Aparecida., 2006] se reconoce que para FJSSP con disposiciones de tiempo son muy escasos los trabajos de investigación que abordan este tipo de problemas.

1.3 Objetivos de la investigación

1. Desarrollo de un algoritmo secuencial y paralelo que aporte conocimiento en el área de optimización combinatoria para el problema de calendarizar tareas en un taller de manufactura flexible.
2. Aplicar el algoritmo a instancias de prueba existenten en la literatura para el problema de FJSSP con disposiciones de tiempo y sin disposiciones de tiempo. Ademas, mediante un conjunto de estudios experimentales demostrar la eficiencia y eficacia del algoritmo propuesto mediante la comparación de su rendimiento con otros algoritmos ya reportados en la literatura aplicados al mismo grupo de instancias de prueba seleccionadas.

1.4 Alcance de la investigación

1. El algoritmo de recocido simulado propuesto estará integrado por dos componentes, el primero un mecanismo que permite acelerar la búsqueda en el espacio de soluciones mediante una

calendarización parcial y el segundo componente es utilizar una estrategia para sintonizar los parámetros de control que permitan acelerar la convergencia de la metaheurística utilizada.

2. El algoritmo propuesto de recocido simulado, se ejecutará de forma secuencial y en forma paralela.
3. El grupo de instancias de prueba que se utilizarán serán las propuestas en [Brandimarte, 1993].
4. Para todos los casos, se utilizarán las mismas instancias de prueba y con los resultados obtenidos se hará un comparativo con los resultados ya reportados en la literatura existente.

1.5 Organización de la tesis

Después del capítulo introductorio; en el capítulo 2 se describe el problema de asignación de tareas en un taller de manufactura flexible en la sección 2.1, en la sección 2.2 se describe de manera conceptual el problema de FJSSP, en la sección 2.3, se explica a detalle el modelo de grafos disyuntivo para entornos de manufactura flexible, en la sección 2.4 se describe el modelo matemático de formulación disyuntiva partiendo del modelo de grafos disyuntivo aplicado a FJSSP. Por último en la sección 2.5 se describen los tipos de problemas de FJSSP.

En el capítulo 3, se da una introducción a los métodos aplicados a FJSSP, en la sección 3.1 se explican dos de los principales métodos enumerativos y en la sección 3.2 se explican los principales métodos de aproximación. En el mismo capítulo, en la sección 3.3 se describen las principales funciones de vecindad propuestas en la literatura y las estrategias de evaluación de los movimientos que realizan a fin de mejorar las soluciones encontradas.

El capítulo 4 describe a detalle el algoritmo de recocido simulado (SA) propuesto, en la sección 4.1, se presenta una introducción al

algoritmo de SA, en la sección 4.2 se presenta el esquema generalizado del mismo algoritmo y en la sección 4.3 se presenta una propuesta para sintonizar los parámetros de control de SA mediante un análisis de la convergencia de la metaheurística de SA sintonizando el parámetro de control de temperatura mediante el uso de una medida de dispersión, y en la sección 4.4, se describe el mecanismo de calendarización que forma parte del algoritmo de SA propuesto y en la sección 4.5 se presenta un análisis de la complejidad del mismo.

El capítulo 5 describe la paralelización del algoritmo de recocido simulado operando con algoritmo propuesto para calendarización parcial. En la sección 5.1 se da una introducción, en la sección 5.2 se describe la propuesta de paralelización del SA, y en la sección 5.3 se presenta el análisis de la complejidad del algoritmo paralelizado propuesto. En la seccion 5.4 de este mismo capítulo se presentan los resultados experimentales y por último, en la sección 5.5 se dan las conclusiones.

El capítulo 6 presenta los resultados obtenidos al aplicar el algoritmo propuesto de recocido simulado tanto en forma secuencial como en forma paralela y se muestra la eficiencia y eficacia de ambos algoritmos. En la sección 6.1 se da una introducción, en la sección 6.2 se presentan los resultados que muestran la convergencia del algoritmo propuesto en forma secuencial. En la sección 6.3 se muestran los resultados con el algoritmo de calendarización parcial (Parcial-S), que forma parte del SA propuesto y se hace un comparativo entre el Parcial-S y el Total-S. En la sección 6.4 se muestran los resultados obtenidos con el algoritmo de SA propuesto en forma secuencial y en la sección 6.5 se presentan los resultados obtenidos en forma paralela y en la sección 6.6 se confrontan los resultados obtenidos en forma secuencial contra los resultados obtenidos en forma paralela, y por último en la sección 6.7 se presentan las conclusiones.

El capítulo 7 presenta las conclusiones y los trabajos futuros derivados de este trabajo de investigación.

Capítulo 2

Flexible Job Shop Scheduling Sujeto a Disposiciones de Tiempo

Este capítulo describe el problema de asignación de tareas en un taller de manufactura flexible; partiendo de una descripción conceptual del problema FJSSP (por sus siglas en ingles, Flexible Job Shop Scheduling Problem), y se describe a detalle el modelo de grafos disyuntivo como una propuesta de este trabajo de investigación partiendo del modelo propuesto por Roy y Sussman en 1964 para el JSSP clásico y el cual es adaptado para entornos de manufactura flexible para el FJSSP. Por último, se describe el modelo matemático partiendo del modelo de grafos disyuntivo aplicado a FJSSP y los tipos de problemas existentes.

2.1 Introducción

El Flexible Job Shop Scheduling Problem (FJSSP) es una extensión del problema de asignación de tareas en talleres de manufactura clásico, el JSSP [Mastrolilli *et al.*, 1998; Kacem *et al.*, 2002; Ho *et al.*, 2004; Kacem *et al.*, 2003; Zribi *et al.*, 2004; Tamaki *et al.*, 2001; Xia *et al.*, 2005]. Una instancia del JSSP consiste de n trabajos y m máquinas, donde cada uno de los trabajos es una secuencia de O operaciones, M=O, (exactamente una operación por máquina) que deben ser ejecutadas en un cierto orden. Cada operación tiene una asociada una duración $\tau(O_i)$, en una asignación de tareas se fija un tiempo de inicio $st(o)$ para cada operación, tal que:

1. Una máquina no pueda procesar más de una operación en un mismo instante (restricción de capacidad de recursos), y
2. El orden de las operaciones en cada trabajo es respetado (orden tecnológico o restricción de precedencia).

El tiempo total de la calendarización (Makespan) es igual al tiempo máximo de terminación (el tiempo de inicio más la duración) de la última operación asignada. El objetivo del JSSP es encontrar una asignación que minimice el makespan. Estas mismas características son heredadas al FJSSP, pero además se le agrega una serie de restricciones al FJSSP que su antecesor no tiene: para cada una de las operaciones de un trabajo existen más de una máquina que puede ser elegida para ser procesada y en ciertos casos, se requiere considerar un grupo de tiempos utilizados para diversas tareas, aparte de los tiempos requeridos (costos) para el procesamiento de las operaciones mismas: primero, un tiempo requerido para preparar a las máquinas para que puedan procesar un conjunto de operaciones que forma parte de un proceso; segundo, un tiempo de limpieza en las máquinas para recibir el siguiente grupo de operaciones del siguiente proceso, y tercer y último tiempo a considerar es el tiempo de intercambio de una operación a otra. Esta última característica empata perfectamente con lo que se experimenta en los sistemas de manufactura real, ya que conforme avanzan los desarrollos de equipos para procesar las operaciones, los sistemas de manufactura se tienen que flexibilizar a fin de cumplir con las demandas de producción, esto hace interesante el estudio de este problema por que tiene que ver con la administración de recursos. En la literatura, este grupo de tiempos son conocidos como disposiciones de tiempo.

Otros autores [Choi *et al.,* 1997; Cheung *et al.,* 2001; Choi *et al.*, 2002; Souissi *et al.*, 2004] describen el uso de disposiciones de tiempo aplicado al JSSP. Se reconoce que para FJSSP con disposiciones de tiempo son muy escasos los trabajos de investigación que abordan este tipo de problemas [Xia, *et al.* 2005; Freitas y Aparecida., 2006]. Algunos acercamientos para fJSSP se describen en la literatura encontrada [Ho *et*

al., 2004; Kacem *et al.*, 2003; Kacem *et al.*, 2002; Zribi *et al.*, 2004; Tamaki *et al.*, 2001; Xia *et al.*, 2005].

2.2 Descripción conceptual del problema

El problema FJSSP en general consiste de un conjunto de N trabajos, un conjunto de M máquinas y un conjunto O de operaciones, donde cada uno de los trabajos es un subconjunto de O en forma secuencial. Cada operación tiene una duración $\mu(O_i)$. En una asignación de trabajos, un tiempo de inicio $st(O_i)$ es definido para cada operación, de tal manera que:

1. Una máquina no puede procesar más de una operación en un tiempo.
2. El orden de ejecución de las operaciones en cada trabajo es respetado.
3. Cada una de las operaciones tiene más de una máquina que la puede procesar.

El tiempo en el cual la última operación es ejecutada dentro del proceso de FJSSP es conocido como el makespan. Usualmente uno de los objetivos de las búsquedas de FJSSP es encontrar una calendarización que minimice el makespan, el cual involucra disposiciones de tiempo que van desde la preparación de cada una de las máquinas para recibir la primer operación, los tiempos de intercambio entre operaciones y el tiempo de limpieza de cada una de las máquinas para recibir el siguiente proceso.

2.3 El modelo de grafos disyuntivo

Este trabajo de investigación propone un modelo matemático que representa el FJSSP, el cual tiene como base al modelo de grafo disyuntivo $G = (V, A, E, F, \mu)$ como una extensión al modelo introducido por Roy and Sussmann [Roy y Sussman, 1964] para el problema clásico JSSP.

$$V = O \cup \{I, D\},$$

$$A = \{[I, O_{1j}], [O_{ij}, O_{(i+1)j}], [O_{mj}, F] \mid \forall i, j : O_{ij} \in O\}$$

$$E = \{\{O_{ij}, O_{i'j'}\} \mid \forall i, j, i', j', j \neq j' : O_{ij}, O_{i'j'} \in O \wedge M_k(O_{ij}) = M_k(O_{i'j'})\}$$

$$F = \{\{O_{ij}\} \mid \forall i, j : O_{ij} \subseteq O \wedge M_{ij}(O_{ij}) \subseteq M\}$$

$$P = \{\{p_{ijk} + st_{ijk}\} \mid \forall i, j, k : O_{ij} \in O \wedge M_k(O_{ij}) \in M \wedge (p_{ijk} + st_{ijk}) > 0\}$$

$$\mu : V' \rightarrow IN$$

Donde:

Los vértices en V representan las tareas. Adicionalmente se agregan dos vértices que no tienen tiempo de procesamiento, estos son el origen I y el destino D. El peso de un vértice es $\mu(O)$ dado por el tiempo de procesamiento $p(O)$ por lo que $\mu(O) := p(O) + s(O), (\mu(I) = \mu(F) = 0)$, dado que I y D no consumen recursos. La fuente I es un nodo del cual salen todas las primeras tareas de cada trabajo y el destino D es el nodo en el cual llegan todas las tareas finales de cada uno de los trabajos. La restricción de precedencia entre $O_{ij}, O_{(i+1)j}$ es representado por un arco dirigido $[O_{ij}, O_{(i+1)j}] \in A$.

Similarmente, para cada par de tareas $O_{ij}, O_{i'j'} \in O$ existe una máquina que las puede procesar por lo que $M(O_{ij}) = M(O_{i'j'})$, la restricción disyuntiva es representada por un arco disyuntivo $\{O_{ij}, O_{i'j'}\} \in E$ y las dos

formas de dirigir la disyunción corresponden a las dos posibles orientaciones de $\{O_{ij}, O_{i'j'}\}$ (restricciones de capacidad de recursos). El conjunto de elementos F, corresponde a la propiedad que tienen los sistemas flexibles, donde una operación que forma parte del conjunto O, puede hacer uso de más de una máquina del sistema para ser procesada. El conjunto P corresponde a los tiempos de procesamiento p_{ijk} más los tiempos correspondientes a disposiciones de tiempo definido por st_{ijk} y que corresponde a los tiempos de preparación, intercambio y limpieza en una misma máquina, donde $p_{ijk} + st_{ijk} > 0$ y para toda máquina $M_k(O_{ij}) \in M$.

2.4 El modelo de formulación disyuntiva

En la literatura se reconoce que los problemas de optimización pueden ser modelados utilizando técnicas de programación matemática, y una de las formas más comunes de modelación matemática para este tipo de problemas es mediante el modelo de programación entera mixta (MIP) el cual es un lenguaje de modelación estándar [Agarwal y Grossman, 2004]. En [Fattahi *et al.,* 2007] se propone un modelo para describir de manera formal el problema de FJSSP mediante MIP. La función objetivo que se plantea es minimizar el tiempo total de procesamiento de todas las operaciones dadas ($C\max$), considerando las restricciones de precedencia y de capacidad de recursos, por lo que el modelo matemático que lo representa es el que se describe a continuación bajo lo siguientes criterios y parámetros:

Parámetros: (para i = 1,…,m; j = 1,…., n; h = 1,….,hj;)
n: Número de trabajos
m: Número de máquinas
$a_{i,j,k}$ Describe que máquina del conjunto de $m_{j,h}$ de máquinas

disponibles es asignada a la operación $O_{i,h}$

$$a_{i,j,h} = \begin{cases} 1 \ si \ O_{j,h} \ \ puede\ ejecutarse\ en\ la\ màquina\ i \\ 0 \ \ lo\ contrario \end{cases}$$

$p_{i,j,h}$ es el tiempo de procesamiento de la operación $O_{j,h}$ si se ejecuta sobre la máquina $m_{j,h}$.

$y_{i,j,h}$ Describe el conjunto de máquinas disponibles

M es un número muy grande.

Variables de decisión (para i = 1,…,m; j = 1,…,n; h = 1,…,hj; k = 1,…,ki;)

$C_{\max}$: Makespan o máximo tiempo de término del último de los trabajos.

$C_{\max} = t + Ps$, donde t es el tiempo de inicio de la operación y Ps es el tiempo de procesamiento de la misma operación.

Bajo estos criterios y anotaciones, el problema es encontrar una calendarización que minimice el makespan, por lo que el problema se determina por la siguiente función objetivo:

Minimice $C_{\max}$

Sujeta a:

$C_{\max} \geq t_{j,hj} + Ps_{j,hj}$	para j=1,…,n;	(1)
$\Sigma y_{i,j,h} \cdot p_{i,j,h} = Ps_{j,h}$	Para j=1,…,n; h=1,…,h_j	(2)
$t_{j,h} + Ps_{j,h} \leq t_{j,h+1}$	Para j = 1, . . . , n; h = 1, . . , $h_j - 1$	(3)
$Tm_{i,k} + Ps_{j,h} \cdot x_{i,j,h,k} \leq Tm_{i,k+1}$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h j ; k = 1, . . . , ki − 1;	(4)
$Tm_{i,k} \leq t_{j,h} + (1 - x_{i,j,h,k}) \cdot L$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h_j ; k = 1, . . . , k_i ;	(5)
$Tm_{i,k} + (1 - x_{i,j,h,k}) \cdot L \geq t_{j,h}$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h_j ; k = 1, . . . , k_i ;	(6)

$y_{i,j,h} \leq a_{i,j,h}$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h_j ;	(7)
$\sum_j \sum_h x_{i,j,h,k} = 1$	Para i = 1, . . .m; k = 1, . . . , k_i ;	(8)
$\sum_i y_{i,j,h} = 1$	Para j = 1, . . . n; h = 1, . . . , h_j ;	(9)
$\sum_k x_{i,j,h,k} = y_{i,j,h}$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h_j ;	(10)
$t_{j,h} \geq 0$	Para j = 1, . . . , n; h = 1, . . . , h_j ;	(11)
$Ps_{j,h} \geq 0$	Para j = 1, . . . , n; h = 1, . . . , h_j ;	(12)
$Tm_{i,k} \geq 0$	Para i = 1, . . . ,m; k = 1, . . . , ki ;	(13)
$x_{i,j,h,k} \in \{0, 1\}$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h j ; k = 1, . . . , ki ;	(14)
$y_{i,j,h} \in \{0, 1\}$	Para i = 1, . . . ,m; j = 1, . . . , n; h = 1, . . . , h j ;	(15)

Donde la restricción (1) determina el makespan. La restricción (2) determina el tiempo de procesamiento de la operación $O_{j,h}$ para la máquina seleccionada. La restricción (3) forza a que cada trabajo siga una secuencia de operación específica. La restricción (4) restringe a que cada máquina procese una operación en un tiempo. Las restricciones (5) y (6) forzan que cada operación $O_{j,h}$ pueda comenzar después de que le sea asignada una máquina y cuando su operación precedente $O_{j,h+1}$ haya sido procesada (restricción de precedencia). La restricción (7) determina el conjunto de máquinas disponibles para una operación. La restricción (8) asigna las operaciones a una máquina y la secuencia de operaciones en las demás. Las restricciones (9) y (10) forzan a que cada operación sea ejecutada por una sola máquina. La restricción (11)

corresponde al tiempo de inicio de procesamiento para la operación $O_{j,h}$. La restricción (12) determina el tiempo de procesamiento de la operación $O_{j,h}$ después de seleccionar una máquina. La restricción (13) corresponde al tiempo de inicio del trabajo en turno para la máquina i con prioridad k. La restricción (14) toma valores de 1, si la operación $O_{j,h}$ es ejecutada sobre la máquina i con prioridad k y toma el valor de cero para el caso contrario. La restricción (15) toma valores de 1 si la máquina i es seleccionada para ejecutar la operación $O_{j,h}$ y toma el valor de cero para el caso contrario. El modelo anterior representa el problema abordado en este trabajo de investigación.

2.5 Tipos de Problemas de FJSSP

En la literatura se reconocen dos tipos de problemas de Flexible Job Shop Schedulig de manera general: Total-FJSSP y Parcial-FJSSP [Ho *et al.,* 2004; Kacem *et al.,* 2002; Noureddine *et al.,* 2007; Hong *et al.*, 2006]. En el caso de Total-FJSSP todas las máquinas están disponibles para procesar el total de las operaciones involucradas en el proceso como se puede ver en la F figura 2-1.

Se sabe que revisten de mayor complejidad los problemas de tipo parcial [Kacem *et al.*, 2002; Ho *et al.*, 2004], aunque con el algoritmo de calendarización propuesto en este documento es posible abordar ambos tipos de problemas.

		M_1	M_2	M_3
J_1	$O_{1,1}$	6	3	5
	$O_{1,2}$	2	4	2
	$O_{1,3}$	6	5	6
J_2	$O_{2,1}$	4	6	5
	$O_{2,2}$	6	5	6
	$O_{2,3}$	4	5	3
J_3	$O_{3,1}$	2	3	4
	$O_{3,2}$	4	2	6
	$O_{3,3}$	5	4	2

Figura 2-1 Total FJSSP

En [Xia, *et al.* 2005; Freitas y Aparecida., 2006] se reconocen dos tipos de disposiciones de tiempo entre operaciones, uno conocido como tiempo de intercambio independiente en el cual el tiempo de intercambio esta asociado a una operación en particular. El otro tipo de disposición de tiempo, se conoce como disposiciones de tiempo dependiente, donde el tiempo de intercambio en una máquina en particular, de una operación a otra requiere un tiempo de intercambio. En este trabajo de investigación se aborda el FJSSP con disposiciones de tiempo dependiente, considerando que este tiempo puede corresponder a tres eventos diferentes:

1. Disposición de tiempo para preparar una máquina y pueda recibir la primera operación.
2. Disposición de tiempo que corresponde al tiempo de intercambio entre una operación y otra en una máquina.
3. Disposición de tiempo que corresponde al tiempo de limpieza en una máquina.

Este trabajo de investigación aborda problemas del tipo parcial Figura 2-2, con disposición de tiempo, por tener aún más similitud con los problemas que se presentan en la industria pues del total de las máquinas que puedan existir en un taller, no siempre todas pueden procesar todos los trabajos como sucede en un ambiente de Flexibilidad Total.

		M_1	M_2	M_3
J_1	$O_{1,1}$	6	*	5
	$O_{1,2}$	2	4	*
	$O_{1,3}$	6	*	*
J_2	$O_{2,1}$	4	6	5
	$O_{2,2}$	6	*	6
	$O_{2,3}$	*	5	3
J_3	$O_{3,1}$	2	3	*
	$O_{3,2}$	4	2	6
	$O_{3,3}$	5	4	*

Figura 2-2 Parcial FJSSP

Los tiempos asociados a la disposición de tiempo en cada máquina se reflejan en una matriz de NxN para cada una de ellas, y en la cual se identifican los tiempos de intercambio entre una operación y otra en una máquina en particular. El problema FJSSP en general consiste de un conjunto de N trabajos, un conjunto de M máquinas y un conjunto O de operaciones.

En este tipo de problema, donde cada una de las operaciones es un subconjunto de O en forma secuencial. Cada operación O_i tiene una duración por lo que un tiempo de inicio $st(O_i)$ es definido para cada operación y además se le asocia un conjunto de disposiciones de tiempo que corresponden a: un tiempo de preparación en la máquina que la ha de procesar siempre y cuando ésta sea la primera en ella, o caso contrario se le agrega un tiempo de intercambio entre una operación y otra en la misma máquina. Al finalizar el proceso de todas las operaciones en la máquina, se le añade un tiempo de limpieza o de preparación para recibir el siguiente proceso.

Capítulo 3

Métodos Aplicados a FJSSP

Este capítulo presenta una descripción de los principales métodos de aproximación aplicados a FJSSP. Así mismo, se describen las principales funciones de vecindad propuestas en la literatura y las estrategias de evaluación de los movimientos que realizan a fin de mejorar las soluciones encontradas para este tipo de problemas.

3.1 Métodos de aproximación

En optimización combinatoria, el uso de algoritmos de aproximación para encontrar soluciones a problemas Np-duros en tiempos polinomiales son los más comúnmente utilizados, ya que los algoritmos exactos que resuelven problemas de este tipo resultan prohibitivos en costo debido al tamaño de la entrada para los problemas de este grupo. Por ello, lo que se busca con los algoritmos de aproximación, es encontrar soluciones aproximadas en tiempos polinomiales de ejecución restringidos a cotas conocidas. El tiempo requerido para resolver el problema de FJSSP se incrementa de manera exponencial de acuerdo al tamaño del problema, por esta razón, se utilizan metaheurísticas aplicadas a la búsqueda de soluciones para las instancias de FJSSP [Ho *et al.*, 2004; Kacem *et al.*, 2003; Zribi *et al.*, 2004]. Un gran número de metaheurísticas han sido propuestas para encontrar buenas soluciones para FJSSP en un tiempo polinomial. Estos algoritmos incluyen Recocido Simulado (SA) [Kolonko *et al.,* 1997],[Najid *et al.*, 2002] Búsqueda de Tabú(TS) [Barnes *et al.*, 1996-1999; Dauzere *et al.,* 1997; Novicki *et al.,* 1996; Hurink *et al.*, 2005], Colonia de Hormigas

(AS) [Bonabeau *et al.*, 2000; Kumar *et al.,* 2003; Doringo *et al.*, 1997; Noureddine *et al.,* 2007; Stutzle *et al.,* 2000; Colorni *et al.,* 1994], Algoritmos Genéticos(GA) [Rossi *et al.,* 2000; Rossi *et al.*, 2000; Ho *et al.,* 2004; Kacem *et al.*, 2002; Hong *et al.*, 2006], entre otros.

3.1.1 Recocido Simulado

Recocido Simulado (SA, por sus siglas en inglés, Simulated Annealing) es una técnica de búsqueda local estocástica que aproxima el valor mínimo de la función de costo $f : S \to \Re$ sobre un conjunto finito de soluciones S. En SA los umbrales son positivos y estocásticos. El SA realiza una búsqueda aleatoria orientada que fue introducida como una analogía de la física estadística del proceso simulado de una fundición de metal hasta que su energía mínima se alcanza, por lo que las configuraciones son análogas a los estados de un sólido, mientras que el costo de la función *f* y el parámetro de control *c* son equivalentes a la energía y a la temperatura respectivamente. La técnica de SA esta basado en propuestas independientes que se encuentran en [Kirkpatrick *et al.,* 1983] y [Cerny, 1985] quien adapto el trabajo presentando por [Metrópolis *et al.*, 1953] a los problemas de la optimización de restricciones.

El método de SA, se mueve de manera iterativa en el espacio de soluciones utilizando una función de vecindad denominada N(x). Cuando genera una nueva solución x´ de x, la solución candidata x´ es aceptada como la nueva solución si $f(x') < f(x)$ o si $f(x') > f(x)$ es rechazada o aceptada en base de la función de probabilidad de aceptación de Boltzmann P(x), la cual involucra el parámetro de control T, y la diferencia de los valores de la calidad de la solución (x´-x). Inicialmente T tiene valores muy altos y conforme el algoritmo progresa, T decrementa e influye en la probabilidad de aceptación de la solución x´.

Una contribución de la función de vecindad para SA es la que se propone en [Laarhooven *et al.*, 1992]. La cual consiste en hacer un intercambio entre operaciones que se encuentran en bloques críticos y

que son adyacentes entre si en una máquina. Dicha función de vecindad se basa en las características siguientes:

- Si $x \in R$ es una solución factible, entonces intercambiando dos operaciones críticas adyacentes que requieren la misma máquina puede nunca conducir a una solución infactible;
- Si el intercambio de dos operaciones no críticas adyacentes conduce a una solución factible x', entonces la trayectoria crítica en x' no puede ser más corta que la trayectoria crítica en x (porque la trayectoria crítica en x todavía existe en x');
- comenzando con cualquier solución factible x, allí existe una secuencia de los movimientos que alcanzarán una solución óptima x*(conocido como propiedad de conectividad).

Basado en la función de vecindad propuesta por Van Laarhooven y otros, se construyen un SA genérico gobernado por el tamaño de la vecindad la cuál esta dada por n(n-1), donde n corresponde al total de trabajos a ser procesados en problemas simetricos, y en el cual el número de máquinas es igual al número de trabajos. Otra definición importante de la vecindad para SA es proporcionada en [Matsuo *et al.,* 1988] donde se prueba que si dos operaciones adyacentes críticas i y j entonces son intercambiadas la nueva solución encontrada nunca será mejor si estas operaciones forman parte de la ruta crítica. En [Yamada *et al.,* 1994] se formula un método llamado recocido simulado de bloque crítico (CBSA) el cuál incrusta una función de vecindad derivada de bloques críticos dentro de SA, y en donde los valores de temperatura inicial y final se definen de tal forma que se pueda re intensificar el proceso de recocido a fin volver más eficiente la búsqueda. Yamada y Nakano [Yamada y Nakano, 1995a, 1996a] proponen un CBSA aumentado con el generador de calendarización activo de Giffler y de Thompson [Giffler y Thompson, 1960]. Por otra parte en [Kolonko, 1998] indica que SA aplicado a problemas de asignación de tareas en talleres de manufactura no es un proceso convergente y presenta un método híbrido que consiste en un

algoritmo genético sobrepuesto en un esquema de SA. La mayoría de los trabajos encontrados con SA para problemas de asignación de tareas en talleres de manufactura estan orientados a resolver el problema clásico de JSSP, por lo que resultan escasos aquellos aplicados para resolver FJSSP. Uno de los trabajos más importantes orientados a FJSSP se describe en [Najid *et al,* 2002] donde los autores proponen un algoritmo de recocidos simulado modificado para el control de la secuencia de enfriamiento. En el mismo sentido en [Xia y Wu, 2005] se describe un trabajo con SA orientado a resolver el FJSSP mediante un método de optimización híbrido para el control de la temperatura de SA. Por otra parte, se presenta un método de sintonización del parámetro de control de la temperatura de inicio de SA basado en la desviación estándar [Martínez-Rangel, *et al.* 2007] y que en este documento se extiende para abordar al problema de FJSSP mediante la aplicación de un algoritmo de calendarización y una estrategia de sintonización de la temperatura para controlar la secuncia de enfriamiento mediante el uso de datos estadisticos obtenidos a partir de la generación de una población de soluciones para generar la desviación estandar que se utilizará como medio de inicializar la temperatura en el SA.

3.1.2 Algoritmos genéticos

Otro grupo de algoritmos propuestos para resolver problemas de asignación de tareas en talleres de manufactura son los algoritmos Genéticos (GA) que se basan en un modelo abstracto de la evolución natural, tal que la calidad de las estructuras individuales al nivel más alto son compatibles con el ambiente en el que operan, lo que en los problemas de computación corresponden a las restricciones del propio problema a resolver. En [Holland, 1975; Goldberg, 1989] se presentan los primeros trabajos al respecto. Lo que caracteriza a un algoritmo genético es que debe tener un esquema de representación. Estos esquemas de representación se pueden agrupar en dos enfoques de codificación básicos: los directos y los indirectos. El enfoque directo codifica una calendarización de una estancia en particular, mientras que un

cromosoma y los operadores genéticos son usados para enviar estos cromosomás en schedules que han sido optimizados. Mientras que en el enfoque indirecto, una secuencia de preferencias de decisión se codifica en el cromosoma, por ejemplo enviar reglas para asignaciones de trabajo, y los operadores genéticos son aplicados para mejorar el orden de varias preferencias. Uno de los métodos indirectos pioneros en esta área es el algoritmo presentado en [Davis, 1985]. Su técnica construye un orden preferido de operaciones para cada máquina. Tal enfoque es ampliado más a fondo en [Falkenauer y Bouffouix, 1991] donde se codifican las operaciones para ser procesadas en una máquina como lista de preferencia que consiste en una cadena de símbolos. Uno de los trabajos que basan su representación en una lista de preferencias es el que se presenta en [Kobayashi *et al.,* 1995] donde un cromosoma es una cadena de símbolos de longitud n y de cada símbolo identifica la operación para ser procesada en una máquina. En [Tamaki y Nishikawa ,1992] aplican una representación indirecta basada en un grafo disyuntivo, donde un cromosoma corresponde a una secuencia binaria de arcos disyuntivos.

En [Nakano y Yamada, 1991] se presenta una codificación binaria basada en la relación de precedencia de operaciones en la misma máquina. Los mismos autores presentan en [Nakano y Yamada, 1992] se extiende su trabajo definiendo a un operador de cruzamiento llamado GA/GT basado en el conocido algoritmo de Giffler y Thompson [Giffler y Thompson, 1960]. En [Bierwirth, 1995] se presenta un algoritmo genético basado en permutación y que mejora algunos métodos existentes. Otro trabajo relacionado es el que se presenta en [Shi, 1997] en el cual aplica una técnica de cruzamiento que divide aleatoriamente un cromosoma elegido en dos subconjuntos, de los cuales produce un cromosoma descendiente. Para superar algunos de los problemas que presentan los algoritmos genéticos se requiere que éstos tengan mecanismos de búsqueda local, que en cómputo evolutivo son conocidos como búsqueda local genética (GLS por sus siglas en inglés) o también en algunas referencias se conoce como búsqueda memética [Grefenstette, 1987];

[Moscato, 1989]. Uno de los trabajos más conocidos de búsqueda local genética es el que se presenta en [Dorndorf y Pesch, 1995] cuya propuesta está basada en dos enfoques, el primero de ellos dirige una combinación probabilística, mientras que el segundo enfoque, designado SB-GA, controla la selección de nodos. Una investigación detallada de la aplicación de técnicas evolutivas para problemas de calendarización puede ser encontrada en [Mattfeld, 1996].

En [Yamada y Nakano, 1995b] se generan dos calendarizaciones (cromosomas padres) que serán utilizadas como base para generar los cromosomas descendientes, estos cromosomas padres son generados tan lejos como sea posible. Comenzando la búsqueda en el primer padre substituyendo de manera iterativa una solución en la población actual con una secuencia mejorada parcial hacia el segundo padre. Otras mejoras a este método han sido llevadas a cabo por Yamada y Nakano, cuyos resultados se presentan en [Yamada y Nakano, 1996b] donde se adiciona un esquema de reemplazo estocástico parcial que favorece las soluciones que están más cercas al segundo padre y a se aplica una función de vecindad de bloque crítico, ambos métodos se basan en la idea de trayectoria en búsqueda tabú presentada por Glover y Laguna en [Glover y Laguna, 1997]. Uno de los trabajos más recientes en ésta área y enfocados al problema de FJSSP es el que se presenta en [Kacem et al.. 2002a] donde se propone un estudio basado en el modelado de un algoritmo genético para resolver el problema y los mismos autores en [kacem et al., 2002b] mejoran su propuesta mediante una hibridación de un algoritmo genético utilizando lógica difusa. En [Ho *et al.*, 2004] se propone un algoritmo Genético eficiente cultural para resolver el FJSP mediante el uso de una metodología denominada GENACE. Y más recientemente en [Ong *et al.*, 2005] se presenta un trabajo donde aplican el principio de selección clonal del sistema inmune humano para resolver de manera más específica el problema FJSSP con re-circulación.

3.1.3 Búsqueda Tabú

La técnica de optimización iterativa denominada búsqueda tabú (TS) es en esencia un procedimiento de búsqueda orientado determinista simple [Laguna *et al.,* 1993a], el cuál restringe la búsqueda a fin de optimizar la búsqueda local a través del almacenamiento de la historia del proceso en su memoria. Se prohiben los movimientos en la vecindad que tiene ciertas cualidades ya almacenadas a fin de dirigir el proceso de búsqueda lejos de las soluciones que se duplican o se asemejan a soluciones generadas anteriormente. La función de la memoria a corto plazo permite mediante ciertos olvidos, a fin de aceptar movimientos que no estén en la lista tabú más reciente [Lagunas *et al.,* 1991]. Sin embargo el estado tabú de un movimiento no es absoluto. Los criterios de la aspiración permiten a un movimiento tabú ser seleccionado si es que éste logra un nivel de cierta calidad en la solución generada.

Las funciones a medio y largo plazo de la memoria se pueden también aplicar para proporcionar una exploración más amplia del espacio de la búsqueda. Las estrategias a medio plazo o intermedias se basan en las reglas de elección modificadas para fomentar los movimientos y las buenas soluciones históricamente encontradas donde éstos esquemas se vuelven generalmente atractivas en el dominio de búsqueda, ya que intensifican la búsqueda en estas regiones. Los métodos a largo plazo diversifican la búsqueda en las áreas previamente no exploradas. Se basan a menudo en la modificación de reglas para incorporar las cualidades en la solución que no se utilizan con frecuencia. En [Laguna *et al.* 1993b] sugieren la inclusión de las transferencias de trabajo que mejora la calidad de la solución, reduce tiempo de cómputo y permite que problemas más grandes sean solucionados. Por otra parte en [Barnes y Laguna, 1993] sugieren seis componentes primarios que permitan calendarizaciones de producción eficaz de TS. También acentúan la necesidad de los esquemas de memoria de medio y largo plazo que junto con una estructura de búsqueda tabú restringida mejoran la búsqueda. Por otra parte en [Barnes y Chambers, 1995] presentan un algoritmo que

aplica la ocupación en un rango específico de los valores de memoria a corto plazo y cada vez que se encuentra una nueva mejor solución, ésta se almacena en una lista principal que más tarde se recupera y se mete en una lista tabú, que sirve de referencia para encontrar nuevas soluciones. Un trabajo a considerar por su importancia es el que se presenta en [Gonzales *et al*, 2005] donde se incorpora una estrategia que acelera la búsqueda sin la necesidad de re calcular los tiempos de inicio de todas las operaciones para determinar el costo del movimiento, mediante una estimación de los mismos. Sin embargo, esta estrategia es solamente eficaz cuando las calendarizaciones semi-activas se permiten y como no puede dar el makespan exacto del movimiento, puede ser considerado solamente como una estrategia rápida de valoración. Sin embargo Taillard incorpora este esquema, con la vecindad en [Laarhooven *et al.,* 1992] en su algoritmo de TS. Por su parte en [Dell' Amico y Trubian, 1993] incrustan en su algoritmo de TS un método que utiliza una característica bidireccional. Se formulan dos funciones de vecindad. La primera vecindad considera la inversión posible de hasta tres arcos o movimientos para generar una solución, si cualquiera de estos movimientos concuerdan con una solución que es tabú, el movimiento entero con los tres cambios se considera prohibido, en caso contrario, cada vez que un movimiento se acepta este se incluye en la lista tabú. La segunda función de la vecindad se basa en bloques críticos y para acelerar la búsqueda Dell' Amico y Trubian adaptan la estrategia de valoración de Taillard [Taillard, 1994] a sus vecindades. Otro de los métodos a tomar muy en cuenta es el que se presenta en [Nowicki y Smutnicki, 1996] donde se aplica una vecindad altamente restringida que divide una sola trayectoria crítica en bloques, y en donde solamente se aceptan intercambios de estos bloques críticos, este trabajo es uno de los más referenciados en la literatura al respecto. En [Thomsen, 1997] se presenta un algoritmo que mejora las soluciones de varios benchamarks mediante la combinación de la búsqueda tabú con un algoritmo de ramificación y acotamiento (B&B). La estrategia B&B se utiliza para la

diversificación. No obstante, para mantener tiempos de cálculo razonables, se realiza solamente heurísticamente, sobre un número restringido de iteraciones y para un límite de profundidad en la búsqueda del árbol. La vecindad de Nowicki y de Smutnicki se utiliza como el esquema de ramificación. En la literatura más reciente podemos citar el trabajo de [Chambers y Barnes, 1996] que presentan un método eficiente para resolver el FJSSP basado en búsqueda tabú, y de manera muy especial resalta el trabajo de [Saidi-Mehrabad y Fattahi, 2007] que presentan un acercamiento efectivo basado en los algoritmos de búsqueda tabú para resolver el FJSSP mediante un enfoque jerarquico basado en dos mecanismos heurísticos para resolver el problema. Los mecanismos que ellos proponen están basados en la forma de asignar cada una de las operaciones en las máquinas que las pueden procesar. Por otra parte en [Mastrolilli y Gambardella, 2000] se propone un acercamiento integrado basado en una búsqueda tabú mediante el uso de un grafo disyuntivo para representar soluciones y sus vecindades. La principal contribución de este acercamiento es la reducción del tamaño de la vecindad, lo que permite que se mejoren las soluciones obtenidas por otros métodos aplicados al mismo tipo de problemas.

3.1.4 Colonia de hormigas

La optimización de la colonia de hormiga (ACO) fue descrita por Dorigo en su tesis Doctoral [Dorigo, 1992] e inspirada por la capacidad y la organización de la colonia de hormiga real que usa los rastros químicos externos de ferómonas que actúan como medios de comunicación. Los algoritmos Colonia de Hormigas (o algoritmos sistema hormiga) son algoritmos inteligentes utilizados para resolver problemas de optimización en general, donde el estudio desde el punto de vista biológico de una colonia de hormigas presenta que su comportamiento es altamente estructurado y además se conoce que una hormiga tiene capacidades limitadas [Colorni *et al.*, 1994]. Los algoritmos del sistema de hormiga se han empleado extensamente desde entonces en los problemas de

optimización combinatoria NP-hard del tipo JSSP. En [Noureddine *et al.*, 2007] se presenta un algoritmo basado en el método de colonia de hormigas y búsqueda tabú para el FJSSP.

La metaheurística se compone de 3 pasos fundamentales:

- Actividad de las hormigas.
- Evaporación de la ferómona.
- Decisiones globales (opcionales).

El sistema de Hormigas simula el comportamiento de las hormigas reales, en los que iterativamente se añaden componentes de soluciones parciales tomando en cuenta entre otros elementos la información del problema a resolver, que consiste en identificar trazos de ferómona artificial que cambia dinámicamente durante la ejecución del algoritmo para reflejar la experiencia adquirida por las hormigas en la búsqueda de cada solución. Donde el principio básico del algoritmo son los siguientes parámetros:

- Una población de L hormigas artificiales que construyen de manera cíclica soluciones para problemas de optimización combinatoria.
- El algoritmo impone una definición del problema de optimización en un grafo.
- Las hormigas para construir rutas que representan soluciones utilizan una regla de transición. Esta regla esta representada por la ecuación (9).

$$P_{ij} = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[1/d_{ij}\right]^{\beta}}{\sum_{j \notin denodospermitidos}\left[\tau_{ij}(t)\right]^{\alpha}\left[1/d_{ij}\right]^{\beta}} \qquad (9)$$

Donde

τ_{ij} - Es la cantidad de feromonas en el arco que conecta a un *nodo*$_i$ y a un *nodo*$_j$

d_{ij} - Distancia Heurística entre un *nodo*$_i$ y a un *nodo*$_j$

p_{ij} - Es la probabilidad de ramificarse desde un *nodo*$_i$ y a un *nodo*$_j$

Los parámetros α y β sintonizan la importancia relativa en probabilidad de el monto de feromonas versus la distancia heurística entre dos nodos. Cuando todas las hormigas tienen construido una solución completa, esto es una secuencia de nodos visitados que resultan en una solución completa para el problema, el ciclo es completado y las feromonas actualizadas mediante una regla llamada Regla Global de Actualización (10).

$$\tau_{ij}\left(t+n\right)=\left(1-\rho\right).\tau_{ij}\left(t\right)+\rho.\Delta\tau_{ij}\left(t+n\right) \qquad (10)$$

Donde:

$$\Delta\tau_{ij}(t+n)=\begin{cases} Q/f_{Evaluación}(\text{de\_lo\_mejor\_que\_hay}) \\ \\ 0 \qquad \text{en otro caso} \end{cases}$$

ρ - Es el coeficiente de evaporación

Q - Es la cantidad de feromonas por unidad de distancia

3.2 Métodos de Exactos

En optimización combinatoria, el uso de algoritmos exactos o enumerativos para encontrar soluciones a problemas Np-duros en tiempos polinomiales no son comúnmente utilizados, ya que los algoritmos exactos que resuelven problemas de este tipo resultan prohibitivos en costo debido al tamaño de la entrada para los problemas de este grupo. Al aplicar este tipo de algoritmos, lo que se hace de manera regular es relajar los problemas a fin de poder encontrar mejores soluciones. Para el problema que nos ocupa, el FJSSP no existen referencias en la literatura donde se hayan aplicado métodos exactos o enumerativos para tratar de resolver el problema, pero dada las características de éste y siendo una extensión del problema clásico JSSP, es posible aplicarlos, por ello se presentan dos de los principales métodos exactos que se conocen y son aplicados para JSSP [Bruker *et al.*, 1996] y que pueden extenderse para ser aplicados a FJSSP.

3.2.1 Programación Entera Mixta

El método simplex es un método de programación entera mixta [Agarwal and Grossmann, 2004; Kallrath, 2000], y forma parte de los métodos enumerativos, este método fue creado en 1947 por el matemático George Dantzig, es un procedimiento iterativo que permite ir mejorando la solución a cada paso. El proceso concluye cuando no es posible seguir mejorando más dicha solución. Partiendo del valor de la función objetivo en un vértice cualquiera, el método consiste en buscar sucesivamente otro vértice que mejore al anterior. La búsqueda se hace siempre a través de los lados del polígono (o de las aristas del poliedro, si el número de variables es mayor). Cómo el número de vértices (y de aristas) es finito, se podrá encontrar la solución, siempre y cuando el espacio de soluciones del problema no sea intratable [Papadimitriou *et al.,* 1998]. El método simplex, es útil para resolver instancias pequeñas de JSSP y FJSSP (cuyo número de trabajos y máquinas no sobrepasen las 5 unidades) dado que el espacio de soluciones esta dado por $(n!)^m$, es evidente que un problema de 20x20 para ambos tipos de problemas resulta un espacio de soluciones intratable por el método simplex, ya que $(20!)^{20}$ equivale a la edad del universo dado en microsegundos [Muth y Thompson, 1963], por lo que con el método simplex, se tendría que recorrer en forma secuencial cada una de esas aristas.

El método simplex se basa en la siguiente propiedad: si la función objetivo (***f***) no toma su valor máximo en el vértice A, entonces hay una arista que parte de A, a lo largo de la cual ***f*** aumenta. El formato de la programación entera mixta para el FJSSP es simple, ya que solamente se requiere tener una función objetivo, que es minimizar el tiempo total de procesamiento de todas las operaciones dadas (Cmax). En este modelo sobresalen dos conjuntos de restricciones a considerar por su importancia, de las cuales uno de ellos tiene variables enteras binarias (de decisión) que son utilizadas para implementar las restricciones disyuntivas (restricciones de capacidad).

El primer grupo tiene que ver con las restricciones de precedencia, donde se indica que operación se inicia antes que otra, y el segundo grupo de restricciones tiene que ver con las restricciones de no traslape (disyuntivas) y cuyo propósito es garantizar que una máquina no procesará dos operaciones a la vez.

Para las restricciones de precedencia se va a denotar Cjk como el tiempo de terminación del trabajo j en la máquina k y a t_{jk} como el tiempo de procesamiento del trabajo j en la máquina k. Primero se va a considerar la restricción de precedencia de operaciones para una secuencia dada. Para un trabajo j, si el procesamiento en una máquina h precede al de una máquina k, se necesita seguir la siguiente restricción:

$$C_{jk} - t_{jk} \geq C_{jh}$$

Ahora se considerará la restricción de no-traslape de operaciones para una máquina dada. Para dos trabajos i y j, donde ambos necesitan ser procesados en la máquina k, si el trabajo i se procesa primero que el trabajo j necesitamos la siguiente restricción:

$$C_{jK} - C_{ik} \geq t_{jk}$$

De otra forma si j se procesa primero que i, entonces se requiere que se cumpla la siguiente restricción:

$$C_{iK} - C_{jk} \geq t_{ik}$$

Para el manejo de estas restricciones, es útil definir un coeficiente indicador como sigue:

X_{ijk}=1, si el trabajo i precede al trabajo j en la máquina k, o en su defecto

X_{ijk}=0, si el trabajo j precede al trabajo i en la máquina k.

Entonces se puede reescribir las restricciones anteriores como sigue:

$$C_{jK} - C_{ik} + M(1 - x_{ijk}) \geq t_{jk}$$

Donde i,j=1,2,3,..,n k=1,2,...,m

Donde M es un número positivo grande. La desigualdad anterior representa claramente la restricción de no traslape. Si el Job *i* se ejecuta

primero, x_{ijk} adquiere el valor de uno, lo que hace M(1- 1) adquiera el valor de cero, quedando la desigualdad falsa.

De otra manera si el Job j se ejecuta primero x_{ijk} toma el valor de cero, lo que hace que M(1-0) se hace un número muy grande haciendo la desigualdad verdadera.

Para el caso que nos ocupa, el problema de FJSSP, a la programación de las restricciones anteriores, mediante el método simplex, se le debe adicionar un mecanismo que identifique la máquina que ha de procesar la operación y el tiempo requerido para tal caso.

3.2.2 Ramificación y Acotamiento

Los algoritmos de ramificación y acotamiento (RA) utilizan una estructura arborescente dinámicamente construida como medios de representar el espacio de la solución de todas las secuencias factibles que puede tener un FJSSP. Este método es conocido como B&B (por sus siglas en inglés, Branch and Bound). La búsqueda comienza en el nodo más alto (nodo raíz) y se alcanza una selección completa (óptima) una vez que se haya evaluado el nodo del nivel más bajo. Cada nodo en un nivel p en el árbol de la búsqueda representa una secuencia parcial de las operaciones de p. Según lo implicado por una ramificación así como un acotamiento es aplicado para realizar la búsqueda. Apartir de un nodo (activo) no seleccionado, la operación de ramificación determina al sistema siguiente de los nodos posibles en los cuales la búsqueda podría progresar.

En los problemas de asignación de tareas, cada nodo del árbol representa frecuentemente a cada operación posible a ser secuenciada en la construcción de una calendarización (Schedule), de aquí que cada vez que se elija a un nodo para ramificarlo aparte de generar un nuevo nivel, se definirá la precedencia de la operación con respecto al Schedule parcial que se está construyendo. Una vez que se hayan elegido en el árbol a todas las operaciones del problema, el Schedule será total y por lo tanto se tendrá una solución del problema. Existen varios esquemas de

ramificación [Brucker y Thiele, 1996]: Ramificación sobre intervalos, ramificación sobre disyunciones, ramificación sobre conjuntos críticos y ramificación sobre permutaciones en rutas críticas. El éxito del método de B&B depende de la calidad de los acotamientos realizados en el espacio de soluciones, aún con ello, el método no será el adecuado para resolver problemas de este tipo (JSSP-FJSSP) cuyo instancias del problema estén catalogados como medianos o grandes en la literatura [Muth y Thompson, 1963].

Algunos de los algoritmos que utilizan procedimientos de ramificación y acotamiento son los siguientes:

Algoritmo de Pinedo [Pinedo, 2001]: El cual construye un árbol a partir de la elección de la operación con mayor prioridad de ejecución, la cuál ocurre cuando una operación candidata a ser secuenciada obtiene un retraso mayor que cualquier otro candidato, al resolver el problema en cuestión de una forma relajada como el de una máquina que minimiza el máximo retraso.

Algoritmo de Brucker [Brucker *et al.,* 1994]: Su característica principal se basa en un esquema de ramificación basado sobre un enfoque del bloques y se apoya con la representación del problema en forma de un grafo disyuntivo, donde es necesario partir de un Schedule completo para contar con el conjunto de bloques de operaciones, y así poder ramificar. Ellos aplican una heurística para encontrar una primera solución.

Algoritmo de Carlier y Pinson [Carlier y Pinson, 1989]: Este algoritmo utiliza el cálculo de los tiempos de descarga rj (release date) y las colas qj (delivery times), de cada operación j cuando se relaja el problema a una máquina. Con estos tiempos mediante un procedimiento propuesto que calcula la prioridad de ejecución de cada operación en la máquina que se evalúa, se obtienen las prioridades de las secuencias en las operaciones; el algoritmo crea conjuntos de operaciones con los que se puede iniciar la secuencia en una máquina de acuerdo al rj más pequeño que se tenga conjunto de operaciones con los que se puede

terminar la secuencia en una máquina de acuerdo al qj más grande que se tenga.

Algoritmo de Applegate y Cook [Applegate y Cook, 1991]: Es prácticamente el algoritmo de Carlier y Pinson, El procedimiento para el cálculo de la cota inferior para el problema relajado a una máquina es el mismo. La principal diferencia de este algoritmo respecto al de Carlier y Pinson, es que ellos calculan la cota superior mediante una heurística. Después lo mejoran aplicando las condiciones de Carlier y Pinson para dirigir arcos disyuntivos en un conjunto de máquinas. En las máquinas restantes se considera fijo el orden de las operaciones.

3.3 Funciones de vecindad

3.3.1 Antecedentes

La complejidad del tiempo de una búsqueda depende (1) del tamaño de la vecindad y (2) de la complejidad en la determinación de costo de los movimientos [Mastrolilli *et al.*, 2000]. Por lo tanto para que la búsqueda sea eficaz y eficiente, la vecindad debe ser altamente restringida y necesita ser evaluada rápidamente. Una función de la vecindad determina el sistema de las soluciones de vecindad que se pueden alcanzar de la solución actual realizando un movimiento o una transición. Tres características importantes de las funciones de la vecindad son tamaño, conectividad y viabilidad [Mattfeld, 1996].

El tamaño de la vecindad, es decir, el número promedio de movimientos posibles, es un parámetro importante para determinar el tiempo computacional. Una función de vecindad lleva a cabo la característica de la conectividad si la solución global óptima se puede alcanzar de cualquier solución con un número finito de movimientos, incluyendo movimientos que no mejoran la solución. Un movimiento sobre una solución actual, puede conducir a una solución factible infactible. Las vecindades que tienen la característica de la conectividad conducen siempre a las soluciones factibles, lo que es muy deseable al realizar la exploración. La

mayoría de las aplicaciones de búsqueda locales en JSSP se han adaptado para FJSSP, las cuales adoptan el esquema disyuntivo mediante una representación a través del modelo de grafos disyuntivo, y los movimientos de la vecindad son obtenidos invirtiendo arcos en la ruta crítica o realizando otros cambios en relaciones de la precedencia en la misma ruta crítica. A continuación se presentan las principales funciones de vecindad utilizadas en la literatura.

3.3.2 Función de vecindad de Van LaarHooven (N1)

Laarhoven y sus colaboradores [Laarhoven *et al.*, 1992] describen la función de vecindad conocida en la literatura como N1. Que se compone de los vecinos que se forman al realizar una permutación en un par de operaciones adyacentes que pertenecen a una misma máquina. Dado un diagrama de la solución D tal como se presenta en el grafo de la figura 3-1, un movimiento es generado invirtiendo un arco (v, W) en la ruta crítica, tal que la operación W es el sucesor inmediato de la operación v en una máquina K. La revocación de (v, W) da lugar siempre a una solución (acíclica) factible. Por otra parte, la revocación de arcos en la ruta crítica son las únicas revocaciones del arco que pueden reducir el valor del makespan. N1 tiene la característica de la conectividad. Por ejemplo, considerar la solución representada por el diagrama D en la figura 3-1 que es la ruta más larga (fuente, O_{31}, O_{11}, O_{12}, O_{21}, O_{32}, O_{33}, O_{13}, destino). La vecindad de D, N1 (D), consiste en las soluciones obtenidas invirtiendo los arcos siguientes: (O_{31}, O_{11}), (O_{12}, O_{21}), (O_{21}, O_{32}), y (O_{33}, O_{13}).

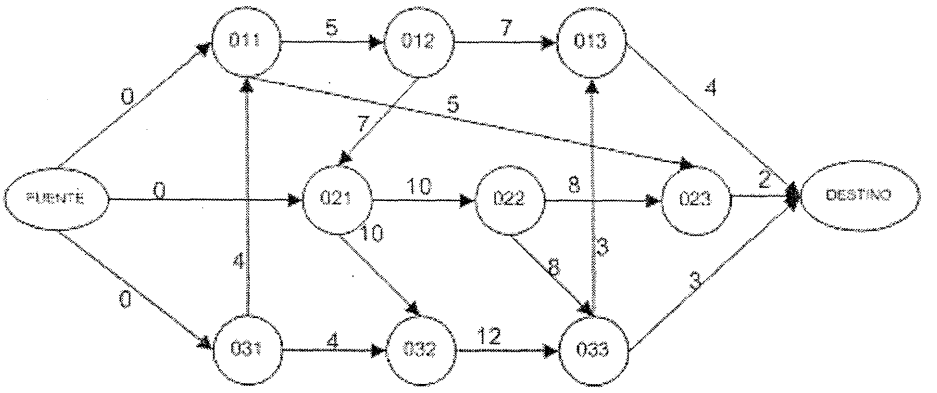

Figura 3-1 Estructura N1 [Laarhoven *et al.*, 1992]

Las características de N1 se pueden resumir de la siguiente manera:

a) Al hacer una permutación entre un par de operaciones adyacentes que pertenecen a la ruta crítica, el resultado es un Schedule factible.

b) Es posible obtener un Schedule con un menor makespan, si se permuta un par de operaciones que pertenecen a la ruta crítica del Schedule original.

c) Cuando se permuta en un Schedule un par de operaciones que no pertenecen a la ruta crítica, nunca se obtendrá un Schedule con un makespan menor al makespan del Schedule original.

d) Existe una alta probabilidad de que los schedules obtenidos en base a una permutación de operaciones no pertenecientes a la ruta crítica, sean infactibles, esto es, se obtendrán schedules con ciclos.

Lo anterior, ha permitido que esta función de vecindad sea la más referenciada en la literatura a través del uso de esta en los algoritmos que tratan de resolver problemas de Schedule.

3.3.3 Función de vecindad Matsuo (N2)

Matsuo y otros [Matsuo *et al.,* 1988] describen la función de vecindad conocida en la literatura como "Función de vecindad o estructura de vecindad de Matsuo" o N2. Dada una solución en el modelo de grafos disyuntivo para la solución que se presenta en [Laarhoven *et al.*, 1992] figura 3-1 de este documento, un movimiento es generado invirtiendo el arco (v, W) en la ruta crítica, tal que W es el sucesor inmediato de v en la máquina k, y el predecesor de v o el sucesor de W en la máquina k no está en la ruta crítica. Como se describió en la función de vecindad N1, la revocación de (v, W) produce siempre una solución factible. La revocación de los arcos definidos en N2 son las únicas revocaciones del arco que pueden mejorar una solución. Sin embargo, como lo muestra Amico y Turbian [Amico y Tubian, 1993] N2 no tiene la característica de la conectividad. Mientras que Aarts y otros [Aarts *et al.*, 1994] demuestran que la función de vecindad N2 superó la función de vecindad N1 para varios benchmarks de JSSP. La función de vecindad N2 es más pequeña que la función de vecindad N1, puesto que solamente un subconjunto de movimientos de N1 está disponible en N2.

Se puede concluir que la función de vecindad N2 es parecida a N1, en cuanto a que se generan tres permutaciones, cada permutación se hace en un par distinto de operaciones, y cada uno de los tres pares pertenece a diferente máquina.

3.3.4 Función de vecindad Nowicky y Smutnicki

La idea de usar una vecindad reducida para mejorar las características de cómputo de algoritmos de búsqueda local no es nueva. Por ejemplo, en [Laarhoven *et al.*, 1992] se presenta un trabajo que utiliza el concepto de arcos críticos en los problemas de scheduling para eliminar los movimientos que no mejorarán definitivamente la solución. Esta idea fue ampliada para otros problemas similares por otros autores como [Nowicky y Smutnicki, 1998] quienes de manera más formal proponen una definición específica de la vecindad basada en las rutas críticas, lo que

reduce la vecindad y hace que muchas metaheurísticas mejores las cotas encontradas. Con esta vecindad reducida y puestas en práctica junto con un algoritmo de búsqueda de tabú, mejoraron de manera drástica la velocidad del algoritmo propuesto.

El procedimiento local de la búsqueda de Nowicki y Smutnicki comienza identificando la ruta crítica, es decir la trayectoria más larga del modelo de grafos disyuntivo que representa una solución dada. Las operaciones que pertenecen a la ruta crítica se llaman operaciones críticas, agrupadas en bloques, tal como se presenta en la figura 3-2, en donde las operaciones que los integran no tienen tiempo de ocio entre sí y que pertenecen a una máquina en particular.

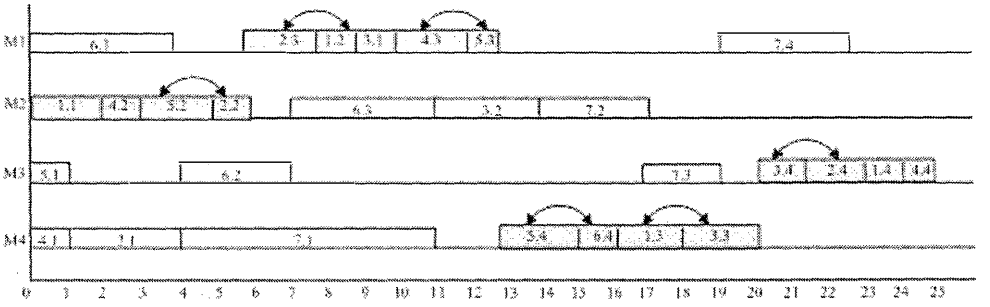

Figura 3-2 La ruta crítica compuesta por los bloques críticos y las operaciones posibles a intercambiarse [Nowicki y Smutnicki, 1996].

Capítulo 4

Algoritmo de Recocido Simulado Secuencial

Este capítulo describe el algoritmo de recocido simulado o SA (por sus siglas en ingles, simulated annealing) propuesto en este trabajo de investigación, y se explica a detalle dos elementos que lo hacen eficiente y eficaz, el primero de ellos, corresponde a la estrategia de sintonización mediante una medida de dispersión de uno de sus parámetros de control más importante que es la temperatura o secuencia de enfriamiento y el segundo elemento es un mecanismo de calendarización parcial cuya función permite que SA pueda explorar un mayor número de soluciones en un menor tiempo.

4.1 Introducción

La metaheurística de Recocido Simulado (SA) es una técnica de búsqueda local estocástica que aproxima el valor mínimo de la función de costo $f : S \rightarrow \Re$ sobre un conjunto finito de soluciones S y en la cual, la sintonización de la temperatura de inicio dentro de SA ayuda a converger o acercarse de manera más acelerada a un óptimo global para distintos problemas de scheduling. En SA los umbrales son positivos y estocásticos. El SA realiza una búsqueda aleatoria orientada que fue introducida como una analogía de la física estadística del proceso

simulado de una fundición de metal hasta que su energía mínima se alcanza, por lo que las configuraciones son análogas a los estados de un sólido, mientras que el costo de la función *f* y el parámetro de control *c* son equivalentes a la energía y a la temperatura respectivamente. Este método está basado en propuestas independientes que se encuentran en [Kirkpatrick *et al.,* 1983] y [Cerny, 1985] quien adaptó el trabajo presentado por [Metrópolis *et al.*, 1953] a los problemas de la optimización de restricciones.

En esta investigación, la propuesta hecha en [Martínez-Rangel *et al.,* 2007] para JSSP (por sus siglas en inglés, Job Shop Scheduling Problem) se retoma a fin de que se puedan abordar los problemas tipo FJSSP (por sus siglas en inglés, Flexible Job Shop Scheduling Problem). El problema más crítico que presenta SA tiene que ver con la sintonización de sus parámetros en especial la temperatura de inicio que determina la secuencia de enfriamiento, que es considerada dentro de SA como un parámetro de control [Kolonko *et al.,* 1997, Cruz-Chávez *et al.,* 2005, Martínez-Rangel *et al.*, 2007; Yamada y Nakano, 1995a, 1996a], pues de éste depende el rendimiento del algoritmo. Hay otros parámetros no menos importantes como son la calidad de las soluciones generadas, la distribución de probabilidades en el algoritmo de Metrópolis, que forma parte del SA para la aceptación de nuevas soluciones, así como la velocidad de enfriamiento.

Como se describe en [Martínez-Rangel *et al.*, 2007] respecto a la variabilidad de los datos encontrados en cada una de las instancias de prueba para problemas tipo JSSP, el FJSSP, al ser una extensión de JSSP, presentan por lo tanto la misma característica. Hace más de medio siglo, los matemáticos Bienaymé y Chebyshev [Kendall *et al.,* 1958] examinaron por separado dicha propiedad de los datos en torno a la media, y encontraron que, no importa cómo se distribuye un conjunto de datos, el porcentaje de observaciones que están contenidas dentro de distancias de mas-menos K desviaciones estándar alrededor de la media deben ser, cuando menos $(1-1/k^2)100\%$ (regla de Bienaymé y Chebyshev).

Este mismo análisis de variabilidad y del comportamiento del proceso de SA respecto a la distribución de los datos y a la aplicación de esta regla, se acepta que la temperatura inicial en el algoritmo de SA propuesto sea igual a dos veces la desviación estándar obtenida de una muestra de soluciones generada de manera aleatoria para problemas tipo FJSSP, y que en el análisis que hace la función de Boltzmann para aceptar o rechazar una solución dependa de si la nueva solución que se propone está en un rango (límite superior y límite inferior) determinado por la media de las soluciones generadas más o menos dos veces la desviación estándar a fin de aceptar por principio probabilístico el 75% de todas las soluciones generadas en el proceso de SA, que equivale al 95% en procesos que siguen una distribución normal.

4.2 Esquema generalizado de Recocido Simulado

El procedimiento general de SA [Kirkpatrick *et al.,* 1983] es definido de la forma siguiente:

1. Seleccionar un valor de inicio alto a T_0, un límite T_f para decrementar a T_0 y un estado inicial x_0

   $T_k \leftarrow T_0, \quad x \leftarrow x_0$

2. Para cada iteración K, $K = 1$ ....k_f hacer lo siguiente:
   a. Repetir hasta alcanzar el equilibrio:
      i. Calcular el valor del estado *x* mediante la función de costo, $E_k \leftarrow f(x)$
      ii. Generar un nuevo estado *x´* utilizando una función de vecindad, $x' \leftarrow N(x)$
      iii. Calcular el valor del estado x´ mediante la función de costo, $E_k' \leftarrow f(x')$.
      iv. Hacer $x \leftarrow x'$ de acuerdo a la probabilidad determinada por la función de aceptación P(x)

3. Reducir T para *k+1* utilizando un factor de control γ, $T_{k+1} \leftarrow \gamma * T_k$, donde $0 < \gamma < 1$.
4. Cuando T_{k+1} es menor que T_f, terminar
5. Retornar la mejor solución encontrada x y su valor de costo E

En general el método de Recocido Simulado consiste de un sistema de estados x y de las relaciones entre los siguientes elementos [Kolonko *et al.,* 1997, Cruz-Chávez *et al.,* 2005, Martínez-Rangel *et al.*, 2007; Yamada y Nakano, 1995a, 1996a]:

1. $f(x)$: Una función de costos a ser minimizada.
2. N(x): Un mecanismo de generación de vecindades (genera nuevos estados)
3. P(x): Una función de aceptación que decide si el nuevo estado se acepta o se rechaza
4. T(k): Un parámetro de control del recocido

Para problemas de optimización numérica, x es definida como un vector de números enteros o reales, y la función de Boltzmann P(x) es usada para la aceptación de nuevos estados. La función de Boltzmann emplea una función de densidad de probabilidades del tipo Gaussiana. La función de Boltzmann se define como:

$$P(x') = \begin{cases} 1 & \text{Si } f(x') <= f(x) \\ e^{(-f(x')-f(x))/T} & \text{En caso contrario} \end{cases} \qquad (1)$$

Para FJSSP, un estado x es definido por una solución S (un schedule) del problema. La función de costo f(x) es definida en este trabajo por el makespan $C_{max}(S)$. La vecindad N(S) de S se define como el conjunto de soluciones factible que pueden ser generadas a partir de S mediante un sólo paso, este paso es, una perturbación de un par de operaciones (*i, j*) asignadas en una máquina M_j.

4.3 Estrategia de sintonización de Recocido Simulado utilizando una medida de dispersión

El algoritmo de Recocido Simulado requiere que los parámetros utilizados tengan ciertos valores de inicio, los cuales, generalmente se determinan a priori. Dentro de los parámetros requeridos se incluyen los parámetros de control T_0 y T_f, el factor de temperatura γ que define la velocidad de decremento de T, así como la longitud de la cadena de Markov que define el número de iteraciones a efectuar en el algoritmo de Metrópolis antes de generar un decremento en T.

Cada problema de scheduling tipo FJSSP tiene diferentes características, y en consecuencia diferentes grados de dificultad, por lo que es necesario poder encontrar los valores adecuados de los parámetros involucrados en el proceso de recocido simulado. De acuerdo a la propuesta en [Martínez-Rangel *et al.*, 2007] para sintonizar SA, se requiere de índices que permitan conocer a lo largo del proceso si los valores de los parámetros utilizados son los correctos. Dos de los índices son la distribución uniforme P(x) de la probabilidad de aceptación de la función de Boltzmann (ecuación 3 definida más adelante) y la distribución P(n) de los n número aleatorios (0<n<1) generados para determinar la aceptación o rechazo de una solución en base a la función de distribución de Boltzmann. La siguiente ecuación:

$$\sum P(e^{(-f(x')-f(x))/T}) \cong \sum P(n) \qquad (2)$$

indica que la suma de la distribución de la probabilidad de aceptación de la función de distribución de Boltzmann debe ser aproximadamente igual a la suma de la distribución de probabilidad P(n) generada de manera aleatoria como se muestra en la figura 4-1 que muestra una distribución normal.

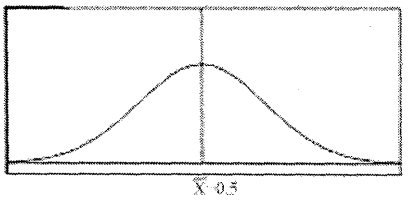

Figura 4-1 Distribución de $P(n)$

El problema que se tiene para lograr la igualdad en cuanto a la distribución uniforme de probabilidades es que $P(e^{(-f(x')-f(x))/T})$ tiene una distribución de Poisson, mientras que $P(n)$ tiene una distribución normal. Por principio probabilístico se acepta que $P(e^{(-f(x')-f(x))/T})$ tiene una mayor dispersión, esto por el comportamiento propio del fenómeno de SA, que al principio acepta grandes variaciones de energía (probabilidades muy altas) cuando T es muy alta: Conforme T decrece y tiende a cero, la variación de E_k aceptada por P(x) son mínimas. Por lo tanto la ecuación (2) se toma como un parámetro valido el cual esta compuesto de dos índices, para evaluar el comportamiento de SA (ver figura 4-2).

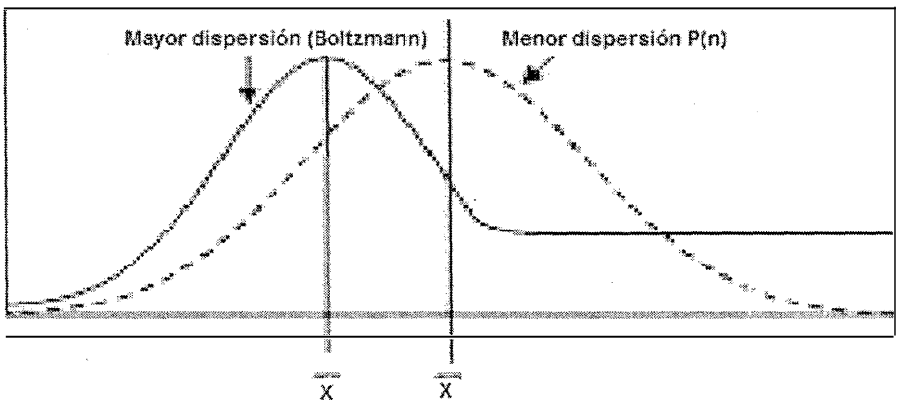

Figura 4-2 Probabilidad de distribución de $P(e^{(-f(x')-f(x))/T})$ y $P(n)$

Además de la ecuación 2, que nos muestra el comportamiento del proceso de SA, se tiene otro índice que nos permite sintonizar los parámetros requeridos en SA. Este índice es la desviación estándar de la calidad de las soluciones generadas que entran al proceso de SA

dentro del procedimiento de Metrópolis, y nos muestra el grado de dispersión que presentan las variaciones de E_k en SA. En el área de estadística, es de mucha utilidad conocer el grado de dispersión que tienen las soluciones a fin de considerar el grado de aceptación de dichas soluciones. Sin importar como se distribuye un conjunto de datos, el porcentaje de observaciones que están contenidas dentro de distancias de más o menos K desviaciones estándar alrededor de la media deben ser, cuando menos:

$$(1-\frac{1}{k^2})100\% \qquad (3)$$

Por lo tanto, para los datos cuyos polígonos adoptan cualquier forma, en el caso de SA, cuando menos [1-(1/22)] 100%=75.0% de las observaciones deben estar contenidas dentro de distancias de más o menos 2 desviaciones estándar alrededor de la media. 88.89% deben estar contenidas dentro de distancias más o menos 3 desviaciones estándar, y 93.75 con más o menos 4 desviaciones estándar. Esto es aplicable cuando se sabe que un fenómeno aleatorio particular no siguió al patrón de distribución normal, en el caso de SA, éste sigue un patrón de distribución de Poisson.

En esta investigación doctoral, se acepta que para sintonizar el grado de aceptación de una nueva solución de FJSSP, se puede elegir entre estas tres posibilidades, para el proceso de SA se aumenta en dos veces la desviación estándar al valor medio para obtener su límite superior y disminuirla en la misma proporción para obtener el límite inferior (figura 4-3), lo que equivale a aceptar el 75.00% de las soluciones en SA o el 95.44% en otros procesos con distribuciones normales como se observa en la tabla 4-1.

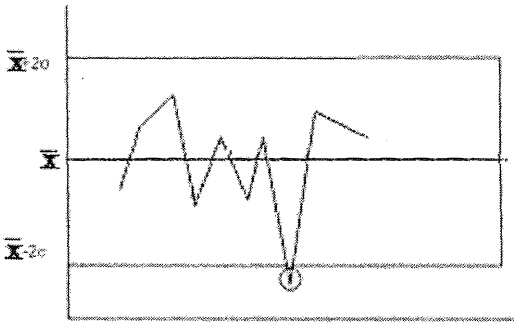

Figura 4-3 Zona mínima de ocurrencia para aceptar el 75% de las soluciones para esta evaluación en el SA propuesto

Tabla 4-1 Variación de los datos alrededor de la media de acuerdo al patrón de distribución

Número de K unidades de la desviación Estándar	Porcentaje de observaciones contenidas entre la media y K	
	Distribución de Poission (Bienayme-Chebyshev)	Distribución Normal
1	No Calculable	Exactamente 68.26%
2	Al menos 75.00%	Exactamente 95.44%
3	Al menos 88.89%	Exactamente 99.73%
4	Al menos 93.75%	Exactamente 99.99%

Dado que solamente se aceptan diferencias de energía, dado por $\Delta E_k = -(f(x') - f(x))$ que se encuentran en el rango delimitado por 2 veces la desviación estándar, podemos hacer que T_0 sea igual a 2 veces la misma desviación estándar dentro del proceso de SA (T0=2 $*\sigma$) a fin de que la distribución de $P(e^{(-f(x')-f(x))/T})$ sea uniforme dentro de SA. La

importancia de elegir una buena solución de inicio en SA para obtener buenas soluciones [Cruz-Chávez, *et al.*, 2005].

Entonces, en esta propuesta, los valores de cada uno de los parámetros involucrados en el proceso de SA en función de la desviación estándar se definen como sigue:

1. Para cada uno de los benchmarks utilizados en las pruebas, se genera un conjunto de soluciones Ω de forma aleatoria lo suficientemente grande que permite identificar la desviación estándar de la calidad de las soluciones que la componen, su valor medio, el valor mínimo y máximo encontrado.
2. La Velocidad de enfriamiento (longitud de la cadena de Markov), estará dada por el doble del tamaño de la vecindad del problema [Cruz-Chavéz *et al.*, 2007] de acuerdo a la función de vecindad utilizada. En [Laarhoven *et al.*, 1992] se describe la función de vecindad conocida en la literatura como N1 que se compone de los vecinos que se forman al realizar una permutación en un par de operaciones adyacentes que pertenecen a una misma máquina, y que en este trabajo es definida por la variable Ve = (n*(n-1))*2; donde n es el número de trabajos, los cuales determinan el ciclo de Metrópolis.
3. El valor de T_0 es igual a dos veces la desviación estándar ($2 * \sigma$) encontrada para el conjunto de soluciones generado en el punto 1.
4. El decremento de la temperatura esta dado por : $T \leftarrow \gamma * T$,donde γ=0.998. Este valor es obtenido mediante sintonización.
5. Una solución S´ del problema podrá ser evaluada para ser aceptada o rechazada mediante la función de distribución de probabilidades de Boltzmann, siempre y cuando ΔE_k no sea mayor a dos veces la desviación estándar encontrada para la vecindad (límite superior y límite inferior).

Los anteriores parámetros se usan como una sintonización inicial para ejecutar el proceso de recocido simulado. En la figura 4-4 se presenta el algoritmo de SA, el cual, en este trabajo de investigación se denomina Algoritmo de Recocido Simulado Acelerado (SA-A), que permite converger de manera más rápida a soluciones óptimas para los problemas de prueba de FJSSP.

```
1. Datos de entrada: S, σ, Makespan
2. Resultado: Makespan
3. T=2 *σ;
4. Tamaño_vecindad←n*(n-1)*2;
5. Mientras (T >0){
6. NVecindad←0;
7.     Mientras (NVecindad < Tamaño_vecindad ){
8.           Genera__nueva_solución (S´);
9.        Si (S´<= S)
10.       {
11.           S=S´ ;
12.           Si (Makespan > S)  Makespan←S;
13.           NVecindad++;
14.        } //fin  de si
15.       caso contrario
16.       {
17.          ΔS ←S´-S
18.          Si(ΔS <= 2 *σ ) {
19.       n←(0<probabilidad<1)
20.             if (n<exp (-ΔS /T)))  Then   {
21.                S ← S´  //Acepta la nueva solución
22.                 NVecindad++;
23.                 }
24.             caso contrario
25.               S←S;  //rechaza la nueva solución
```

26. *}//fin de si*
27. caso contrario
28. S←S; // rechaza la nueva solución
29. } //fin de si DeltaS <=
30. caso contrario
31. S←S; // rechaza la nueva solución
32. }
33. } //mientras
34. $T \leftarrow \gamma * T$ //decrementa la temperatura
35. }// fin de mietras $T_f > 0$

Figura 4-4 Algoritmo de SA-A con Desviación Estándar

De acuerdo a la figura 4-4, la diferencia de energia viene dado por $\Delta E_k = -(f(x') - f(x))$ la cual es equivalente a $\Delta S_k = -(f(S') - f(S))$ y la distribución de probabilidades $P(e^{(-f(x')-f(x))/T})$ es equivalente a la función de distribución de Boltzmann que viene dado por $P(e^{(-f(S')-f(S))/T})$. En en la linea 8 se hace referencia al mecanismo de calendarización que pemite explorar las soluciones dentro del espacio de soluciones para una instancia en particular de FJSSP. Este mecanismo se describe a continuación en la siguiente sección.

4.4 Algoritmo de Calendarización para Recocido Simulado

Una característica particular que tienen los algoritmos propuestos para resolver el problema de asignación de tareas en talleres de manufactura es que integran un mecanismo que les permite encontrar y evaluar las soluciones encontradas dentro de un espacio de soluciones que corresponden a un problema en particular. El tiempo requerido para encontrar la solución optima y su evaluación se incrementa de manera exponencial de acuerdo al tamaño del problema, [Garey *et al.*, 1976; Yamada *et al.*,1992; Fisher and Thomson, 1993; Maqrini *et al.*, 1995; Jain *et al.*, 1998], por esta razón, se utilizan metaheurísticas aplicadas a la búsqueda de soluciones para las instancias tanto de JSPP clásico como de FJSSP [Ho *et al.*, 2004; Kacem *et al.*, 2002; Noureddine *et al.*, 2007; Hong *et al.*, 2006; Kolonko *et al.*, 1997; Brandimarte *et al.*, 1993]. Estos algoritmos realizan sus búsquedas a través de vecindades. Por ello, el desarrollo de mecanismos para la generación de vecindades para este tipo de problemas de manera eficiente es importante. Estos mecanismos ayudan al trabajo que hacen las metaheurísticas incrementando su eficiencia cuando éstas son usadas para encontrar la solución a problemas del tipo FJSSP.

4.4.1 Mecanismo de calendarización Parcial-S para SA

En [Xia, *et al.* 2005; Freitas y Aparecida, 2006; Aanen *et al.*,1993; Antami, 1995; Allahverdi, 1999] se reconocen dos tipos disposiciones de tiempo entre operaciones, uno conocido como tiempo de intercambio independiente y otro conocido como tiempo de intercambio dependiente. Estos tiempos de intercambio entre operaciones en una máquina se encuentran reflejados en una matriz NxN (Tablas 4-2, 4-3 y 4-4), donde el primer renglón de la tabla 4-2, corresponde a los tiempos de preparación en la máquina cero para recibir cada una de las operaciones a ser procesadas, la primer columna (operación cero) corresponde a los tiempos de limpieza después de procesar la última operación en dicha máquina (pudiendo ser cualquiera de las 9 operaciones, la que resultara

ser la última), la diagonal de la matriz se encuentra en ceros porque una operación no puede dejar una máquina para continuar en la misma, por lo que su disposición de tiempo es cero. Del segundo renglón en adelante, corresponde a cada una de las operaciones, por ejemplo para pasar de la operación 1 a la 3 en la máquina 0, de acuerdo a la misma tabla 4-2, se requieren 2 unidades de tiempo (disposición de tiempo=2). Los tiempos de intercambio entre operaciones son generados de manera aleatoria con un rango de valores de 1 a 3 para cada una de las máquinas.

Tabla 4-2 Disposición de tiempo para la máquina 0 para el FJSSP de 3x3

		Tiempos de Limpieza	Operación								
		0	1	2	3	4	5	6	7	8	9
Tiempos de Preparación	0	0	2	1	2	2	3	1	2	1	1
Operación	1	3	0	1	2	3	1	1	3	2	1
	2	2	1	0	1	1	2	2	1	2	3
	3	3	1	1	0	2	1	1	1	3	2
	4	3	1	2	1	0	1	1	2	1	1
	5	2	1	1	2	1	0	1	1	2	1
	6	1	1	1	2	1	3	0	1	2	1
	7	1	2	1	1	1	3	1	0	2	1
	8	3	1	2	1	2	3	2	1	0	1
	9	1	2	1	1	1	1	2	2	3	0

Las tablas 4-3 representan los datos requeridos para intercambiar operaciones en la máquina 1. Al igual que como sucedió con los datos de la máquina 0, las disposiciones de tiempo corresponden a los tiempos de preparación, intercambio y limpieza en dicha máquina.

Tabla 4-3 Disposición de tiempo para la máquina 1 para el FJSSP de 3x3

		Tiempos de Limpieza	Operación								
		0	1	2	3	4	5	6	7	8	9
Tiempos de Preparación	0	0	1	3	2	2	3	1	2	2	1
Operación	1	3	0	1	2	3	1	1	3	2	1
	2	2	1	0	2	1	2	2	1	2	3
	3	2	1	1	0	2	1	2	1	3	2
	4	3	3	2	1	0	1	1	2	1	1
	5	2	1	2	2	1	0	1	1	2	1
	6	1	1	1	2	1	3	0	1	2	1
	7	1	2	1	1	1	2	1	0	2	1
	8	3	1	1	1	2	3	1	1	0	1
	9	1	2	1	1	1	1	2	2	1	0

Y por último, en la tablas 4-4 se representan los datos requeridos para intercambiar operaciones en la máquina 2.

Tabla 4-4 Disposición de tiempo para la máquina 2 para el FJSSP de 3x3

		Tiempos de Limpieza	Operación								
		0	1	2	3	4	5	6	7	8	9
Tiempos de Preparación	0	0	1	1	2	2	3	1	2	1	1
Operación	1	3	0	3	2	3	1	1	3	2	1
	2	2	1	0	1	1	2	2	1	2	3
	3	1	1	2	0	2	1	1	1	3	2
	4	3	1	2	1	0	1	1	2	1	1
	5	2	3	1	1	1	0	1	1	1	1
	6	1	3	1	2	1	2	0	1	2	1
	7	1	2	1	2	1	3	1	0	2	1
	8	1	1	2	1	3	3	2	1	0	1
	9	1	3	1	1	1	1	2	1	3	0

El método para generar una calendarización es el utilizado en [Cruz-Chávez *et al*., 2007] aplicado al JSSP clásico, y adaptado en este trabajo para el FJSSP con Disposición de tiempo. Para resolver el problema de FJSSP, como cualquier otro de su tipo, aplicando el mecanismo propuesto de calendarización, es necesario comenzar con una solución inicial [Zalzala y Flemming, 1997], y la preparación al inicio de cada una de las máquinas (Disposición de tiempo) para recibir las operaciones que serán procesadas en ellas, por lo que las disposición de tiempo pueden variar de acuerdo a la operación que inicie los procesos en cada una de las máquinas. Al iniciar la calendarización de las operaciones en cada una de las máquinas, se toma en consideración, la información que se encuentra en la matriz NxN descrita anteriormente para cada una de las máquinas, para pasar de una operación a otra. Para lograr la primer configuración o calendarización, ésta selecciona de manera aleatoria cada una de las operaciones, y en principio, estas operaciones pertenecen al conjunto de operaciones que no tienen un predecesor (operación inicial en el mismo trabajo) o que en su defecto, son operaciones que tienen un predecesor pero que este ha sido ya asignado. El tiempo de inicio lo determinará el tiempo de término de su predecesor (si lo tiene, en caso contrario no tiene restricción de precedencia) más el tiempo que corresponde a la disposición de tiempo, considerando la disponibilidad de la máquina, tomando el de mayor valor para marcar el inicio de la operación seleccionada. El tiempo de término será la suma del tiempo de inicio más el tiempo de procesamiento asignado a la operación en turno, más la disposición de tiempo para pasar de una operación a otra en la misma máquina, lo anterior permite asegurar que se cumple con la restricción de precedencia y con la restricción de capacidad de recursos. De acuerdo a los datos presentados en la tabla 4-5, para una instancia de 3 trabajos y hasta tres máquinas, para cada una de las operaciones, se establece un número consecutivo a cada una de ellas que se encuentran dentro del proceso. De acuerdo a la información presentada, se aprecia

cada una de las máquinas que pueden procesar cada una de las operaciones y sus tiempos respectivos. En el caso de la operación 3, solamente existe una sola máquina que la puede procesar, máquina 0 con 6 tiempos. Caso contrario corresponde a la operación 8, la cual tiene 3 distintas máquinas que la pueden procesar, cada una de ellas con su tiempo respectivo.

Tabla 4-5 Instancia de un FJSSP de 3 trabajos y hasta 3 máquinas por operación

Operación	Job	Máquina/tiempo		
		Maq.0	Maq.1	Maq.2
1		6	*	5
2	1	2	4	*
3		6	*	*
4		4	6	5
5	2	6	*	6
6		*	5	3
7		2	3	*
8	3	4	2	6
9		5	4	*

La tabla 4-6 presentan los datos resultado de la primera solución generada, á cada una de las operaciones les fue asignado un turno, en el primer renglón se puede apreciar que a la operación 4, la cual corresponde a la primera operación del trabajo 2, le fue asignado el turno 1, la máquina seleccionada fue la máquina 0 con una duración (columna d) de 4 tiempos, y el tiempo de preparación (disposición de tiempo) en esta máquina para recibir la operación 4 fue de 2 unidades de tiempo (tabla 4-6). Esta operación no es intercambiable por la única razón de que es la primer operación asignada a la máquina 0, por lo que su valor de intercambio en la columna (In) es igual a cero, así mismo esta operación no tiene ningún predecesor al ser la primer operación del trabajo 2, por lo que en la columna de predecesor (Pr) es de igual forma 0 y en consecuencia el tiempo final del predecesor en la columna (Ep) es igual a cero. El tiempo de inicio de la operación 4(columna B) es de 3 y el tiempo de término (columna E) es 6 (considerando su disposición de tiempo). En la misma tabla 4-6 se observa que las operaciones susceptibles de ser intercambiadas son las operaciones 5, 7, y 2

(marcadas con un 1 en la columna In), ya que en la figura 1 se observa que estas operaciones no tienen tiempo de ocio con respecto a su operación que les precede en la misma máquina, y además dichas operaciones corresponden a distintos trabajos.

Tabla 4-6 Datos de la primera solución (S_0) con Disposiciones de tiempo

	Control de Procesos										
T	O	Job	Mach	Disposición de tiempo	d	In	Ao	Pr	Ep	Ini	E
1	4	2	0	2	4	0	0	0	0	3	6
2	1	1	2	1	5	0	0	0	0	2	6
3	5	2	2	1	6	1	1	4	6	8	13
4	7	3	0	2	2	1	4	0	0	9	10
5	8	3	1	2	2	0	5	7	10	11	12
6	2	1	0	1	2	1	7	1	6	12	13
7	9	3	1	1	4	0	0	8	12	14	17
8	3	1	0	1	6	0	0	2	13	15	20
9	6	2	2	1	3	0	0	5	13	15	17

Nomenclatura de tablas.

T=Turno asignado a la operación, orden ascendente.

O =Operación asignada en el turno T.

Job= Trabajo al que pertenece la operación asignada.

Mach= Máquina que procesa la operación con turno T.

Disposiciones de tiempo =Corresponde al tiempo asignado a disposiciones de tiempo

D=Duración de la operación en la máquina

In= Si es intercambiable la operación con alguna otra, esta es marcada con un 1, de lo contrario con 0.

Ao= Operación adyacente con la cual la operación en turno podría ser intercambiada.

Pr=Operación que precede a la operación asignada en el turno T.

EP= Tiempo de terminación de la operación Pr.

B= Tiempo de inicio de la operación del turno T (calendarización).

E= Tiempo final de la operación en el turno T.

En este caso las disposiciones de tiempo pueden corresponder a tres eventos diferentes: el tiempo asignado para preparar la máquina y pueda

recibir una operación, o puede corresponder al tiempo de intercambio entre una operación y otra, o en su defecto puede corresponder al tiempo de limpieza en la máquina cuando ésta termina de procesar la última operación.

En la misma tabla 4-6 se aprecia que el makespan de la solución S_0 es de 20 unidades de tiempo, dado que la operación 3 del trabajo 1 que se encuentra en el turno 8 (turno 8, operación 3, Trabajo 1) en la máquina 0 es la última en terminar con 20 unidades de tiempo, pero al adicionarle la disposición de tiempo que corresponde al tiempo de limpieza a cada una de las máquinas, observamos en la tabla 4-7, que corresponde a la información de disposición de tiempo de la máquina 0, el tiempo requerido para ir de la operación 3 a la 0, es de 3 unidades de tiempo, por lo que la máquina 0 termina en 23 unidades de tiempo.

Tabla 4-7 Disposición de tiempo de limpieza en la máquina 0 al terminar operación (3)

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

El mismo proceso se repite para la máquina 1, donde en la tabla 4-8 se aprecia que el tiempo de limpieza requerido al finalizar la última operación en ella, operación 9, es de 1 unidad de tiempo, por lo que su tiempo de termino final es de 18 unidades de tiempo.

Tabla 4-8 Disposición de tiempo de limpieza en la máquina 1 al terminar operación (9)

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

Y para la máquina 2, tabla 4-9, el tiempo requerido para limpieza al terminar la última operación, operación 6, es de 1 unidad de tiempo, por lo que el tiempo final en la máquina 2 es de 18 unidades.

Tabla 4-9 Disposición de tiempo de limpieza en la máquina 2 al terminar operación (6)

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

Por lo que el makespan es de 23 unidades de tiempo como se aprecia en la figura 4-5, donde se representa la solución. En la misma figura se aprecia que existen más de 3 operaciones que no tienen tiempo de ocio entre sí en las máquinas asignadas, estas no pueden considerarse candidatas para ser intercambiadas por el solo hecho de que dichas operaciones pertenecen al mismo trabajo (por lo que intercambiarlas equivale a violar restricciones de precedencia), como es el caso de la operación que se encuentra en el turno 6, número de operación 2, del trabajo 1 (6, 2, 1) y la operación que se encuentra en el

turno 8, número de operación 3 y trabajo 1, programadas ambas en la máquina 0.

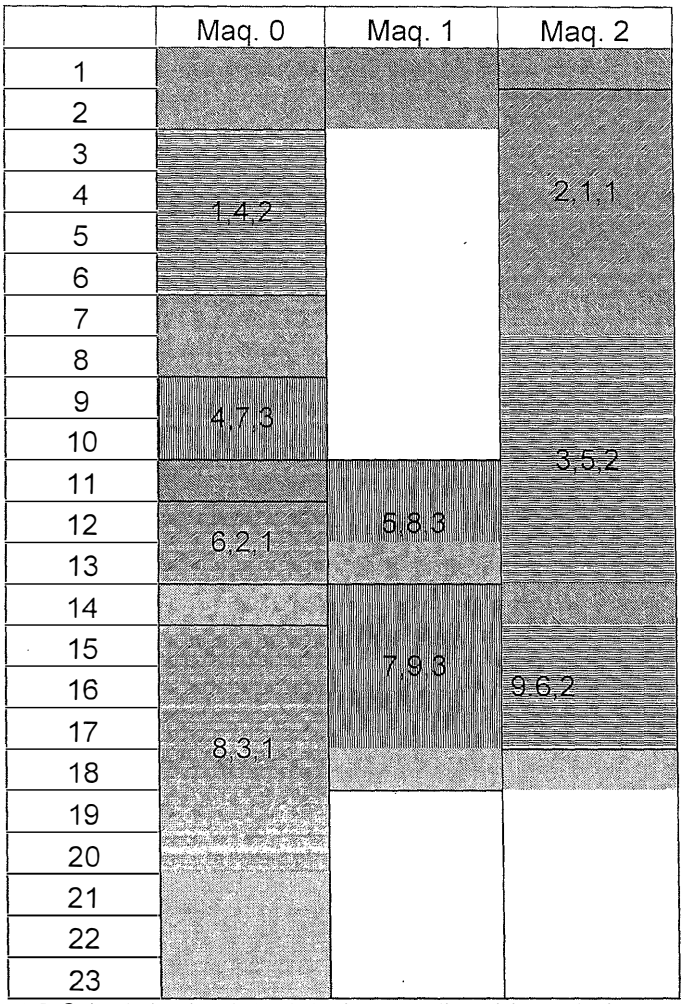

Figura 4-5 Calendarización de las Máquinas con disposiciones de tiempo en S_0.

Esta es la base para el algoritmo de recalendaración parcial, el identificar cuáles son las operaciones adyacentes en cada una de las máquinas y seleccionarlas como operaciones candidatas susceptibles de ser intercambiadas (perturbación del proceso) para generar una nueva solución mediante la misma función de vecindad propuesta en [Cruz-Chávez *et al.*, 2005] y utilizando el mismo mecanismo de calendarización utilizado en [Cruz-Chávez, *et al.*, 2007] para JSSP pero ahora con tres grandes variantes para ser aplicado al FJSSP. La primer variante es un

mecanismo que permite seleccionar una máquina de un conjunto de ellas para procesar la operación, la segunda variante, es un mecanismo que permite balancear las cargas de trabajo en las máquinas que integran el proceso y la tercer y última variante a considerar es tomar en cuenta las disposición de tiempo, mismo que puede corresponder a tres eventos diferentes: preparación de una máquina para recibir la primer operación, limpieza de la máquina o el tiempo de intercambio de una operación y otra.

4.4.2 Generación de un nuevo vecino con disposiciones de tiempo

La estrategia de intercambio es la siguiente: de la información generada para la solución S_0 seleccionar en forma aleatoria un par de operaciones que son susceptibles de ser intercambiadas, es decir que no tengan tiempo de ocio entre si en una máquina en particular y considerando la disposición de tiempo con la que cuente. Cuidando que el par de operaciones seleccionadas no correspondan al mismo trabajo, dado que no es posible intercambiar ese par de operaciones por que daría lugar a violar la restricción de precedencia. Al identificar el par de operaciones a ser intercambiadas, verificar que máquinas son objeto de ser seleccionadas (dado que existe más de una que puede procesar una operación) para cada una de las operaciones, para ello se requiere tomar en consideración la carga de trabajo de cada una de ellas (tomar la que tenga menor carga). Teniendo en claro qué máquina será seleccionada para procesar cada una de las operaciones, identificar su disposición de tiempo correspondiente para pasar de una operación a otra (descartando que la disposición de tiempo no corresponda a tiempos de preparación de inicio de una máquina y a tiempos de limpieza de la misma), teniendo estos elementos, proceder a calendarizar todo el proceso. De la figura 4-5, si se selecciona el par de operaciones 2 y 7 (dado que cumplen las condiciones antes descritas), en la tabla 4-10 se identifican la máquinas que pueden procesar dichas operaciones, donde se aprecia que para la operación 2 se puede seleccionar la máquina 0 con 2 unidades de tiempo y la máquina 1 con 4 unidades de tiempo. Mientras que para la operación

7 pueden usarse las mismas máquinas pero con distinto tiempo de procesamiento, para la máquina 0 se tienen 2 unidades de tiempo, mientras que para la máquina 1 se tienen 3 unidades de tiempo. Para ambas operaciones (2 y 7) seleccionar la máquina ideal (la que tenga menor carga de trabajo).

Tabla 4-10 Máquinas que pueden ser seleccionadas para la operación 2 y 7 respectivamente.

Operación	Job	Máquina/tiempo		
		Maq.0	Maq.1	Maq.2
1	1	6	*	5
2		2	4	*
3		6	*	*
4	2	4	6	5
5		6	*	6
6		*	5	3
7	3	2	3	*
8		4	2	6
9		5	4	*

Partiendo de la información que se observa en la figura 4-6, se puede apreciar que al intercambiar las operaciones 2 y 7 (la operación 7 cede su turno a la operación 2), por lo que da lugar a que se perturbe el Schedule a partir del tiempo 6 (columna etiquetada con Tiempo). Si se observa en la misma figura 4-6, la máquina 1 solamente tiene una programación que corresponde a una disposición de tiempo de 2 unidades, mientras que la otra opción, que corresponde a la máquina 0 tiene una tarea programada hasta el tiempo 6, más una disposición de tiempo de dos unidades. Por lo anterior, la máquina con menor carga de trabajo es la máquina 1.

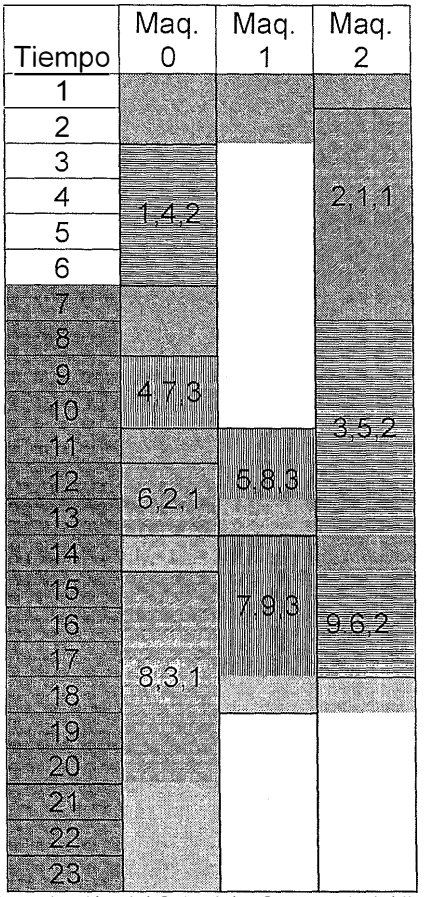

Figura 4-6 Perturbación del Schedule S_0 a partir del tiempo 7 al 23.

Al programarse la operación 2 en la máquina 1, la disposición de tiempo (tiempo de preparación) para recibir la operación cambiará, por que de acuerdo a la matriz de datos de NxN para la máquina 1, tabla 4-11, la disposición de tiempo requerido es de 3 unidades.

Tabla 4-11 Disposición de tiempo (tiempo de preparación) para recibir la operación 2 en la máquina 1

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

La operación 2 puede ser programada entonces a partir del cuarto tiempo, pero considerando la restricción de precedencia, se observa en la misma figura 2, que su operación precedente termina en 6 unidades de tiempo, por lo que podrá ser programada a partir del tiempo 7, como se aprecia en la nueva configuración que se representa en la figura 4-7. En la misma figura 4-6 descrita anteriormente, se puede ver que para programar la operación 7, puede elegirse entre dos posibilidades, dejarla en la máquina 0 o moverla a la máquina 1, ya que en apariencia ambas tienen la misma carga de trabajo, pero si se tiene en cuenta la disposición de tiempo para ambas máquinas encontramos que para pasar de la operación 4 a la operación 7 en la máquina 0, es de dos unidades de tiempo como se aprecia en la tabla 4-12.

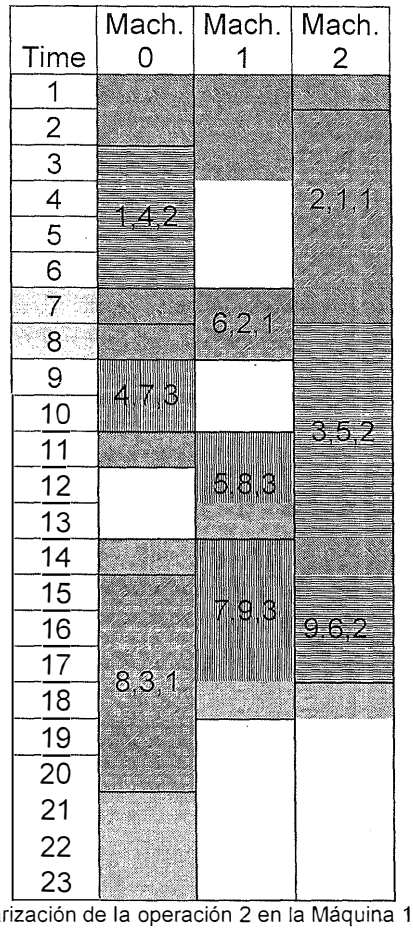

Figura 4-7 Calendarización de la operación 2 en la Máquina 1, del tiempo 7 al 8.

Tabla 4-12 Disposición de tiempo para pasar de la operación 4 a la 7 en la máquina 0

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

Mientras que para pasar de la operación 2 a la operación 7 en la máquina 1 es de 1 unidad de tiempo, como se puede apreciar en la tabla 4-13.

Tabla 4-13 Disposición de tiempo para pasar de la operación 2 a la operación 7 en la máquina 1

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

Por lo tanto si agregamos la disposición de tiempo a la máquina 1, el tiempo de inicio más pronto es de 10 unidades en la máquina 1, mientras que en la máquina 0 es de 9 unidades, por lo que la operación 7 se queda en la misma máquina que está actualmente, que es la máquina 0. Quedando la nueva programación para la solución S_1 como se aprecia en la figura 4-8, donde se calendariza el resto de las operaciones sin cambiar de máquina, cuidando las restricciones de precedencia y de capacidad de recursos.

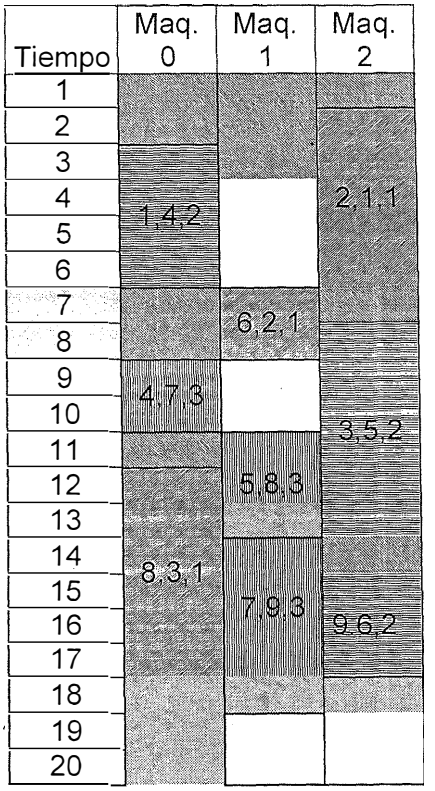

Figura 4-8 Nueva calendarización de la solución S_1.

Como se puede apreciar en la figura 4-8, el nuevo makespan para el problema que se describe es de 20 unidades de tiempo, y aún que las operaciones 3, 9 y 6 terminan al mismo tiempo (en 17 unidades de tiempo) en las 3 máquinas, al adicionarles la disposición de tiempo que corresponde al tiempo de limpieza 3, 1 y 2 unidades de tiempo para la máquina 0, 1 y 2 respectivamente, el makespan se incrementa, por lo que este es determinado por la máquina 0. Como se puede ver, al hacer el intercambio de operaciones en S_0, al inicio su makespan era de 23 unidades de tiempo, pero con el intercambio, la nueva solución S_1 baja a 20 unidades de tiempo.

La tabla 4-14 presenta el algoritmo de calendarización que aplica el mecanismo de calendarización parcial propuesto sin considerar el criterio de disposiciones de tiempo. El algoritmo es iniciado eligiendo una instancia del problema FJSSP. Después una solución inicial S0

(calendarización). NT es el número total de soluciones que se desean generar para obtener la calendarización y el makespan para cada solución. Con la función lista _ restringida (S_i), la lista de los pares de operaciones que pueden ser perturbadas en la solución es obtenida. La función turno _ perturbación (S_i), es elegido aleatoriamente el turno T que pertenece a la operación que será perturbada en la solución S_i. La función perturbación (S_i, posición, lista), genera una nueva S_j sobre la perturbación del par de operaciones del turno T en la solución S_i mediante la función de vecindad N1.

La función selecciona_máquina_trabajo(S_i,posición) selecciona una máquina del total de máquinas disponibles para cada operación a ser intercambiada, tomando en cuenta las máquinas que tengan una menor carga de trabajo. La función calendariza_MS (S_j, posición) obtiene la calendarización de la nueva solución S_j por calendarización. La calendarización inicial del turno T el cual es definido para la variable posición.

Tabla 4-14 Algoritmo de Calendarización (Parcial - S) sin disposición de tiempo

```
problema = Instancia de FJSSP;
S0 = solución_inicial (Problema);
for (i = 0, j = 1; j <= NT; i++, j++){
        list = lista_restringida(Si);
      posición = turno_perturbación(Si);
       Sj = perturbación(Si, posición, list);
     Mik=selecciona_máquina_trabajo(Si,posición);
       Sj = calendariza_MS (Sj, posición);
}
```

La tabla 4-15 presenta el algoritmo de calendarización que aplica el mecanismo de calendarización parcial propuesto pero ahora, este mismo se le incluye el mecanismo para tomar en cuenta las disposiciones de tiempo que corresponden a los tiempos de preparación, tiempos de intercambio entre operaciones y el tiempo de limpieza para cada una de

las máquinas que intervienen en el proceso para encontrar una mejor solución de acuerdo a la instancia seleccionada. Al igual que el algoritmo sin el criterio de disposición de tiempo, este es iniciado eligiendo una instancia del problema FJSSP. Cada una de las máquinas son inicializadas con una disposición de tiempo determinado por la primer operación a procesar, dado que este tiempo varia de una operación a otra y dados los criterios para asignar estos tiempos de espara, los tiempos de inicialización para cada máquina se determinan de forma aleatoria. Al igual que su algoritmo antecesor, para lograr una primera calendarización, las operaciones son asignadas de manera aleatoria, y los tiempos de intercambio entre una y otra en cada una de las máquinas también son determinados de manera aleatoria en un rango de 1 a 3 unidades. Al asignar el total de las operaciones, se asigna a cada una de las máquinas un tiempo de limpieza, mismo que podrá variar de acuerdo a la última operación procesada, para este caso, cada uno de los tiempos de limpieza son generados de manera aleatoria, siendo estos tiempos los que pueden determinar el makespan de la solución encontrada, ya que estos pueden determinar el tiempo último en el que todas las operaciones son procesadas. La solución inicial S0 (calendarización) es la solución inicial al proceso de búsqueda en el espacio de soluciones. NT es el número total de soluciones que se desean generar para obtener la calendarización y el makespan para cada solución. Con la función lista_ restringida (S_i), la lista de los pares de operaciones que pueden ser perturbadas en la solución es obtenida. La función turno_ perturbación (S_i), es elegido aleatoriamente el turno T que pertenece a la operación que será perturbada en la solución S_i. La función perturbación (S_i, posición, lista), genera una nueva S_j sobre la perturbación del par de operaciones del turno T en la solución S_i. La función Selecciona_Máquina_para_Carga(S_i, Posición) selecciona una máquina del total de máquinas disponibles para cada operación a ser intercambiada, tomando en cuenta las máquinas que tengan una menor carga de trabajo. La función calendarización_MS (S_j, posición) obtiene la

calendarización de la nueva solución S_j por calendarización. La calendarización inicial del turno T el cual es definido para la variable posición. En la calendarización de cada una de las operaciones que fueron perturbadas con el intercambio de un par de operaciones, y determinado el turno T a partir del cual será llevada a cabo la nueva calendarización, los tiempos de intercambio entre operaciones, los tiempos de preparación en las máquinas (cuando corresponda) y los tiempos de limpieza en cada una de ellas serán determinados en la función *Calendariza (S_j, position,* disposiciones de tiempo*);*

Tabla 4-15 Algoritmo de Calendarización (Parcial - S) con disposición de tiempo

```
Problema =Instancia de FJSSP;
problema = Instancia de FJSSP;
S0 = solución_inicial (Problema);
for (i = 0, j = 1; j <= NT; i++, j++){
        lista = lista_restringida(Si);
        posición = turno_perturbación(Si);
        disposición_de_tiempo=selecciona_tiempo(máquina,
                                                trabajo,operación)
        Sj = perturbación(Si, posición, list);
      Mik=selecciona_máquina_trabajo(Si,posición);
        Sj = calendariza_MS (Sj, posición);
}
```

4.5 Análisis de la complejidad del algoritmo de Recocido Simulado

La complejidad del algoritmo recocido simulado es impactado en gran medida por el algoritmo Parcial-S bajo el criterio de disposiciones de tiempo y sin disposiciones de tiempo. El anális realizado está basado en el número de operaciones para cada una de las instancias de prueba utilizadas como benchmarcks. Los resultados encontrados en el análisis de complejidad se observa que no existe diferencia en la complejidad de los algoritmos implementados, tanto para el que utiliza el criterio de disposiciones de tiempo como para el algoritmo que no lo usa. En ambos casos se reportan 3 posibles comportamientos de la complejidad de los algoritmos. El primer caso es cuando en una calendarización previa, se

requiera calendarizar, la mayor parte de las operaciones existentes en la instancia de prueba, producto de intercambiar una operación que se encuentre al inicio de todo el proceso, por lo que tendrían que calendarizarse a la mayoría de las operaciones involucradas. Por lo que el análisis para el peor caso es el siguiente:

$$T(n) = n + n + n(n+n) + n^2 + k(n) + n + n$$

$$T(n) = 4n + n^2 + n^2 + n^2 + k(n)$$

$$T(n) = 3n^2 + n(4+k)$$

Por lo que en el peor de los casos se tiene una complejidad de $O(3n^2)$

Un segundo caso, se presenta cuando del total de operaciones contenidas en una instancia de prueba, el algoritmo tiene que calendarizar aproximadamente el 50% del total de las operaciones, producto de intercambiar un par de operaciones que se encuentran en la mitad de la calendarización. Este análisis es el que se presenta a continuación:

$$T(n) = n + \frac{n}{2} + n(\frac{n}{2} + \frac{n}{2}) + \frac{n^2}{2} + k(n) + \frac{n}{2} + n$$

$$T(n) = 2n + \frac{n}{2} + \frac{3}{2}n^2 + k(n)$$

$$T(n) = \frac{3}{2}n^2 + n(5/2 + k)$$

$$T(n) = \frac{8}{2}(n^2 + n) + nk$$

Por lo que la complejidad para el caso promedio es de $O(0.5n^2)$

Y por último, un tercer caso, y el mejor de los tres, es cuando del total de las operaciones existentes en una instancia de prueba, se requiere calendarizar solamente algunas de las operaciones que se

encuentran al final de la calendarización previa, producto de intercambiar un par de operaciones.

$$T(n) = n+1+n(1+1)+\frac{1}{2}+k(n)+1+n$$

$$T(n) = 2n+2n+\frac{5}{2}+k(n)$$

$$T(n) = n(4+k)+\frac{5}{2}$$

Por lo que en el mejor de los casos, se tiene una complejidad $O(n)$.

De lo anterior podemos decir, que el comportamiento queda determinado por el peor caso $O(3n^2)$.

De manera integral, la complejidad del algoritmo de recocido simulado secuencial se puede calcular de la siguiente forma: Dado que el SA propuesto integra el algoritmo Parcial-S para la calendarización a fin de construir schedules y su respectiva evaluación, para el cálculo de la complejidad del SA se tomará en cuenta el peor caso, por lo que la función de complejidad para el SA trabajando en forma secuencial con el algoritmo de calendarización Parcial-S estará dada por $O(n^2+3n^2) = O(n^2(1+3))= O(4n^2)$. De acuerdo al comportamiento del SA, el análisis de la complejidad del algoritmo puede apreciarse en la tabla 4-16.

Tabla 4-16 Análisis de la complejidad de SA de acuerdo a su comportamiento

Caso	Complejidad
Mejor Caso	$O(n^2+n)=O(n\ (n+1))=O(n^2)$
Caso Promedio	$O(n^2+0.5n^2)=O(n^2(1+0.5))=O(1.5n^2)$
Peor Caso	$O(n^2+3n^2)=O(n^2(1+3))=O(4n^2)$

Capítulo 5

Algoritmo Paralelo de Recocido Simulado

Este capítulo describe la paralelización del algoritmo SA-A operando con el algoritmo de calendarización Parcial-S descrito en la sección 4.4. En la parte introductoria de este capítulo se describen las principales estrategias utilizadas para paralelizar el SA, incluyendo la estrategia MPSA propuesta por Sanvicente-Sánchez y Frausto-Solis [Sanvicente-Sánchez y Frausto-Solis, 2002] y utilizada como metodología para paralelizar el algoritmo de SA-A propuesto para FJSSP.

5.1 Introducción

Para la paralelización de un algoritmo, es necesario identificar dos aspectos muy importantes, el primero tiene que ver con la arquitectura del equipo a utilizar y el segundo aspecto tiene que ver con el ambiente de programación. En cuanto al primer aspecto, se sabe por la literatura [Authie *et al.,* 1994; Buyya, 1999; Culler *et al.*, 1999] que existen dos tipos de máquinas, las de memoria compartida, en donde cada uno de los procesadores que se integran en un equipo pueden accesar a la misma región de memoria y la comunicación entre tareas se hace através de operaciones de lectura y escritura en la misma memoria. En las máquinas de memoria distribuida, la memoria es distribuida entre los procesadores y cada uno de ellos puede direccionar su propia memoria, y la comunicación entre ellos se logra mediante el paso de mensajes en la red

que se conforma. Para este trabajo de investigación se utiliza una computadora cuya arquitectura es de memoria compartida. En cuanto a los ambientes de programación paralela, la arquitectura de la computadora tiene una gran influencia en el grado de paralelización o de la granularidad que se puede lograr en la aplicación, por ejemplo el radio entre el tiempo de computación y el tiempo de comunicación, para ello se requiere de establecer una estrategia para el desarrollo de programación paralela. Cung y otros [Cung et al., 2001] reconocen que existen varias estrategias de programación. Una de ellas consiste en usar lenguajes de programación paralela, los cuales son esencialmente lenguajes secuenciales aumentados con un conjunto de funciones especiales que hacen llamadas al sistema; estas llamadas proveen de primitivas de bajo nivel para el paso de mensajes, sincronización de procesos, creación de procesos, exclusión mutua, y otras funciones. Knies y otros [Knies et al.1994; Koelbel et al., 1994] citan a HPF (por sus siglas en inglés, High Performance Fortran) y OpenMP [Dagum and Menon, 1988], ambos lenguajes soportan el paralelismo de datos, es decir, el mismo conjunto de instrucciones puede ser aplicado a múltiples grupos de estructuras de datos. Una segunda estrategia de programación es utilizando librerías que permiten intercambiar mensajes entre las tareas, tales como PVM (Máquinas Virtuales Paralelas, por sus siglas en inglés) [Gueist et al. 1994] o MPI (Interfaces para el paso de mensajes) [Gropp et al., 1988; Foster, 1995] este tipo de lenguajes son utilizados generalmente en redes de computadoras o clúster de computadoras cuyas arquitecturas son de memoria distribuida, tanto HPF, OpenMP, PVM y MPI tienen instrucciones embebidas en lenguajes tales como C, C++ o Fortran. Por el tipo de paralelización que se pretende en el SA y el uso de la computadora donde se harán las experimentaciones, la cual es una IBM pSerie 690 (Regatta), cuya arquitectura es de memoria compartida, con 4 POWER CHIPS (32 procesadores), se utilizará el lenguaje de programación OpenMP, en ese ambiente de programación se identificaran cuatro elementos a medir, uno tiene que ver con el tiempo de corrida secuencial (TS) que es definido

como el tiempo que un algoritmo tarda en resolver una instancia de un problema en particular, en este sentido, cada algoritmo tiene su propio comportamiento para TS. Para el algoritmo que se propone en este trabajo de investigación, El comportamiento del algoritmo es el mismo, y su variación de TS está determinada por cada una de las instancias de prueba de los problemas de FJSSP. El segundo aspecto a considerar es el tiempo de corrida en paralelo (TP), el cual esta definido como el tiempo que toma el algoritmo en resolver una instancia del conjunto de pruebas o benchmarks sobre P procesadores (recurso computacional) por lo que TP es una función entre el hardware y el software (algoritmo propuesto junto con sus parámetros de sintonización). El tercer elemento a considerar es la velocidad o ganancia del algoritmo paralelizado (Speedup o S) del sistema en paralelo, definido precisamente como la ganancia computacional (S=Ts/Tp) usando P procesadores con respecto a uno sólo. Y por último, se tiene que considerar la eficiencia (E=S/P), que denota la utilización efectiva de los recursos computacionales, esto es el radio del Speedup con respecto al número de procesadores utilizados.

En cuanto a la metodología utilizada para la paralelización del algoritmo propuesto, se reconoce a partir de una revisión que se hace en [Sanvicente-Sanchez, et al, 1997, 1998] que los trabajos de diseño en paralelo del algoritmo de recocido simulado son agrupados de acuerdo a dos grandes enfoques.

- Psuedo-paralelización de SA: Algoritmos secuenciales de SA son ejecutados en diferentes procesadores, al mismo tiempo o la estructura de datos es asignada a diferentes procesadores donde un SA secuencial es ejecutado, y los métodos principales son: Partición de la estructura de Datos (PED), Recocidos Independientes Paralelos (RIP) y Cadenas de Markow Paralelas (CMP).
- Algoritmos de recocido simulado paralelo: El algoritmo SA se divide en tareas las cuales son distribuidas entre varios procesadores. Algunos de los algoritmos de recocido simulado paralelos son:

Cadenas de Markov Paralelas, Ensayos de Markov Paralelos, Recocido Simulado Adaptivo Paralelo, Árboles Especulativos y el Algoritmo sistólico.

Para paralelizar o psudo-paralelizar SA en la literatura existen diversos estudios de estrategias utilizadas, como por ejemplo la propuesta por [Greening et al, 1990] que hace uso de un método directo que utiliza las ventajas del hardware y ejecuta P ensayos en forma paralela y toma la mejor solución generada. Esta idea fue tomada por [Schwefel, 1981] y la denominó evolución estratégica -(1,p) que [Bertocchi y Sergi, 1992] retoman para paralelizar SA. Esta idea ha sido bastante explotada, y en la cual se asume que existen p procesadores que están disponibles y cada uno de ellos es capaz de generar su propio proceso de recocido simulado, lo que equivale a correr en forma secuencial múltiples procesos de SA y tomar la mejor solución del total de ellos. En [Der y Steinhofel, 2001] describen cómo a través de un sistema de cómputo distribuido, conformado por 12 computadoras personales, ejecutan un algoritmo de SA, dando un salto a los resultados mejor conocidos en su tiempo para JSSP, posteriormente estos mismos autores reportan una versión en paralelo del algoritmo de SA en el cual presenta un procedimiento para paralelizar el cálculo más pesado de su algoritmo, que es la obtención de la ruta crítica del grafo disyuntivo que utiliza. En [Aydin y Fogarty, 2004] se reporta la paralelización de un algoritmo de SA utilizando un sistema distribuido con 20 computadoras manejando el concepto de poblaciones de los algoritmos genéticos, pero sin utilizar ningún operador genético. En [Cruz-Chávez et al., 2005] se presenta una estrategia basada en aplicar cooperación dirigida hacia un espacio de búsqueda de buenas soluciones, y la forma en que se realiza es mediante la generación de un conjunto de SA con reinicio (CRS), donde cada vez que se termine un SA se aplique un cruzamiento con la solución obtenida de ese SA y la mejor solución obtenida por el CRS. En [Sanvicente-Sanchéz and Frausto-Solís, 2000; 2002] se presenta

una metodología para paralelizar algoritmos tipo recocido simulado denominada MPSA (por las siglas de Methodology to Parallelize Simualted Annealing) y se establecen los pasos para implementarla, para ello, partiendo de un estudio que los mismos autores hacen en [Sanvicente-Sanchéz 1997 y 1998] hacen notar lo siguiente:

- Dos ciclos anidados forman los algoritmos tipo recocido simulado. El ciclo externo o de temperaturas, establece un esquema de enfriamiento y el ciclo interno o constructor de una cadena de Markov, efectúa una caminata aleatoria sobre el espacio de soluciones.
- La implementación de los algoritmos tipo recocido simulado puede ser efectuada a través de cadenas de Markov homogéneas. Las cadenas de Markov individuales son construidas a través de un conjunto de ensayos donde una solución actual es transformada en una nueva solución a través de una función de generación y un criterio de aceptación (en esta propuesta de investigación, la función de generación utilizada es la propuesta presentada en [Cruz-Chávez, et al., 2007]. El criterio de aceptación está basado en la propuesta que se presenta en [Martínez-Rangel et al., 2007].
- Esquemas de enfriamiento conocidos garantizan la convergencia asintótica, sin embargo, la experimentación con algunas corridas efectuadas sobre estos esquemas pueden consumir tiempos impracticables, por lo que deben tratarse como algoritmos de aproximación. De esta manera, los parámetros del esquema de enfriamiento establecen un balance natural entre la eficiencia y la eficacia del algoritmo. En este sentido, la propuesta que se presenta en [Martínez-Rangel et al., 2007] favorece la convergencia del algoritmo de recocido simulado mediante la sintonización del parámetro de control de la temperatura mediante la desviación estándar obtenida a partir de los datos de entrada de cada una de las instancias de prueba utilizadas para medir la eficiencia y la eficacia del algoritmo propuesto en este trabajo de investigación.

- No existe una manera precisa para determinar la longitud L de la cadena de Markov. En general, L es determinada en la literatura a través de técnicas experimentales.
- El algoritmo de recocido simulado (SA) a través de su analogía en el proceso físico de recocido de sólidos es esencialmente un proceso secuencia y los algoritmos tipo recocido simulado por trabajar en el espíritu del algoritmo SA son también esencialmente secuenciales. Este hecho ha provocado que el obtener un algoritmo paralelo eficiente llegue a ser una tarea difícil.
- Los métodos de recocido simulado paralelos propuestos en la literatura, se centran principalmente en efectuar un trabajo distribuido del ciclo interno (distribuir el trabajo del algoritmo de Metrópolis o construir una cadena de Markov de manera distribuida o paralela) o en ejecutar varios recocidos simulados secuenciales (con cadenas de Markov recortadas) para obtener una mayor exploración del espacio de soluciones.
- El algoritmo Sistólico idealmente permite altos niveles de SpeedUp y un paralelismo masivo, pero éste deteriora la solución del algoritmo secuencial. El deterioro de la solución crece conforme el número de procesadores crece y puede llegar a ser significativa en problemas de tamaño medio.

Como se puede apreciar, en la metodología MPSA en cuyo kernel se efectúa una paralelización del ciclo de temperatura en algoritmos tipo SA, la paralelización del ciclo de temperaturas implica que cada procesador construya una cadena de Markov a su propia temperatura. Esto es, en lugar de que una nueva cadena de Markov a temperatura C_{k+1} sea comenzada al final de la cadena de Markov a la temperatura C_k todas las cadenas de Markov en el algoritmo que se esté paralelizando son comenzadas al mismo tiempo (Ver figura 5-1). Idealmente esto permite una paralelización masiva debido a que se tiene un procesador por cada temperatura del proceso de recocido y

con base en la convergencia asintótica se requeriría una secuencia infinita de temperaturas $\{c_k\}$ para alcanzar el óptimo del problema.

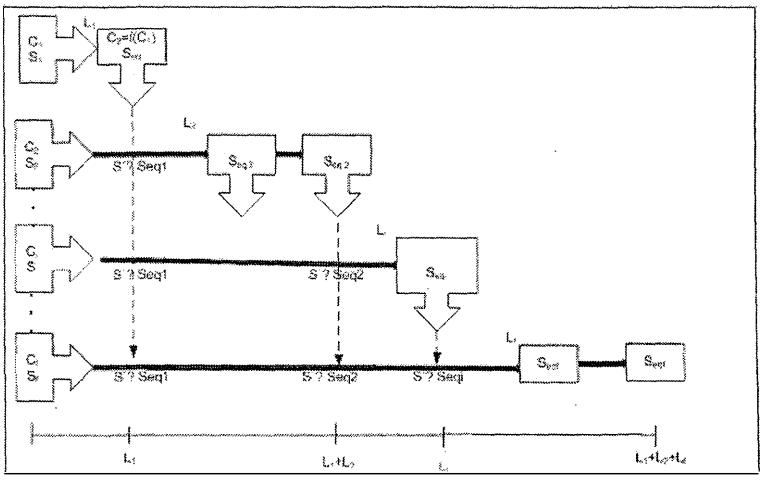

Figura 5-1 Esquema de interacción entre procesadores en el Kernel de MPSA

Para el Kernel de MPSA cada procesador envía su solución final después de generar una cadena de Markov de la longitud establecida, a todos los procesadores que están trabajando a temperaturas inferiores a la suya. Cuando un procesador recibe una solución, la evalúa a través de un criterio de aceptación o rechazo; si es aceptada el procesador reinicia la cadena de Markov que está construyendo con esta nueva solución, si ésta es rechazada continúa de manera normal.

5.2 Paralelización del algoritmo de SA con el algoritmo Parcial-S

Para este trabajo de investigación, se utilizó la metodología MPSA (ver Apéndice 1) para la paralelización del algoritmo de recocido simulado Acelerado (SA-A) ya que ésta establece que cualquier algoritmo tipo SA paralelo tiene el mismo esquema de enfriamiento que el algoritmo secuencial. Sin embargo, para llevar a cabo la paralelización del ciclo de temperaturas, por lo menos la temperatura inicial, C_1; la función de enfriamiento, $f(C_k)$, y la temperatura final, C_f, del esquema de enfriamiento deben ser conocidas con anterioridad a la implementación del algoritmo, lo que el algoritmo secuencial propuesto en este trabajo de investigación cumple de manera satisfactoria, ya que en [Martínez-Rangel, et al, 2007]

se propone la sintonización de la temperatura de SA, misma que se utilizará en MPSA.

En MPSA un procesador no puede dar por finalizado su trabajo si no hasta que se hayan terminado los procesos trabajando a una temperatura superior, pues requiere de las soluciones de éstos para saber si debe reiniciar y rehacer su cadena de Markov. En la figura 5-1 se observa que el procesador trabajando a temperatura C_2 reinició su cadena de Markov tomando ahora como solución inicial la enviada por el procesador trabajando a C_1, por lo que en lugar de terminar al tiempo TIEMPO(L_2) terminó al tiempo TIEMPO(L_1+L_2). En la figura 5-2 se presenta el esquema paralelizado del algoritmo de SA-A.

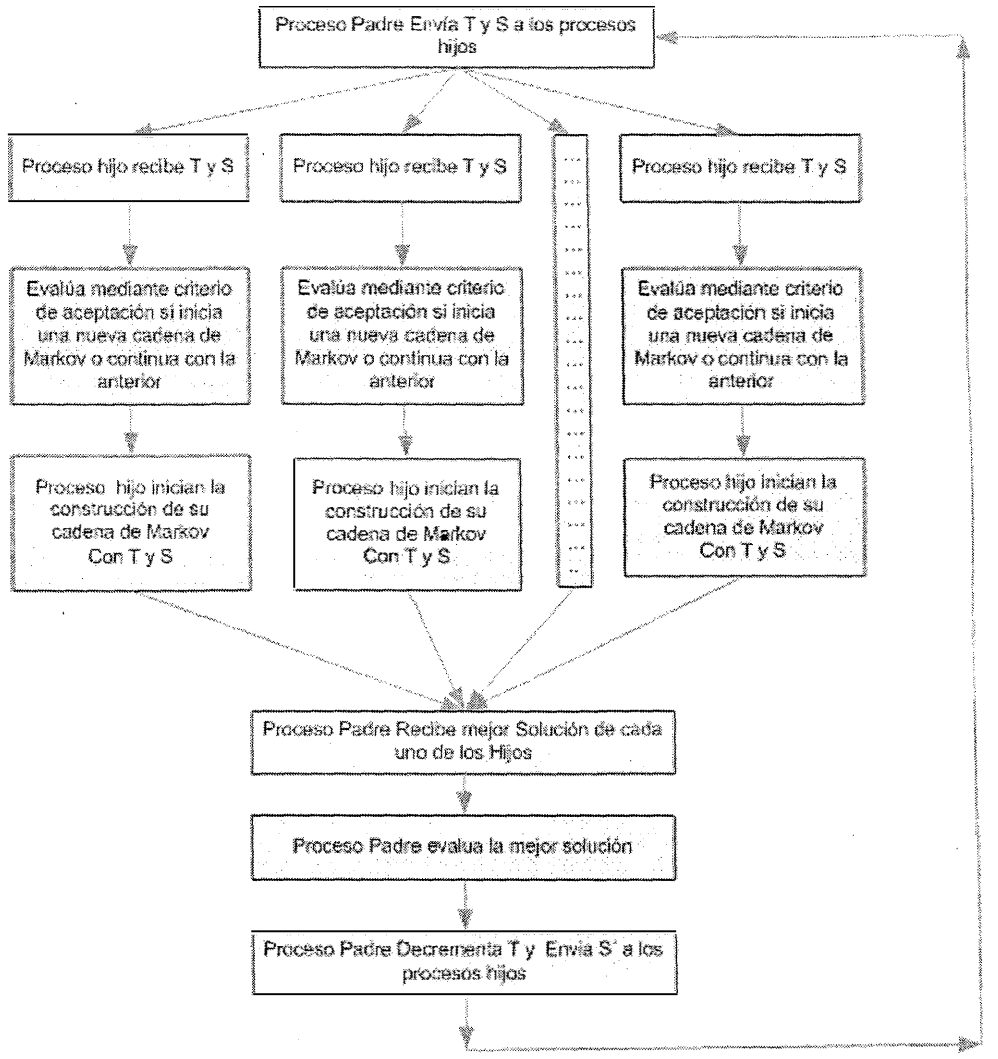

Figura 5-2 Esquema paralelizado del algoritmo SA.

La figura 5-3 presenta el algoritmo maestro (proceso P_0) que coordinará todo el proceso de SA en paralelo. El valor de la temperatura inicial, T_0 es igual a dos veces la desviación estándar ($2 * \sigma$) encontrada para el conjunto de soluciones generados mediante el algoritmo de calendarización descrito en la sección 4.4 y denominado Parcial-S en este trabajo de investigación. Del algoritmo de recocido simulado, lo que no se paraleliza es el algoritmo de calendarización parcial (Parcial-S), el cual corre de manera secuencial en cada uno de los procesadores.

Proceso P_0

1. Datos de entrada: S, σ ,MakeSpan
2. Resultado: Makespan
3. T=2 * σ
4. Tamaño_Vecindad=n*(n-1)/2;
5. Mejor_sol=S;
6. Envia_Mejor_sol(Mejor_sol) y T a los procesos P_j, j=1,2,..,p;
7. Sol_final=S;
8. Mientras (T>0)
9. Recibe Mejor_sol(S´) de P_j, j=1,2..p
10. if (S´<=Mejor_sol) entonces{
11. Mejor_sol=S´;
12. }
13. $T \leftarrow \gamma * T$
14. Envia_Mejor_sol(Mejor_sol) y T a los procesos P_j, j=1,2,..,p; }
15. Fin de Mientras
16. MakeSpan=Mejor_sol;

Figura 5.3 Proceso Maestro del SA-A paralelizado

La figura 5-4 presenta el algoritmo de Metrópolis en forma paralela que forma parte del SA-A para cada uno de los procesos hijos que intervienen en la construcción de las cadenas de Markov de acuerdo a MPSA. En estos procesos, el decremento de la temperatura esta dado por : $T \leftarrow \gamma * T$,donde γ=0.998, valor obtenido por un análisis de sensibilidad para el SA secuencial. Una solución S´ del problema podrá ser evaluada para ser aceptada o rechazada mediante la función de distribución de probabilidades de Boltzmann, siempre y cuando la

diferencia de energía (Δs) no sea mayor a dos veces la desviación estándar encontrada para la vecindad (límite superior y límite inferior). Para evaluar la solución S´, se utiliza el algoritmo de calendarización S-Parcial para evaluar las soluciones, lo que permitirá acelerar las búsquedas locales en SA-A paralelizado.

```
Procesos Pj, j=1,2,..,p;i=proceso actual;
17. Recibe sol_inicial de P0 (S)
18. N_vecindad←0;
19. MakeSpan=10000;
20. Mientras (N_vecindad < Tamaño_vecidad ){
21.   Genera_nueva_sol S´∈ N (S) ;
22.       If (S´<= S)
23.       {
24.           S=S´ ;
25.           if (Makespan > S)  Makespan←S;
26.           N_ vecindad++;
27.        } //end if
28.      else
29.       {
30.          ΔS ←S´-S
31.          if (ΔS <= 2 *σ) {
32.      n←(0<probabilidad<1)
33.              if (n<exp (-ΔS /T)))  Then   {
34.                  S ← S´ //acepta nueva sol
35.                   N_ vecindad++;
36.                   }
37.               else
38.                   S←S;  //rechaza nueva sol
39.            }//end if
40.           else
41.                S←S;  //rechaza nueva sol
```

```
42.          } //end if DeltaS <=
43.          else
44.             S←S;   //rechaza nueva sol
45.          }
46.      recibe_mejor sol de Pj (Sbest),j=1,2,..,p (j<>i)
47.      if (Sbest < MakeSpan)
48.       {
49.          S=Sbest;
50.          MakeSpan=Sbest
51.       }
52.     } //fin de mientras
53. envia_mejor_sol a P0(MakeSpan)
```

Figura 5.4 Algoritmo de Metrópolis paralelizado

5.3 Análisis de la complejidad del algoritmo SA-A paralelizado

Uno de los principales inconvenientes de los algoritmos metaheurísticos en ambientes paralelos es su complejidad computacional. En algunos casos, para algunos algoritmos el tipo de implementación que se haga con ellos les favorece, por ejemplo, para los algoritmos genéticos resulta adecuada una implementación paralela global a fin de mejorar el rendimiento y reducir el costo de cálculo. En el caso de los algoritmos tipo SA, el tipo de implementación no es tan importante a fin de que mejore su rendimiento y el costo de cálculo sea menor, ya que SA es un algoritmo totalmente estocástico, donde el tiempo para la construcción de las cadenas de Markov puede variar entre un procesador y otro, ya que no es posible calcular el tamaño de la cadena para cada uno de ellos, pues dependerá en mucho del mecanismo de discriminación para la aceptación o rechazo de una solución. En este algoritmo de recocido simulado acelerado paralelizado (SA-A paralelizado), se propone que la aceptación o rechazo de una solución pase por tres filtros, el primero de ellos es

mediante la propuesta hecha en [Martínez-Rangel, et al., 2007] donde una solución entra a evaluación siempre y cuando la diferencia de energía entre una solución y otra esté en el rango de dos veces la desviación estándar generada para una población de 65,000 soluciones. El segundo filtro tiene que ver con la función de distribución de la función de Boltzmann [Kirkpatrick et al., 1983] dentro del algoritmo de Metrópolis y por último, de acuerdo a la metodología MPSA para la paralelización del ciclo de temperatura, un proceso podrá generar una cadena de Markov tan larga como se lo permitan los demás procesos que corren a temperaturas inferiores a la suya, ya que dependerá del criterio de discriminación si una solución es aceptada para iniciar una nueva cadena de Markov, o se continúa con la actual.

Por lo anterior, resulta complicado evaluar la complejidad del algoritmo SA-A paralelizado. Un primer acercamiento para evaluar la complejidad del SA-A paralelizado tiene que ver con la forma en que este fue implementeado. De acuerdo a la estrategia de paralelización, se pueden distinguir dos posibles comportamientos en el algoritmo de Metrópolis al construir una cadena de Markow en entornos paralelos: en el peor de los casos, es cuando se tiene que reiniciar la secuencia de una cadena de Markov y el mejor de los casos, cuando se continua construyendo la cadena actual.

De manera integral, la complejidad del SA paralelizado se puede calcular de la siguiente forma: Dado que el SA propuesto integra el algoritmo Parcial-S para la calendarizacióna fin de construir schedules y su respectiva evaluación, se tiene que la complejidad de Parcial-S tiene tres posibles comportamientos como se ve en la sección 4.3, el peor caso, el caso intermedio y el mejor caso, cuyas complejidades son $O(3n^2)$, $O(0.5n^2)$ y $O(n)$ respectivamente. Para el cálculo de la complejidad del SA se tomará en cuenta el peor caso, es decir $O(3n^2)$, por lo que la función de complejidad para el SA-A trabajando en forma secuencial con el algoritmo de calendarización Parcial-S estará dada por $O(n^2+3n^2) = O(n2(1+3))= O(4n^2)$. En entornos paralelos, la función de complejidad

estará en función del número de procesadores (p) utilizados para el algoritmo paralelizado por lo que, la complejidad del SA paralelizado estará dada por la función $O(4n^2) * \frac{1}{\log_p}$.

Lo anterior se justifica por que a medida que se aumenta el número de procesadores que participan en la ejecución del SA-A paralelizado, se observaran tiempos menores o iguales al tiempo consumido por su contraparte el algoritmo SA-A secuencial. Los tiempos menores se obtendrán hasta llegar a un límite llamado "total de paralelismo posible", punto a partir del cual aún cuando se siga aumentando el número de procesadores ya no se ganará tiempo de ejecución de los procesos. Esto se debe a la Ley de amdhal la cual se expresa de la siguiente manera [Sanches, 1997]:

$$SpeedUp \leq \frac{p}{1+(p-1)\alpha}$$

Donde α es igual al número de operaciones secuenciales entre el total de operaciones a procesar, por lo que la aceleración de un algoritmo paralelizado está límitado por este factor, lo que hace que la aceleración de un algoritmo tal como SA-A paralelizado no sea lineal (lo que resulta ideal en un algoritmo paralelizado, pero que no es posible debido a los retardos ocasionados por comunicaciones y sincronización entre procesadores). Mientras que p equivale al número de procesadores que participan en la ejecución del SA-A paralelizado.

Capítulo 6

Resultados experimentales de Recocido Simulado

Este capítulo presenta los resutados obtenidos con el algoritmo propuesto y se muestra la eficiencia y eficacia de ambos algoritmos trabajando de manera independiente bajo entornos secuencial y paralelo respectivamente. Así como también se muestran los resultados de la convergencia del algoritmo y del mecanismo de calendarización parcial (Parcial-S) utilizado en el algoritmo propuesto.

6.1 Introducción

A fin de probar la eficiencia y eficacia tanto del Algoritmo de Recocido Acelerado Secuencial (SA-A secuencial) como del algoritmo de Recocido Acelerado Paralelizado (SA-A paralelizado) se utilizará el mismo conjunto de benckmarks para FJSSP parcial, propuestos por Bradimarte [Brandimarte, 1993]. Las pruebas que se reportan en este capítulo se describen en dos vertientes: La primera para mostrar la eficiencia y la eficacia del algoritmo de recocido simulado acelerado secuencial (SA-A secuencial) sin disposiciones de tiempo y bajo criterios de disposiciones de tiempo. Y la segunda vertiente, nos permite mostrar la eficiencia y la eficacia del algoritmo de recocido simulado acelerado paralelizado. Para ambos algoritmos, tanto para el secuencial como para el paralelo, los

datos utilizados para la sintonización de la temperatura inicial, asi como para el criterio de aceptación o rechazo en el algoritmo de Metrópolis dentro de ambos algoritmos de recocido simulado, es la desviación estándar encontrada para una población de 65,000 elementos para el conjunto Ω, de acuerdo a la propuesta hecha en [Martínez-Rangel et al, 2007].

6.2 Resultados experimentales de la convergencia de SA secuencial

El algoritmo propuesto de SA secuencial fue implementado en lenguaje C en una PC con 2 Ghz, y 1 Gb en RAM. La estrategia utilizada para la sintonización de los parámetros de control del algoritmo de SA es la descrita en la sección 4.3 de este trabajo de investigación, y el algoritmo utilizado para la calendarización dentro de SA es el descrito en la sección 4.4 de este mismo trabajo. Para probar la eficiencia del mecanismo propuesto se usó un conjunto benckmarks para FJSSP de tamaño pequeño, mediano y grande (Mk01, Mk02, Mk03 y Mk07) propuestas en [Brandimarte, 1993]. En la tabla 6-1 se muestran los valores de la desviación estándar que se generaron para las instancias de prueba. El conjunto Ω se compone de 65,000 soluciones para cada una de las intancias utilizadas como prueba.

Tabla 6-1 Desviación Estándar Obtenida para Ω = 65000

instancia	Trab/Maq/Op	Desv. St
mk01	10/6/55	11.9
mk02	10/6/58	10.54
mk04	15/8/90	16.58
mk07	20/5/100	36.79
mk05	15/4/106	22.43
mk03	15/8/150	47.30
mk06	10/15/150	23.87
mk08	20/10/225	26.34
mk09	20/10/240	32.49
mk10	20/15/240	28.45

En la figura 6-1 se muestra el grado de dispersión de las soluciones para una instancia en particular al aplicar el algoritmo propuesto de SA-A

junto con el algoritmo de calendarización, Parcial-S. Para tal fin, se utilizó una instancia considerada dentro de la literatura como una instancia de tamaño grande, la instancia mk07. Si a la media lograda se le agregan más o menos dos veces la desviación estándar, entonces estaremos aceptando el 75% de las soluciones generadas en el proceso de SA-A, que como se puede ver, el patrón de distribución que sigue ΔE_k es similar a una distribución de Poison, donde el mayor número de soluciones óptimas es cuando ΔE_k se aproxima a cero (diferencia de energía entre S´y S es mínima), por lo que solamente se aceptan soluciones buenas cuando T_0 es mínima (se aproxima a cero).

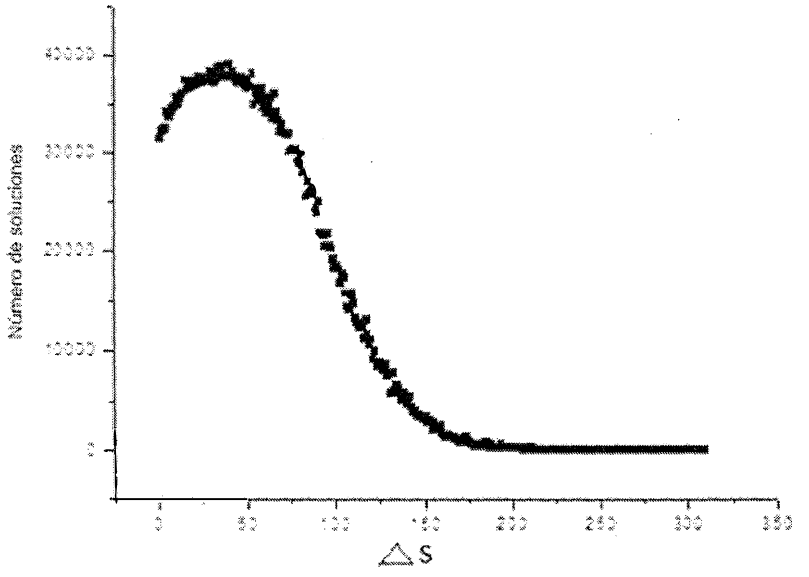

Figura 6-1 Dispersión de la vecindad generada para el problema Mk07 (20x5) de FJSSP

Al acotar el 75% de todas las soluciones dentro del espacio de búsqueda de soluciones factibles y sintonizando la temperatura inicial $T_0=2 * \sigma$, nos permite acelerar la convergencia hacia un Óptimo global, al tener un proceso de SA-A controlado, lo anterior se puede apreciar en la figura 6-2, donde el proceso controlado de SA-A converge a un mejor makespan con 41 unidades para la instancia Mk01, mientras que el proceso no controlado para la misma instancia en el mismo tiempo logra un makespan con 52 unidades, lo que en términos reales el proceso controlado de SA-A tiene un 26% más de rendimiento que el proceso no

controlado, el mismo fenómeno fue observado en el resto de las instancias con las que se probó el algoritmo propuesto.

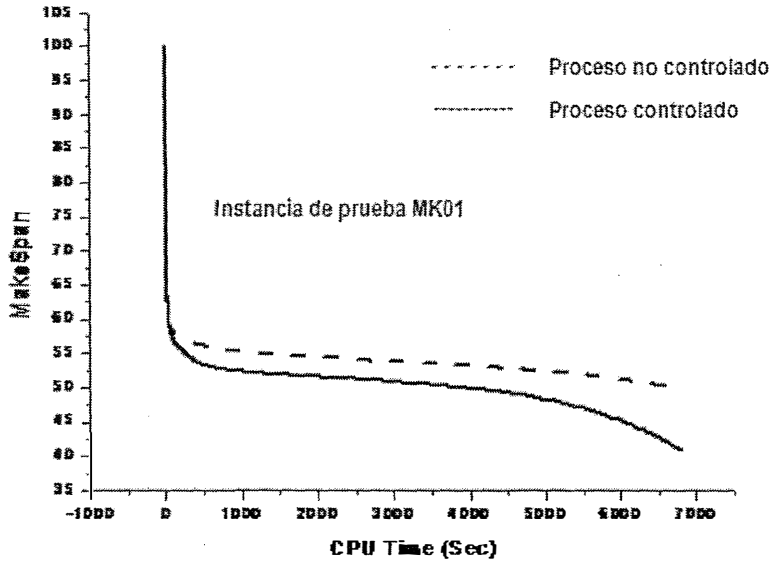

Figura 6-2 Comparación de resultados para el problema Mk01

6.3 Resultados experimentales del Algoritmo parcial-S para SA en forma secuencial

El algoritmo de calendarización (Parcial-S) integrado al algoritmo de recocido simulado propuesto para FJSSP se implementó bajo dos criterios, el primero sin considerar disposiciones de tiempo, y la segundo criterio, considerando disposiciones de tiempo. El algoritmo Parcial-S fue implementado en lenguaje C en una PC con 2 Ghz, y 1 Gb en RAM. El conjunto benckmarks utilizado para FJSSP parcial son los que presentan mayor dificultad para ser tratados dentro del grupo de problemas de FJSSP [Ho *et al.,* 2004; Kacem *et al.*, 2003; Kacem *et al.*, 2002; Zribi *et al.*, 2004; Tamaki *et al.*, 2001; Xian *et al.*, 2005]. Este grupo de problemas fueron propuestos por Brandimarte en [Brandimarte, 1993], ver tabla 6-2, donde se refleja para cada una de las instancias el número de trabajos, el número de máquinas y el total de operaciones a procesar para cada una de ellas.

Tabla 6-2 Benchmarks de Brandimarte[Brandimarte, *et al.*, 1993]

benchmarks	Job/Mach/Op
mk01	10/6/55
mk02	10/6/58
mk04	15/8/90
mk07	20/5/100
mk05	15/4/106
mk03	15/8/150
mk06	10/15/150
mk08	20/10/225
mk09	20/10/240
mk10	20/15/240

Los resultados que aquí se presentan corresponden a problemas generales de FJSSP que permiten mostrar la eficiecia y la eficacia para generar vecindades y su respectiva evaluación de cada una de las soluciones encontradas que forman parte de una vecindad.

Para comparar la eficiencia y la eficacia del algoritmo Parcial-S propuesto, se usaron dos estrategias en la generación de vecindades para cada una de las instancias de prueba de la tabla 6-2. La primera estrategia es llamada Total-S el cual es utilizado por el procedimiento clásico de calendarización propuesto en [Zalzala y Fleming, 1997] para evaluar la calidad de solución y requiere obtener la calendarización para todas las operaciones para problemas de este tipo. La estrategia Tota-Sl es utilizada en varios algoritmos presentados en la literatura. La segunda estrategia, llamada Parcial-S (mecanismo propuesto y descrito en 4.4), logra una calendarización para solo una parte de las operaciones implicadas en la instancia del problema para evaluar la calidad de la solución.

La experimentación se llevó a cabo mediante la generación de una vecindad compuesta de 65,000 elementos para cada una de las instancias utilizando ambas estrategias, El algoritmo Total-S y el algoritmo Parcial-S. En la tabla 6-3 se presentan los resultados encontrados, y los datos utilizados en cada una de las instancias de prueba tales como el número de trabajos, el número de máquinas, el número de operaciones y el tamaño de la vecindad dentro del ciclo de Metrópolis.

Tabla 6-3 Resultados encontrados para Ω = 65000

Problema.	Vecindad (n*(n-1))*2	Tiempo (Seg)		Trabajos	Máquinas	Num.Operaciones
		Parcial-S	Total-S			
mk01	100.00	151.75	1275	10	6	55
mk02	100.00	203.59	2575	10	6	58
mk04	210.00	207.74	2965	15	8	90
mk07	160.00	185.85	1990	20	5	100
mk05	90.00	177.79	2355	15	4	106
mk03	210.00	187.78	1405	15	8	150
mk06	280.00	139.62	2185	10	15	150
mk08	360.00	192.78	2575	20	10	225
mk09	360.00	226.24	2290	20	10	240
mk10	560.00	178.95	1185	20	15	240

La figura 6-3 muestra claramente como Parcial-S requiere menos tiempo para generar el mismo número de soluciones que Total-S para el conjunto de pruebas utilizado, el orden en que se presentan las instancias en esta figura es de acuerdo al número de operaciones involucradas en cada una de ellas.

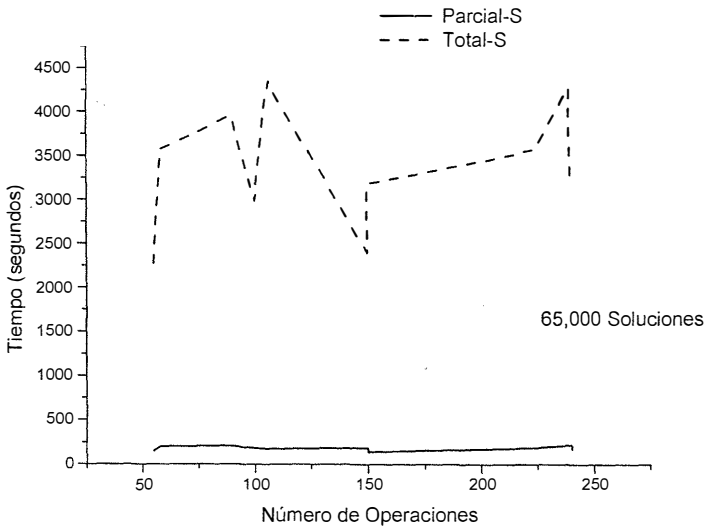

Figura 6-3 Eficiencia del Parcial-S Vs Total-S para el FJSSP

Para mostrar la eficiencia y la eficacia del algoritmo con problemas de FJSSP bajo consideraciones de disposiciones de tiempo, se utilizó el mismo grupo de problemas de la tabla 6-3, propuestos por Brandimarte, las consideraciones de disposiciones de tiempo fueron definidas para

cada una de las máquinas que intervienen en el proceso en cada una de las instancias utilizadas de la siguiente manera: las disposiciones de tiempo, como se explicó en la sección 4.4 de este trabajo de investigación. Estas disposiciones de tiempo son representados en una matriz de NxN, figura 6-4 donde el renglón representado por 0,1,2,..,n corresponde a los tiempos de preparación en una máquina en particular para recibir a cada una de las operaciones que podría procesar, ya que cada una de ellas requiere un tiempo distinto de disposiciones de tiempo, y la columna representada por los valores 0,1,2,..,n corresponde a los tiempos de limpieza que se requieren para cada una de las operaciones involucradas en el proceso, y el valor seleccionado dependerá de cual sea la última operación procesada en dicha máquina. La diagonal representada por ceros, indica que en una máquina no existe una disposicion de tiempo, dado que no se permiten interrupciones al procesar una operación y no existe la posibilidad de procesar una operación dos veces. Los valores representados con x corresponden a los tiempos de intercambio entre operaciones. Cada uno de los valores para las disposiciones de tiempo (preparación, intercambio entre operaciones y tiempos de limpieza) corresponden a valores enteros generados de manera aleatoria en un intervalo de 1 a 3 para cada una de las máquinas, en cada una de las instancias de prueba.

	Operaciones				
Operaciones	0	1	2	...	n
	1	0	x	...	x
	2	x	0	x	x
	..	x	x	0	..
	n	x	...	x	0

Figura 6-4 Consideraciones de disposición de tiempo para una máquina en particular

Para cada una de las instancias de FJSSP con disposiciones de tiempo se generaron vecindades de 65,000 elementos al igual que para las instancias para FJSSP sin disposiciones de tiempo. En la figura 6-5 se muestra una comparativa de los resultados para FJSSP bajo criterios de disposiciones de tiempo Vs FJSSP sin disposiciones de tiempo utilizando

el algoritmo Parcial-S, y lo que se muestra es que el algoritmo resulta eficiente para FJSSP con ambos criterios.

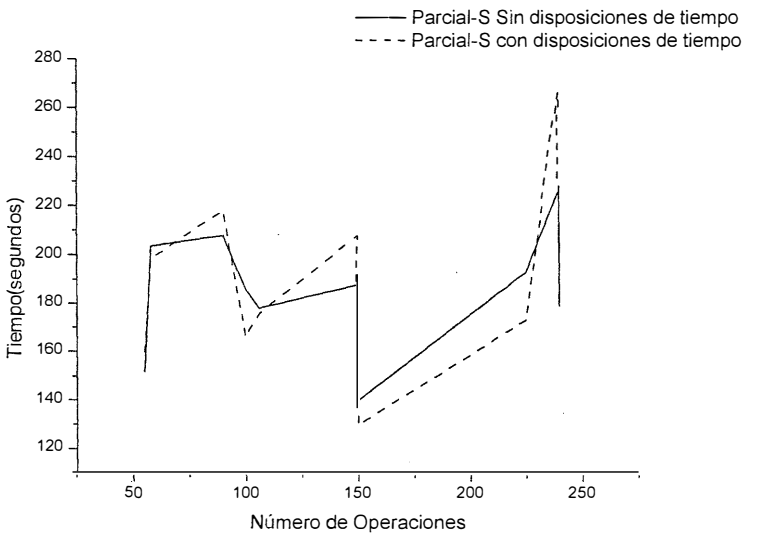

Figura 6-5 Algoritmo Parcial-S para FJSSP sin disposiciones de tiempo Vs FJSSP con disposiciones de tiempo

Como se muestra en la misma figura 6-5, para algunas instancias de FJSSP con criterios de disposiciones de tiempo, el algoritmo Parcial-S mejora en tiempo como sucede por ejemplo con la instancia Mk02 donde el tiempo requerido para generar la vecindad de 65,000 soluciones fue de 203.59 segundos sin considerar la disposición de tiempo, y 198.59 segundos para la misma instancia pero bajo consideraciones de disposiciones de tiempo. Este mismo fenómeno se repite para las instancias Mk04, Mk07 y Mk08, donde el algoritmo Parcial-S resulta mejor bajo criterios de disposiciones de tiempo.

6.4 Resultados experimentales para mostrar la eficiencia y eficacia del algoritmo SA Secuencial

El algoritmo propuesto de recocido simulado fue implementado en forma secuencial utilizando el lenguaje C en una PC con 2 Ghz, y 1 Gb en RAM.

Las instancias utilizadas son de tamaño pequeño, mediano y grande(Mk01, Mk02, Mk03 y Mk07).

En términos generales, en la tabla 6-4 se presentan los resultados obtenidos para las cuatro instancias de prueba ya mencionadas. Los resultados que se presentan muestran que son iguales o en el mejor de los casos superiores a los resultados reportados en la literatura. Para el problema Mk01, que corresponde a un problema de 10 trabajos y 6 máquinas, el error relativo fue de 2.4% respecto al mejor conocido que es de 40 unidades de tiempo, y siendo mejor al menos en uno de los casos reportados. En cuanto al problema Mk02, que corresponde a un problema de 10 trabajos y 6 máquinas, el error relativo fue de 7.1 con respecto al mejor conocido, pero mucho mejor que 3 de los 5 casos que se reportan en este documento. En cuanto al Mk03 que corresponde a un problema de 15 trabajos y 8 máquinas, el error relativo fue de 5.5 % respecto al óptimo reportado en la literatura, no pudiéndose comparar con el resto de los autores, por no presentar un resultado para esta instancia en particular. Y por último en cuanto al Mk07 que corresponde a un problema de 20 máquinas y 5 trabajos y considerado como uno de los más difíciles de tratar del grupo que se analiza, el error relativo fue de 2.0%, siendo el resultado mejor que dos de los 4 casos reportados.

Tabla 6-4 Resultados para 4 instancias de prueba para Flexible-JSSP, usando SA Controlado

Bench marks	Trabajos/ Máq/ Op	UB-LB	ER	SA Controlado	Caso A	Caso B	Caso C	Caso D	Caso E
mk01	10/6/55	36-40	0.00	41	40	42	41	*	*
mk02	10/6/58	24-26	0.00	28	26	32	29	32	28
mk03	15/8/150	204	0.00	216	*204	*	*	*	*
mk07	20/5/100	133-144	0.00	147	144	157	148	*	*

Caso A: Resultados reportados por Mastrolilli y otros [Mastrolilli *Et al.*, 1998]
Caso B: Resultados reportados por Brandimarte y otros [Brandimarte *et al.*, 1993]
Caso C: Resultados reportados por Ho y otros [Ho *et al.*, 2004]
Caso D: Resultados reportados por Kacem y otros [Kacem *et al.*, 2002]
Caso E: Resultados reportados por Neurodine y otros [Neurodinne *et al.*, 2007]

Para el caso que corresponde a FJSSP bajo consideraciones de disposiciones de tiempo, se utilizó el mismo grupo de problemas de prueba de Brandimarte, referidos en la tabla 6-2. Los criterios de disposiciones de tiempo fueron definidos para cada una de las máquinas que intervienen en el proceso en cada una de las instancias utilizadas como prueba tal como se explicó en las secciones 4.4 de este trabajo de investigación. Cada uno de los valores para disposiciones de tiempo (preparación, intercambio entre operaciones y tiempos de limpieza) corresponden a valores enteros generados de manera aleatoria en un intervalo de 1 a 3 para cada una de las máquinas, en cada una de las instancias de prueba.

Para los resultados que se presentan en la tabla 6-5, bajo consideraciones de disposiciones de tiempo, no fue posible compararlos con otros trabajos, dado que para el grupo de instancias seleccionadas no existen reportes en la literatura bajo este criterio, por lo que las comparaciones que se presentan corresponde a los resultados encontrados utilizando el mecanismo de SA secuencial controlado sin disposiciones de tiempo Vs SA-A secuencial controlado con disposiciones de tiempo para las cuatro instancias de prueba seleccionadas.

Para el problema Mk01, que corresponde a un problema de 10 trabajos y 6 máquinas, el error relativo fue de 4.4% respecto al valor encontrado de 41 unidades de tiempo en el proceso de SA controlado sin disposiciones de tiempo vs el valor encontrado de 43 unidades en el proceso de SA controlado bajo disposiciones de tiempo. En cuanto al problema Mk02, que corresponde a un problema de 10 trabajos y 6 máquinas, el error relativo fue de 3.4 en el SA controlado sin disposiciones de tiempo Vs SA controlado con disposiciones de tiempo. En cuanto al Mk03 que corresponde a un problema de 15 trabajos y 8 máquinas, se presenta una situación interesante, dado que el mejor makespan encontrado en el SA controlado bajo disposiciones de tiempo, supera al valor encontrado para el makespan en el SA controlado sin disposiciones de tiempo, siendo el error relativo de 0.0 %. Y por último en

cuanto al Mk07 que corresponde a un problema de 20 máquinas y 5 trabajos y considerado como uno de los más difíciles de tratar del grupo que se analiza, el error relativo fue de 4.5% con respecto al SA Controlado sin disposiciones de tiempo Vs SA Controlado con disposiciones de tiempo.

Tabla 6-5. SA Controlado sin disposiciones de tiempo Vs SA Controlado con disposiciones de tiempo

Bench marks	Job/ Mach/ Op	UB-LB	ER	SA Controlado sin disposiciones de tiempo	SA Controlado con disposiciones de tiempo
mk01	10/6/55	36-40	4.4	41	43
mk02	10/6/58	24-26	3.4	28	29
mk03	15/8/150	204	0.0	216	214
mk07	20/5/100	133-144	4.5	147	154

La tabla 6-6 muestra que para la instancia de prueba Mk01, el error relativo del SA sin disposiciones de tiempo fue de 2.4 respecto al mejor makespan reportado en la literatura, mientras que el mismo algoritmo bajo el criterio de disposiciones de tiempo fue de 7.5. Para la instancia Mk02, el error relativo fue de 7.1 y 11.5 sin disposiciones de tiempo y con disposiciones de tiempo respectivamente respecto al mejor para esta instancia. Para la instancia Mk03 se presenta un fenómeno interesante, pues el error relativo es 5.5 en el SA sin disposiciones de tiempo mientras que su contraparte, el resultado encontrado es mejor, con un error relativo de 4.9 bajo el criterio de disposiciones de tiempo ambos con respecto al mejor conocido. Y por último para el Mk07, el error relativo para el SA sin disposiciones de tiempo fue de 2.0 mientras que su similar bajo el criterio de disposiciones de tiempo fue de 6.9 con respecto al mejor conocido para ambos casos.

Tabla 6-6 SA sin disposiciones de tiempo y SA con disposiciones de tiempo Vs el Mejor Makespan

Benchmark	Mejor makespan conocido	SA sin disposiciones de tiempo	SA con disposiciones de tiempo	(%) ER de SA sin disposiciones de tiempo	(%) ER de SA sin disposiciones de tiempo
mk01	40	41	43	2.4	7.5
mk02	26	28	29	7.1	11.5
mk03	204	216	214	5.5	4.9
mk07	144	147	154	2.0	6.9

La figura 6-5 muestra de manera clara que al menos una instancia, la Mk03 bajo el criterio de disposiciones de tiempo superó a su similar sin disposiciones de tiempo respecto al mejor Makespan reportado en la literatura.

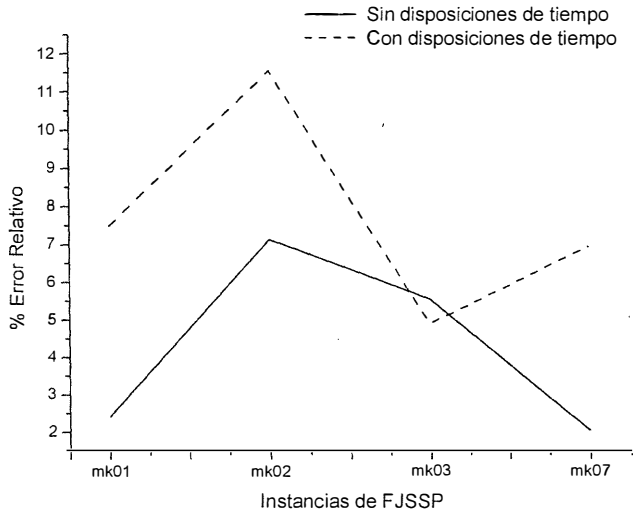

Figura 6-6 Error relativo de SA sin disposiciones de tiempo Vs SA con disposiciones de tiempo

6.5 Resultados experimentales para mostrar la eficiencia y eficacia del Algoritmo SA Paralelizado

Cabe señalar que las soluciones conseguidas por el algoritmo paralelo no siempre tienen que ser mucho mejores que las conseguidas por el algoritmo secuencial, ya que la paralelización del SA sólo introduce una aceleración en el proceso de evaluación de un mayor número de

soluciones en un menor tiempo, más no incide en la ampliación del espacio de búsqueda. Más sin embargo como se puede apreciar en la tabla 6-7 donde se presentan los resultados al aplicar la metaheurística de recocido simulado paralelizado, se logró una mejora en las 4 instancias de prueba utilizadas con respecto a los resultados más recientes reportados en la literatura.

Es importante señalar, que el estudio del FJSSP ha recibido poca atención, por lo que no existe en la literatura resultados obtenidos en ambientes paralelos, más sin embargo sí es posible señalar las mejoras en los resultados logrados con el algoritmo de SA paralelizado respecto a los otros reportados obtenidos en forma secuencial. Para el problema Mk01, que corresponde a un problema de 10 trabajos y 6 máquinas, el error relativo fue de 0% respecto al mejor conocido que es de 40 unidades de tiempo, y siendo mejor en dos de los casos reportados. En cuanto al problema Mk02, que corresponde a un problema de 10 trabajos y 6 máquinas, el error relativo fue también de 0% con respecto al mejor conocido, y mejor que 4 de los 5 casos que se reportan. En cuanto al Mk03 que corresponde a un problema de 15 trabajos y 8 máquinas, se logro encontrar el óptimo reportado en la literatura, no pudiéndose comparar con el resto de los autores, por no presentar un resultado para esta instancia en particular. Y por último en cuanto al Mk07 que corresponde a un problema de 20 máquinas y 5 trabajos y considerado como uno de los más difíciles de tratar del grupo que se analiza, el error relativo fue de 0.0% respecto al mejor conocido y, superando a dos de los 3 casos reportados en la literatura.

Tabla 6-7 Resultados para 4 instancias de prueba para FJSSP de Brandimarte para SA paralelizado

Bench marks	Job/ Mach/ Op	UB-LB	ER	SA Controlado Paralelo	Caso A	Caso B	Caso C	Caso D	Caso E
mk01	10/6/55	36-40	0.00	40	40	42	41	*	*
mk02	10/6/58	24-26	0.00	26	26	32	29	32	28
mk03	15/8/150	204	0.00	204	*204	*	*	*	*
mk07	20/5/100	133-144	0.00	144	144	157	148	*	*

Caso A: Resultados reportados por Mastrolilli y otros [Mastrolilli *Et al.*, 1998]
Caso B: Resultados reportados por Brandimarte y otros [Brandimarte *et al.*, 1993]
Caso C: Resultados reportados por Ho y otros [Ho *et al.*, 2004]
Caso D: Resultados reportados por Kacem y otros [Kacem *et al.*, 2002]
Caso E: Resultados reportados por Neurodine y otros [Neurodinne *et al.*, 2007]

Para analizar la eficiencia y la eficacia del algoritmo SA paralelizado se utilizará la instancia de prueba Mk07, la cual es la más difícil de tratar dentro del grupo de instancias de prueba seleccionadas para probar el algoritmo propuesto para FJSSP. Para ello se definen dos índices respectivamente, el primero corresponde a la aceleración que tiene que ver con la eficiencia lograda por el algoritmo SA paralelizado, dicha aceleración se expresa como la relación entre el tiempo CPU empleado por el algoritmo secuencial en la súper computadora y el tiempo CPU empleado por el SA paralelizado. El segundo índice corresponde con la eficacia lograda con el algoritmo, y la cual se expresa como la relación entre la aceleración lograda y el número de procesos utilizados en el sistema. Por definición, en el caso del algoritmo secuencial que se ejecutó en la súper computadora, utilizando un procesador, su aceleración es 1 y su eficiencia es del 100% como se puede apreciar en la tabla 6-8. Las pruebas fueron realizadas de la siguiente manera: El algoritmo de SA paralelizado fue ejecutado utilizando 1, 2, 4, 8, 16 y 32 procesadores respectivamente, el criterio de paro utilizado fue hasta encontrar makespan promedio de los reportados en la literatura para la instancia de prueba Mk07 (ver tabla 6-6 para los casos A, B, C, D y E) el cual es de 149 unidades.

Tabla 6-8 Resultados obtenidos para la instancia de prueba Mk07 en cuanto a Aceleración y Eficiencia

Número De Procesadores	Tiempo en Segundos	Aceleración (SpeedUp)	%Eficiencia (E)
1	16789.000	1	100.000
2	9530.540	1.7616	88.080
4	5179.552	3.2414	81.035
8	2801.903	5.992	74.900
16	2037.500	8.24	51.500
32	1782.083	9.421	29.4406

La aceleración lograda hace referencia a la velocidad lograda por el algoritmo SA Paralelizado respecto al algoritmo SA secuencial. Como se puede apreciar en la tabla 6-8, al utilizar los 32 procesadores, la velocidad lograda por el SA paralelizado es 9.421 veces más rápido que su contraparte secuencial, y con ello se logra la máxima aceleración pero se sacrifica la eficiencia, de acuerdo a los datos presentados.

En la figura 6-7 se puede apreciar la ganancia en aceleración del algoritmo de SA paralelizado con respecto a su contraparte el algoritmo SA secuencial.

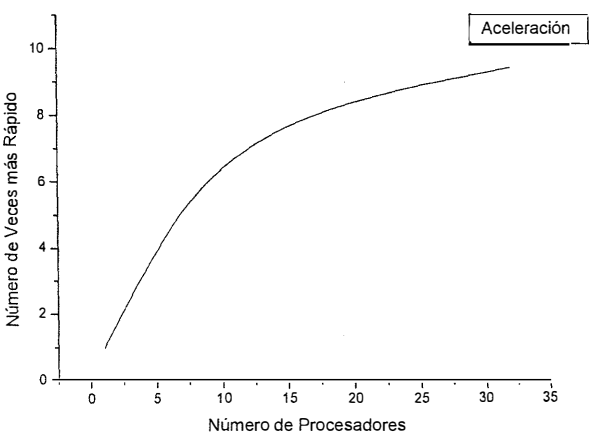

Figura 6-7 Ganancia en aceleración del SA paralelizado Vs SA secuencial

Como se puede apreciar en la figura 6-7 a medida que se aumenta el número de procesadores se observaran tiempos menores o iguales de

rendimiento del algoritmo con respecto a su contraparte el algoritmo SA secuencial, este fenomeno atribuido a la ley de Amdhal ya mencionado en la sección 5.3.

Por otra parte, en la figura 6-8 se observa la eficiencia en % al utilizar 1, 2, 4, 8, 16 y 32 procesadores.

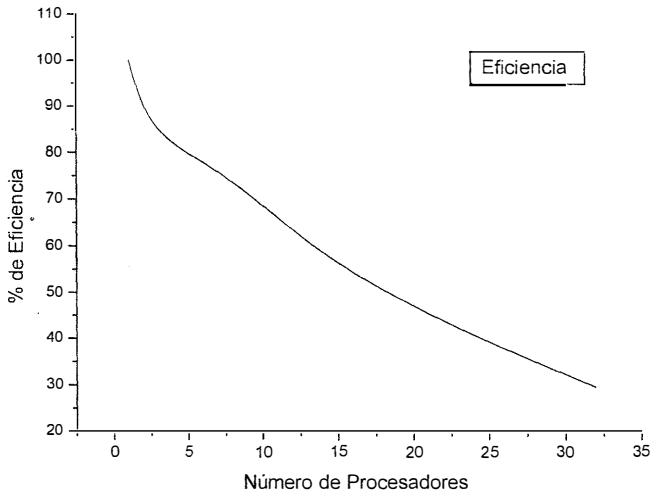

Figura 6-8 Eficiencia del SA-A paralelizado

De acuerdo a los datos presentados en la tabla 6-8, la eficiencia promedio es de 70.82594 que corresponde cuando se hace uso de 8 procesadores, ya que utilizar los 32 procesadores equivale a bajar la eficiencia a 29.4406% como se aprecia en la figura 6-8.

En la figura 6-9 se aprecia la ganancia en tiempo al incrementar el número de procesadores para ejecutar el SA paralelizado donde los tiempos para encontrar el mejor makespan se reducen de manera drástica, ya que debido a la paralelización masiva del SA paralelizado, es posible acelerar la evaluacion de las soluciones al ejecutar en forma paralela el ciclo de Metrópolis, lo que permite encontrar más rápidamente mejores soluciones dentro del proceso de recocido simulado.

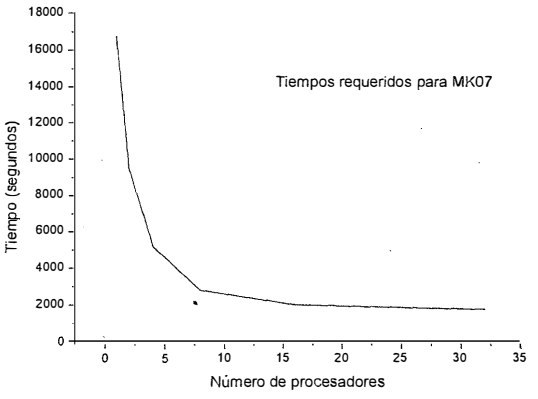

Figura 6.9 Tiempos requridos para la instancia MK07 de acuerdo al número de procesadores utilizados

6.6 Comparación de resultados de Algoritmos Secuencial vs. Paralelo

En la tabla 6-9 se muestra que para la instancia de prueba Mk01, el error relativo del SA secuencial fue de 2.4 respecto al mejor makespan reportado en la literatura, mientras que el algoritmo SA paralelizado 0.0 con un makespan de 26 unidades equivalente al mejor reportado en la literatura, mientras que para la instancia Mk02, el error relativo fue de 7.1 y 0.0 secuencial y paralelo respectivamente respecto al mejor reportado. Para la instancia Mk03 el error relativo fue 5.5 en el SA secuencial, mientras que con el algoritmo de SA paralelizado fue de 0.0, ambos con respecto al mejor conocido. Y por último para el Mk07, el error relativo para el SA secuencial fue de 2.0 mientras que su similar el paralelizado fue de 0.0 con respeto al mejor conocido en ambos casos.

Tabla 6-9 SA secuencial y SA paralelizado vs el mejor makespan reportado

Benchmarks	Job/ Mach/ Op	UB-LB	Mejor Conocido	%ER SA Secuencial	%ER SA Paralelizado	SA Controlado Secuencial	SA Controlado Paralelizado
mk01	10/6/55	36-40	40	2.4	0.00	41	40
mk02	10/6/58	24-26	26	7.1	0.00	28	26
mk03	15/8/150	204	*204	5.5	0.00	216	204
mk07	20/5/100	133-144	144	2.0	0.00	147	144

En la figura 6-10 se observa claramente como el SA paralelizado obtiene mejores resultados que su contraparte el SA secuencial en las cuatro instancias de prueba utilizadas para probar la eficacia y la eficiencia de ambos algoritmos.

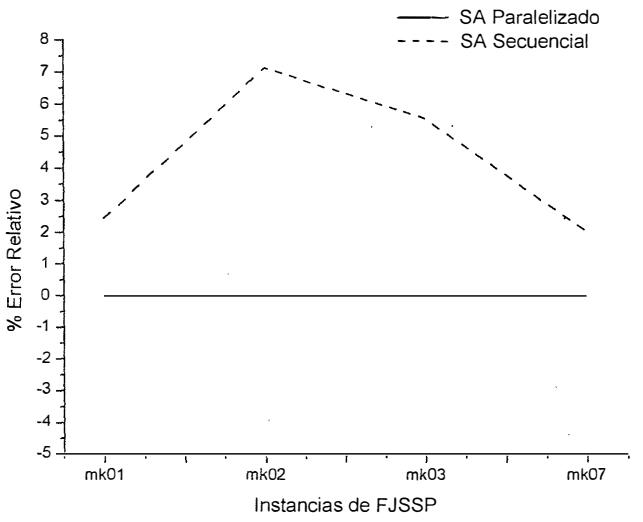

Figura 6-10 Error Relativo de SA paralelizado Vs SA secuencial

6.7 Conclusiones de los resultados

En base a los resultados aquí presentados, podemos resumir las conclusiones en los siguientes puntos:

- La cualificación o mejora de los resultados obtenidos se debe en principio al algoritmo calendarización parcial, denominado Parcial-S, el cual merece una atención especial al permitir que cualquier mecanismo que haga uso de el, pueda evaluar un mayor número de soluciones en un menor tiempo.
- La sintonización de los parámetros utilizados en SA para problemas de scheduling, representa el principal reto a la hora de utilizar esta metaheurística. Los resultados observados en este documento, muestran que es posible sintonizar los parámetros iníciales para Recocido Simulado de acuerdo al grado de dispersión de la calidad

de las soluciones, mediante la desviación estándar para problemas de FJSSP.

- Dependiendo de la complejidad del problema, los valores de los parámetros cambian de manera radical dentro del SA, por lo que la desviación estándar resulta adecuada como un medio para inicializar el parámetro de control para cada tipo de problema.
- Los resultados experimentales obtenidos muestran que el algoritmo de calendarización Parcial-S que forma parte del algoritmo propuesto de recocido simulado es eficiente para problemas de FJSSP bajo criterios de disposiciones de tiempo y sin disposiciones de tiempo.
- En todos los casos donde se aplica el algoritmo Parcial-S, resulta mucho más eficiente para evaluar la calidad de las soluciones obtenidas en un menor tiempo al calendarizar de maneraque cuando se calendariza de manera total. Esto resulta muy conveniente para cualquier algoritmo estocástico tal como SA, que acelera su búsqueda local y por lo mismo pueda explorar un mayor espacio de soluciones para problemas de manufactura (flexible o no flexible).
- Al agregrarle disposiciones de tiempo a las instancias propuestas para FJSSP, el algoritmo de Parcial-S muestra mejor rendimiento en algunas de ellas con respecto a las instancias sin disposiciones de tiempo.
- En los resultados obtenidos al aplicar el algoritmo propuesto de SA secuencial, se observó que el error relativo es mínimo al utilizar SA controlado sin disposiciones de tiempo Vs SA controlado con disposiciones de tiempo, dado que en tres de las instancias el makespan se incrementa pero no de manera significativa cuando se introducen disposiciones de tiempo que corresponden a la preparación de las máquinas, tiempos de intercambio entre las operaciones que se procesan en cada una de las máquinas y el tiempo de limpieza, pero también se observa que en una de las

instancias de prueba, el makespan encontrado en el SA-Controlado con disposiciones de tiempo supera al makespan encontrado en el SA-Controlado sin el criterio de disposiciones de tiempo, por lo que podemos concluir que al establecer el criterio de disposiciones de tiempo en FJSSP no incrementa de manera significativa el makespan, incluso puede ser menor o igual a los resultados en FJSSP sin disposiciones de tiempo.

- La paralelización masiva del SA logra reportar buenos resultados al aplicar el algoritmo propuesto a las instancias de prueba para los problemas de FJSSP, lo anterior permite que a medida que se incremente el número de procesadores, es posible evaluar un mayor número de soluciones.
- Tomando los resultados encontrados, para problemas de scheduling, y en particular para FJSSP, no por muchos procesadores que participen en la ejecución podrá lograrse una aceleración en la ejecución del SA paralelizado, dado que conforme aumenta el número de procesadores que participan en la ejecución del SA paralelizado, este tiende a perder ganancia en la aceleración respecto a su contraparte, el algoritmo SA secuencial. Y una forma de mejorar dicha aceleración dependera en mucho de la estrategia de paralelización utilizada.

Capítulo 7

Conclusiones y Trabajos Futuros

Este capítulo presenta las conclusiones y los trabajos futuros de este trabajo de investigación.

7.1. Conclusiones

En base a los resultados encontrados a lo largo de este trabajo de investigación, es posible llegar a las siguientes conclusiones:

- Los problemas tipo FJSSP caracterizan con mas realismo a los problemas que se presentan en la industria, dado la mejora que día a día se tienen en los máquinas multiprocesos utilizadas.
- La mejora de mecanismos que permitan schedules y su respectiva evaluación permiten lograr la exploración de forma más eficiente del espacio de soluciones para los problemas de scheduling tipo FJSSP como se mostró con los resultados obtenidos mediante el algoritmo Parcial-S propuesto en este trabajo de investigación.
- El análisis de la convergencia de la metaheurística de recocido simulado mediante la variabilidad de los datos en base al análisis estadístico de los mismos, demuestra que siempre es posible lograr mejoras en el trabajo de investigación, ya que la propuesta de sintonización del parámetro de control de la temperatura del algoritmo de recocido simulado acelerado habre una ventana de

posibilidades para explotar aún más esta idea trabajando con otras metaheurísticas, ya sea de manera individual o hibridizáda.

- Para algunas instancias de prueba, donde los resultados fueron mejores en algunos casos, es posible que se haya logrado encontrar el óptimo global para algunas de ellas, como son el caso de Mk01, Mk02 y Mk07. Pero para probar lo anterior, es necesario demostrar que tanto la cota superior (reportadas en esta investigación) es igual la cota inferior para dichas instancias, lo que por el momento no es posible dado que aún no existen reportes en la literatura que muestren dicha convergencia.
- Por último, la combinación de las dos propuestas en este trabajo de investigación: el algoritmo Parcial-S y la sintonización por medio de la desviación estándar, que aceleran la convergencia de SA y permite obtener resultados que compiten en eficacia con los reportados en la literatura para instancias de diferente tamaño para el problema de asignación de tareas en talleres de manufactura tanto en ambientes secuenciales como en ambientes de paralelización.

7.2 **Trabajos Futuros**

Las propuestas que se habren a partir de este trabajo de investigación pueden resumirse de la siguiente manera.

- Aplicar el algoritmo Parcial-S para la calendarización con otras metaheurísticas (algoritmos genéticos, búsqueda tabú, colonia de hormigas, etc) que requieren de un mecanismo eficiente para la generación de vecindades y su respectiva evaluación de cada uno de los vecinos.
- Aplicar la metaheurísticas de recocido simulado a otros tipos de problemas similares dentro del área de optimización combinatoria .
- Explorar con estas nuevas propuestas otros grupos de instancias de pruebas para problemas del área de optimizacion combinatoria.

- Abordar otros tipos de paralelización de la metaheurística de recocido simulado junto con el algoritmo Parcial-S para la calendarización y aplicarlo a problemas scheduling similares dentro del área de optimización combinatoria.

Referencias Bibliograficas

Aanen, E., Gaalman, G., & Nawijn, W. (1993). A scheduling approach for a flexible manufacturing system. International Journal of Production Research, Vol. 31, pp. 2369-2385.

Aarts, B. J. M. (1996) A Parallel Local Search Algorithm for the Job Shop Scheduling Problem, Másters Thesis, Department of Mathematics & Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, April.

Aarts, E. H. L., Van Laarhoven, P. J. M., Lenstra, J. K. and Ulder, N. L. J. (1994) A Computational Study of Local Search Algorithms for Job-Shop Scheduling, ORSA Journal on Computing, Vol. 6(2), Spring, pp. 118-125.

Aarts, E. H. L. and Lenstra, J. K. (eds) (1997) Local Search in Combinatorial Optimization, Wiley,Chichester.

Adams J., Balas E., Zawack D.(1988) The 'shifting bottleneck procedures for job shop scheduling. Management Science, vol. 34, pp. 391-401, 1988.

Agarwal A. and Grossman I. E. (2004) Modeling for Optimization of Hybrid Dynamic Networks. Submitted to Computers Chemical Engineering, 2004.

Allahverdi, A., Gupta, J., & Aldowaisan, T. (1999). A review of scheduling research involving setup consideration. OMEGA, Vol. 27, pp. 219-239.

Antami, A. (1995). A Production planning model for flexible manufacturing systems with setup cost consideration. Computers & Industrial Engineering, Vol. 29, pp. 723-727.

Allahverdi, A., Gupta, J., & Aldowaisan, T. (1999). A review of scheduling research involving setup consideration. OMEGA, Vol. 27, pp. 219-239.

Applegate D., Cook W. (1991) "A computational study of the job shop scheduling problem", ORSA Journal on Computing,Vol. 3, pp. 149-156, 1991.

Aydin M.E., and Fogarty T.C. (2004) A distributed evolutionary simulated annealing algorithm for combinatorial optimisation problems, accepted for publication in Journal of Heuristics, pp. 269-292, May 2004.

Baker, K.R. (1974). Introduction to Sequencing and Scheduling. New York :John Wiley.

Cheung, W., Zhou, H.(2001) Using Genetic Algorithms and Heuristics for Job Shop Scheduling with Sequence-Dependent setup time. Annals of Operational Research, Vol. 107, pp. 65-81 (2001).

Choi, I., & Korkmaz, O. (1997). Job shop scheduling with separable sequence-dependent setups. Annals of Operational Research, Vol. 70, pp. 155-170.

Choi, I.C., & Choi, D.S. (2002). A local search algorithm for job shop scheduling problems with
alternative operations and sequence–dependent setups, Computers & Industrial Engineering, Vol. 42, pp. 43-45.

Amico M.D. and Turbian M. (1993) Applying tahu search to the job shop scheduling problem. Annual Operations Research, Vol. 40, pp. 231-252, 1993

Applegate D., Cook W.(1991) "A computational study of the job shop scheduling problem", ORSA Journal on Computing, Vol. 3, pp. 149-156, 1991.

Artigues C, Roubellat F. (2001). A Petri net model and a general method for on and off-line multi-resource shop floor scheduling with setup time. Int J Prod Econ 2001; pp. 74:63–75.

Authie G., Ferreira A., Roch J.L, Villard G., Roman J., Roućairol C., and Virot B. (1994) Algorithmique Parall`ele: Analyse et Conception, Herm`es, 1994.

Bäck T., Fogel D.B., and Michalewicz Z.(1997), editors. Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Bristol, UK, 1997.

Balas E, Vazacopoulos A.(1998) Guided local search with shifting bottleneck for job shop scheduling. Manage Sci 1998; Vol. 44:, pp. 262–75.

Barnes J.W, Chambers J.B.(1996-1999) Flexible job shop scheduling by Tabu search. Technical report series ORP 96-09. Department of Mechanical Engineering, University of Texas at Austin, 1996.

Barnes, J. W. and Laguna, M. (1993) A Tabu Search Experience in Production Scheduling, Annals of Operations Research, Vol. 41, pp.141-156.

Battiti R. and Tecchiolli G. (1994) The Reactive Tabu Search. ORSA Journal on Computing, Vol. 6(2): pp. 126-140, 1994.

Beasley J. E..(2003) OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society,Vol. 41. No. 11, pp.1069-1072, 1990. Last update 2003.

Bertocchi M., and Sergi P. (1992) "Parallel global optimization over continuos domain by simulated annealing. In P. Messina and A. Murli (Eds), Proccedings of parallel computing: problems, methods and applications. Ámsterdam: Elsevier, pp. 87-97. 1992.

Bilchev, G., Parmee, I.C. (1995) The Ant Colony Metaphor for Searching Continuous Design Spaces. Lecture Notes in Computer Science, Vol. 993,pp. 25-39, 1995.

Bierwirth, C. (1995) A Generalized Permutation Approach to Job-Shop Scheduling with Genetic Algorithms, OR Spektrum, Vol.17(2-3), pp. 87-92.

Blazewicz J, Domschke W, Pesch E.(1996) The job shop scheduling problem: conventional and new solution techniques. Eur J Op Res 1996;Vol. 93:pp.1–33.

Blum C.and Roli A. (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, Vol. 35(3):pp. 268-308, 2003.

Blum C, Sampels M.(2004) An ant colony optimization algorithm for shop scheduling problem. J Math Model Algorithm 2004;Vol. 3:pp. 285–308.

Bonabeau E, Dorigo M, Theraulaz G. (2003) Inspiration for optimization from social insect behaviour. Nature 2000;Vol.406:pp.39–42.

Brandimarte P. (1993) "Routing and Scheduling in a Flexible Job Shop by Tabu Search," Annals of Operations Research, vol.2, pp. 158-183, 1993.

Barnes, J. W. and Chambers, J. B. (1995) Solving the Job Shop Scheduling Problem Using Tabu Search, IIE Transactions, vol 27, pp. 257-263.

Brucker P, Thiele O.(1996). A branch & bound method for the general-shop problem with sequence dependent setup-times. OR Spektrum 1996; Vol.18:pp.145–61.

Bruker P. Jurisch B. Sievers B. (1994) A Branch & Bound Algorithm for the Job Shop Scheduling Problem, Discrete Applied Mathematics, Vol.49, pp.107, 1994.

Burke E.K. and Smith A.J..(1997) A Memetic Algorithm for the Maintenance Scheduling Problem, Proceedings of the ICONIP'97 Conference, Dunedin, New Zealand, 24-28 November 1997, (published by Springer), pp. 469-474.

Buyya R. (1999) (editor), High performance cluster computing: Architectures and systems (vols. 1 and 2), Prentice-Hall, 1999.

Campos V., Glover F., Laguna M, and Martí R. (2001) An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. Journal of Global Optimization, Vol.21:pp. 397-414, 2001.

Carlier P. and Pinson E. (1989) An Algorithm for solving the Job Shop Problem. Manage. Sci., Vol.35(2) . pp.164-176, 1989.

Cerny, V.:(1985) Thermodynamical Approach to the Traveling Salesman Problem: an Eficient Simulation Algorithm. Journal of Optimization Theory and Applications Vol.45 (1985) pp. 41-51

Colorni A, Dorigo M, Maniezzo V, Trubian M.(1994) Ant system for job-shop scheduling. Belg J Oper Res, Stat Comput Soc 1994;Vol.34: pp. 39–53.

Cruz-Chavez (2005) "cooperación de procesos para el problema de Job shop scheduling Problem aplicando Recocido Simulado" Tesis Doctoral. Instituto Tecnológico y de Estudios Superiores de Monterrey, 2005.

Cruz-Chávez M. A., Díaz-Parra O., Juárez-Romero D, Barreto-Sedeño E., Zavala-Díaz J.C, Martínez-Rangel M. G. (2008), Un Mecanismo de Vecindad con Búsqueda Local y Algoritmo Genético para Problemas de Transporte con Ventanas de Tiempo, CICos 2008, 6to Congreso Internacional de Cómputo en Optimización y Software, ACD, ISBN(e) 978-607-00-0165-9, ISBN(i) 978-970-9750-26-3, pp., 25-27 Junio, México, 2008.

Cruz-Chávez M.A, Díaz-Parra O., Hernández J. A., Zavala-Díaz J. C., Martínez-Rangel M. G. (2007), Search Algorithm for the Constraint Satisfaction Problem of VRPTW, Electronics, Robotics and Automotive Mechanics Conference, CERMA2007, IEEE-Computer Society, ISBN 0-7695-2974-7, pp 746-751, 25-28 September, México, 2007.

Cruz-Chávez M. A, Martínez-Rangel M. G. Un Algoritmo de Calendarización Parcial de Recursos para un Taller de Centros de Máquinado, Revista del IEEE América Latina, IEEE-Computer Society, Region 9, ISSN: 1548-0992 , 2008.

Cruz-Chávez M. A., Díaz-Parra O., Juárez-Romero D., Martínez-Rangel M. G. (2008) Memetic Algorithm Based on a Constraint Satisfaction Technique for VRPTW, Lecture Notes in Artificial Intelligence, Springer Verlag Pub., Berlin Heidelberg, ISSN: 0302-9743, Vol. 5097, pp. 376-387, 2008.

Cruz-Chávez M. A., Frausto-Solís J., Romero D. J.(2005) Experimental Analysis in Simulated Annealing to Scheduling Problems when Upper Bounds are used, Transactions on Information Science and Applications, ISSN: 1790-0832, Vol. 2, Issue 5, pp. 581 - 586, 2005.

Cruz-Chávez M. A., Frausto-Solís J., Cora-Mora J.R.(2006) Experimental Analysis of a Neighborhood Generation Mechanism Applied to Scheduling Problems, Proceeding of CERMA2006, IEEE-Computer Society, ISBN 0-7695-2569-7, pp 226-229, 26-29 September, México, 2006.

Cruz-Chávez M. A., Martinez-Rangel M.G., Hernandez-Perez J. A.,.-Zavala-Diaz J. C, Díaz-Parra O.(2007) An Algorithm of Scheduling for the Job Shop Scheduling Problem, Proceeding of CERMA2006, IEEE-Computer Society, ISBN, pp, 25-28 September, México, 2007.

Culler, David, Singh, J.P. and Gupta, A.(1998) Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kauffman Publishers,1998. ISBN 1-55860-343-3.

Culler D.E., Jaswinder P.S., and Gupta A. (1999) Parallel computer architecture: A hardware/software aspproach, Morgan Kaufmann, 1999.

Cung V., Martins S.L., Ribeiro C., and Roucairol C. (2001) Essays and Surveys in Metaheuristics, capítulo Strategies for the Parallel Implementation of Metaheuristics. Kluwer Academic Publisher, 2001

Chambers J.B. and Barnes J. W.(1996a) Flexible Job Shop Scheduling by Tabu Search. Department of Computer Sciences, TAY 2.124, The University of Texas at Austin, Austin, Texas 78712, 1996

Chardaire P., Lutton J.L., and Sutter A. (1995). Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. European Journal of Operational Research, Vol.86:pp.565-579, 1995.

Cheng-Fa Tsai, Chun-Wei Tsai, and -Chin-Chang Tseng.(2003) New and efficient antibased heuristic method for solving the traveling salesman problem; Expert Systems (accepted, will appear in vol. 20, no. 4, 2003)

Cheng-Fa T., Chun-Wei T., and Chi-Ping. C. (2002) A Multiple- Searching Approach to Genetic Algorithms for Solving.Traveling Salesman Problem. 6th Intem. Conf on Computer Science and Informatics, pp. 362-366, Durham; NC, USA.

Cheung W. and Zhou H. (2001) "Using Genetic Algorithms and Heuristics for Job-Shop Scheduling with Sequence-Dependent setup time", Annals of Operations Research, vol. 107, 2001, pp. 65-81.

Choi I-C. and Choi D-S., "A Local Search Algorithm for Job-Shop Scheduling Problems with Altematives Operations and Sequence-Dependent Setups", Computers & Industrial Engineering, vol.42, 2002, pp. 43-58.

Choi I-C. and Korkmaz0. (1997) "Job-Shop Scheduling with Separable Sequence-Dependent Setups Times", Annals of Operations Research, 1997, pp. 155-170.

Dagum L. and Menon R. (1998) OpenMP: An industry-standard API for shared-memory programming", IEEE Computational Science and Engineering Vol. 5 (1998), pp. 46–55.

Dauzere-Peres S. and Paulli J.(1997). An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using Tabu search. Ann Oper Res 1997;Vol.70:pp.281–306.

Davis, L. (1985) Job-Shop Scheduling with Genetic Algorithm, in Grefenstette J. J. (ed) Proceedings of the 1st International Conference on Genetic Algorithms and their Applications, Pittsburgh, PA, USA, Lawrence Erlbaum, pp. 136-140.

De Almeida D, Kellert P. (2000) An analytical queueing network model for flexible manufacturing systems with a discrete handling device andtransfer blockings, Int J Flex Manuf Sys 2000;Vol.12:pp. 25–57.

Dell'Amico, M. and Trubian, M. (1993) Applying Tabu Search to the Job-Shop Scheduling Problem, Annals of Operations Research, vol 41, pp. 231-252.

Der and Steinhofel (2001) "A parallel implementation fo Job Shop Scheduling Heuristic in Sorevik T.,Manne, F., More, R., Gebremedhin, A.H. (eds.), Proc. 5th International Workshop on Applied Parallel Computing, Springer-Verlag (LNCS 1947, pp. 215-222, 2001.

Dorigo M.(1992) Optimization Learning and Natural Algorithms. PhD thesis, DEI, Polytecnico di Milano, Milan, 1992

Dorigo M, Gambardella L.M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE T Evolut Comput 1997;vol.1:pp. 53–66.

Dorigo M. and Stützle T.(2004) Ant Colony Optimization. MIT Press, Cambridge, MA, 2004.

Dorndorf, U. and Pesch, E. (1995) Evolution Based Learning in a Job-Shop Scheduling Environment, Computers and Operations Research, Jan, Vol.22(1), pp.25-40.

Dueck, G. and Scheuer, T. (1990) Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing, Journal of Computational Physics, vol 90, pp. 161-175.

Dueck, G. (1993) New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel, Journal of Computational Physics, vol 104, pp. 86-92.

Fattahi P., Mehrabad M.S, Jolai F.(2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. . J Intell Manuf (2007) Vol. 18:pp. 331–342.

Falkenauer, E. and Bouffouix, S. (1991) A Genetic Algorithm for the Job-Shop, Proceedings of the IEEE. International Conference on Robotics and Automation, Sacremento, California, USA, pp. 824-829.

Feo T.A. and Resende M. G. C. (1995)Greedy randomized adaptive search procedures. Journal of Global Optimization, Vol.6:pp.109-133, 1995.

Fisher H., and Thompson G.L.(1993) Probabilistic Learning Combinations of Local Job Shop Scheduling Rules, In J.F. Muth and G.L. Thompson (Eds.), Industrial Scheduling, Prentice Hall. Englewood Cliffs. New Jersey pp.225-251. 1993.

Foster I. (1995) Designing and Building Parallel Programs. Addisson-Wesley, 1995. ISBN 0-201-57594-9.

Freitas K. G. and Aparecida F. M. (2006) An Approach for Flexible Job-Shop Scheduling with Separable Sequence-Dependent setup time, 2006 IEEE International Conference on Systems, Man, and Cybernetics October 8-11, 2006, Taipei, Taiwán

Garey, M. R. Jonson, D.S and Shethi, R.(1976) "The complexity of Flor Shop and Job Shop Scheduling, in Mathematics Of Operation research, vol. 1; No.2 (1976) pp. 117-129.

Giffler D, Thompson GL.(1960) Algorithms for solving production scheduling problems, Oper Res 1960;Vol. 8:pp. 487–503.

Glover F. (1999) Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, Advanced topics in computer science series. McGraw-Hill, 1999.

Glover, F. and Laguna, M. (1997) Tabu Search, Kluwer Academic Publishers, Norwell, MA.

González M. A., Sierra M., Vela C. R., Varela R (2005) Genetic Algorithms Hybridized with Greedy Algorithms and Local Search over the Spaces of Active and Semi-active Schedules. CAEPIA 2005: pp.231-240.

Goldberg, D. E. (1989) Genetic Algorithms in Search Optimisation and Machine Learning, Addison-Wesley, Reading, Massachusetts.

Grefenstette, J. J. (1987) Incorporating Problem Specific Knowledge into Genetic Algorithms, in Davis, L. (ed) Genetic Algorithms and Simulated Annealing, Pitman, pp. 42-60.

Grama A., Karypis G., Kumar, Gupta A(2003). An Introduction to Parallel Computing: Design and Analysis of Algorithms (2nd Ed.). Addisson-Wesley 2003. ISBN 0201648652.

Greening D., (1990) "Parallel Simulated Annealing Techniques" Physica D 42,1990.

Gropp W., Huss-Lederman S., Lumsdaine A., Lusk E., Nitzberg B., Saphir W., and Snir M.(1998) MPI: The complete reference, Volume 2 - The MPI extensions, MIT Press, 1998.

Gueist A., Beguelin A., Dongarra J., Jiang W., Manchek R., and Sunderam V., (1994) PVM: Parallel Virtual Machine - A user's guide and tutorial for networked parallel computing.

Hansen P. and Mladenovic N. (2001). Variable neighborhood search: Principies and applications. European journal of Operational Research, Vol. 3:pp. 449-467, 2001.

Ho, N.B.; Tay, J.C. (2004) GENACE: an efficient cultural algorithm for solving the flexible job-shop problem. Evolutionary Computation, 2004. CEC2004, Congress on Volume 2, Issue , 19-23 June 2004 pp. 1759 – 1766.

Holland, J. H. (1975) Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor.

Hong B. L., Ajith A, Okkyung C. Seong H. M.(2006) Variable Neighborhood Particle Swarm Optimization for Multi-objective Flexible Job-Shop Scheduling Problems. pág. 197-204, Electronic Edition (link) BibTeX, Revista: Lecture Notes in Computer Science, 2006, ISSN: 03029743.

Hurink J, Knust S. (2005). Tabu search algorithms for job-shop problems with a single transport robot. Eur J Op Res 2005;Vol.162:pp. 99–111.

Ivens P, Lambrecht M. (1996) Extending the shifting bottleneck procedure to real-life applications. Eur J Op Res 1996;Vol.90:pp. 252–68.

Jain A. Meeran. S. (1998) A State of the Art Review of JOBSHOP Scheduling Techniques. Technical Report. Department of Applied Physics, Electronic and Mechanical Engineering University of Dundee, Dundee, Scotland, K,DD1 4HN,1998.

Jain A, Meeran S. (1999)Deterministic Job shop scheduling; past, present and future. Eur J Op Res 1999; Vol.113:pp. 390–434

John B. Chambers and J. Wesley Barnes.(1996): Flexible Job Shop Scheduling by Tabu Search. Department of Computer Sciences, TAY 2.124, The University of Texas at Austin, Austin, Texas 78712, 1996.

KallRath J. (2000) Mixed integer optimization in the chemical process industry, experience, Potential and Future Perspectives .Edited by. ICE The Institution of Chemical Engineers. September 2000. Trans IChemE. Vol. 78. Part A.

Kacem, I.; Hammadi, S.; Borne, P. (2001): Approach by localization and genetic manipulation algorithm for flexible job-shop scheduling problem.Systems, Man, and Cybernetics, 2001 IEEE International Conference on Volume 4, Issue , 2001 vol.4. pp.:2599 - 2604

Kacem, I. Hammadi, S. Borne, P. (2002a):Pareto-Optimality approach based on uniform design and fuzzy evolutionary algorithms for flexible job-shop scheduling problems (FJSPs). Systems, Man and Cybernetics, 2002

IEEE International Conference on 6-9 Oct. 2002 .Volume: 7, pp. 7..ISBN: 0-7803-7437-1

Kacem, I. (2003):Genetic algorithm for the flexible job-shop scheduling problem. Systems, Man and Cybernetics, 2003. IEEE International Conference on Volume 4, Issue , 5-8 Oct. 2003, vol.4 pp. 3464 - 3469

Kacem, I.; Hammadi, S.; Borne, P. (2002b): Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on Volume 32, Issue 1, Feb 2002 . pp.:1 – 13.

Kacem, I., Hammadi, S., Borne, P., (2002c). Lower bounds for evaluating schedule performances in flexible job shops. Performance Metrics for Intelligent Systems Workshop, PerMIS'02, Gaithersburg, MD, 2002. USA.

Kacem I, Ordonnancement multicritere des job-shops flexibles: formulation, bornes inferieures et approche evolutioniste cooperative. Ph.D Thesis. Ecole Centrale de Lille. January 6, 2003. France.

Kacem, I., Hammadi, S.,Borne, P. (2002d) Pareto-optimality Approach for Flexible Job-Shop Scheduling Problem: Hibridization of Evolutionary Algorithms and Fuzzy Logic, Journal of Mathematics and Computer in Simulation, 2002.

Kendall M. G., Stuart A. (1958) The advanced Theory of Statistics, Vol.1 London: Charles Griffin 1958.

Kim, S., & Bobrowski, P. (1992). Impact of sequence-dependent setup time on job shop scheduling performance. International Journal of Production Research, Vol. 32, pp.1503-1520.

Kirkpatrick S., Gelatt S. D. Jr., and Vecchi M. P.(1983) Optimization by simulated annealing. Science, 220(4598), 13 May, pp. 671-680, 1983.

Klahorst, T.H. (1981) Flexible Manufacturing Systems: Combining Elements to Lower Costs, add Flexibility, Industrial Engineering, Vol 32, No. 11, 1981.

Koelbel C., Loveman D., Schreiber R.,. Steele G. Jr., and Zosel M. (1994) The High Performance FORTRAN handbook, The MIT Press, 1994.

Kolonko M. and Tran M.T. (1997) Convergente of Simulated Annealing with Feedback Temperatura Schedule. Problem in The Engineeering and Informational Sciences, Vol.11, pp. 279-304, 1997.

Kolonko, M. (1998) Some New Results on Simulated Annealing Applied to the Job Shop Scheduling Problem, European Journal of Operational Research. 1998.

Kenneth, M. K. and Booker, L. B. (eds) Proceedings of the 4th International Conference on Genetic Algorithms and their Applications, San Diego, USA, pp. 474-479.

Knies A., O'Keefe M., and MacDonald T. (1994) "High Performance FORTRAN: A practical analysis", Scientific Programming 3 (1994, pp. 187–199.

Kumar R, Tiwari MK, Shankar R(2003). Scheduling of flexible manufacturing system: an ant colony optimization approach. J Eng Manuf—Part B 2003;217:1443–53.

Kobayashi, S., Ono, I. and Yamamura, M. (1995) An Efficient Genetic Algorithm for Job Shop Scheduling Problems, Proceedings of the Sixth

International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Francisco, CA, pp. 506-511.

Laarhoven V. Aarts E. and Lenstra. J. (1992) Job shop scheduling by simulated annealing. Operations Research, 40, pp.113-125, 1992

Laguna, M., Barnes, J. W. and Glover, F. W. (1991) Tabu Search Methods for a Single Machine Scheduling Problem, Journal of Intelligent Manufacturing, vol 2, pp. 63-74.

Laguna, M., Barnes, J. W. and Glover, F. W. (1993a) Intelligent Scheduling with Tabu Search: An Application to Jobs with Linear Delay Penalties and Sequence-Dependent Setup Costs and Times, Journal of Applied Intelligence, vol 3, pp. 159-172.

Laguna, M. and Glover, F. (1993b) Integrating Target Analysis and Tabu Search for Improved SchedulingSystems, Expert Systems with Applications, vol 6,pp. 287-297.

Lawrence S., Resource Constrained Project Scheduling:(1984) An Experimental Insvestigation of Heuristic Scheduling Techniques (Suplement). Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, Pensylvania, 1984.

Lee K. M. and Kawa T. Y. (1998) A Genetic Algorithm for General Machine Scheduling Problems, International Journal of Knowledge Based Electronics Systems, vol. 2, 1998, pp. 60-66.

Liouane N, Saad I, HammadiS, Borne P.: Ant systems & Local Search Optimization for flexible Job Shop Scheduling Production. International Journal of Computers, Communications & Control.Vol. II (2007), No. 2, pp. 174-184, 2007.

Lourenço H. R., Martin O, and Stützle T. (2002). Iterated Local Search. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science, pp. 321-353. Kluwer Academic Publishers, Norwell, MA, 2002.

Mattfeld, D. C. (1996) Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for Production Scheduling, Physica-Verlag, Heidelberg, Germany.

Manderick B. and Spiessens P. (1989). Fine-grained parallel genetic algorithms. In J. D. Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pp. 428-433. Morgan Kaufmann, 1989.

Maqrini E.I, Teghem H. (1995) Scheduling complex flexible job shop problems, Emerging Technologies and Factory Automation, 1995. ETFA '95, Proceedings., 1995 INRIA/IEEE Symposium on.Volume 1, Issue , 10-13 Oct 1995 Page(s):541 - 549 vol.1. Paris, France.ISBN: 0-7803-2535-4.

Martínez-Rangel Martín G., Cruz-Chávez Marco Antonio, Zavala-Díaz J Crispín, Juárez-Romero David and Díaz-Parra Ocotlán.(2007) Analysis of the Simulated Annealing Convergence in Function of the Standard Deviation and the Boltzmann Quotient for Scheduling Problems, Research in Computing Science, IPN, ISSN 1870-4069,pp.,282-293 2007.

Martínez-Rangel M. G., Cruz-Chávez M.A, Juárez-Romero D., Díaz-Parra O., Zavala-Díaz J.C. (2008) Metodología para Resolver Problemas de Optimización Discreta, CICos 2008, 6to Congreso Internacional de Cómputo en Optimización y Software, ACD, ISBN(e) 978-607-00-0165-9, ISBN(i) 978-970-9750-26-3, pp., 25-27 Junio, México, 2008.

Martínez-Rangel M. G., Cruz-Chávez M. A, Galván-Montiel D, Zavala-Díaz J.C. (2006) Modelado del Problema de Calendarización de Recursos para la Fabricación de Compresores, 5to Congreso de Cómputo AGECOMP, UAEM, ISBN(i) 970-9750-02-X, ISBN(e) 968-878-273-4, pp. 155-168, 21-23 Noviembre, México, 2006.

Mastrolilli M, Gambardella L.M. (2000). Effective neighborhood functions for the flexible job shop problem. J Sched 2000;3:3–20.

Matsuo, H., Suh, C. J. and Sullivan, R. S. (1988) A Controlled Search Simulated Annealing Method for the General Job-Shop Scheduling Problem, Working Paper #03-04-88, Graduate School of Business, The University of Texas at Austin, Austin, Texas, USA.

Mastrolilli M., Gambardella L. M.(1998) Effective Neighborhood Functions for the Flexible Job Shop Problem. Technical Report. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale. IDSIA-45-98,1998.

Metrópoliss N., Rosenbluth A. W.,. Rosenbluth M, Teller A. H., and Teller. E. (1953) "Equation of State Calculations by Fast Computing Machines." J. Chem. Phys. Vol. 21,pp. 1087-1092, 1953.

Morikawa K., Furuhashi T., Uchikawa. Y. (1992) Single Populated Genetic Algorithm and its Application to Job-shop Scheduling. Proc. Of Industrial Electronics, Control,Instrumentation, and Automation on Power Electronics and Motion Control, pp. 1014-1019, 1992

Moscato,. P. (1989) On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms (Technical Report C3P 826). Pasadena, CA: Caltech Concurrent Computation Program, California Institute of Technology.

Mühlenbein H. and Paass G. (1996) From recombination of genes to the estimation of distributions. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H. -P. Schwefel, editors, Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV, volume 1411 of Lecture Notes in Computer Science,pp. 178-187, Berlin, 1996. Springer.

Muth J.F. and Thompson G.L.(1963) Industrial Scheduling. Prentice-Hall, Englewood Cliffs, N.J., 1963.

Nabeshima I, Maruyama S. (1984) Two- and three-machine flowshop makespan scheduling problems with additional times separated from processing times. J Oper Res Soc Japan 1984;Vol.27:

Najid N.M., Dauzère-Pérès S., Zaidat A. (2002) A modified simulated annealing method for Flexible Job Shop Scheduling Problem. IEEE Inter. Conf. on Systems, Man and Cybernetics, IEEE SMC02, Tunis, Tunisie, Septembre 2002.

Nakano, R. and Yamada, T. (1991) Conventional Genetic Algorithm for Job-Shop Problems, Proceedings of the fourth International Conference on Genetic Algorithm, Morgan Kaufmann; San Mateo, CA, USA, 1991, pp.477-479.

Noureddine Liouane, Ihsen Saad, Slim Hammadi, Pierre Borne.(2007) Ant systems & Local Search Optimization for flexible Job Shop Scheduling Production. International Journal of Computers, Communications & Control.Vol. II (2007), No. 2, pp. 174-184, 2007.

Nowicki E., Smutnicki. C. (1996) A Fast Taboo Search Algorithm: for the Job Shop Problem. Managemenr Science, vol. 42, pp. 797-813, 1996.

Ong Z. X., Tay J. C. and Kwoh C. K. (2005) "Applying the Clonal Selection Principle to find Flexible Job-Shop Schedules", In Proceedings of the 4th International Conference on Artificial Immune Systems, 2005, pp. 442-455.

Pacheco P. Parallel Programming With MPI. Morgan Kaufmann. ISBN 1558603395.

Papadimitriou C H., Steigliths K.(1998) Combinatorial Optimization. Algorithms and Complexity. Dover Publications, Inc. 1998.

Pinedo M. (2001) Scheduling Theory, Algorthms, and Systems, ISBN: 0130281387, 2 ed. Prentice Hall. USA., Aug. 2001.

Philip D. (2005) "Scheduling Reentrant Flexible Job Shops with Sequence-Dependent setup time", MSc. Thesis, Montana State University, Montana, 2005.

Quinn, M. J. Parallel Programming in C with MPI and OpenMP. McGraw-Hill. 2003. ISBN 0072822562.

Ranky, P.(1983) The Design and Operation of FMS: Flexible Manufacturing Systems, IFS Publications Ltd., Bedford, 1983.

Rossi A, Dini G. (2000). Dynamic scheduling of FMS using a real-time genetic algorithm. Int J Prod Res 2000;38:1–20.

Rossi A, Dini G. (2001). An evolutionary approach to complex job-shop scheduling and flexible manufacturing system scheduling. J Eng Manuf—Part B 2001;215:233–45.

Roy B. and Sussman B. (1964) Les problemes d'ordonnancement avec contraintes disjonctives, Note D.S. no 9 bis, SEMA, Paris, France, December 1964.

Saidi-Mehrabad M., Fattahi P. (2007) Flexible job shop scheduling with tabu search algorithms. International journal of advanced manufacturing technology. ISSN 0268-3768, Vol. 32, Nº. 5-6, 2007, pp.. 563-570.

Sánchez J. (1997) "Apuntes de la clase: Cómputo Paralelo y Distribuido", ITESM Campus Edo. De México. 1997.

Sanvicente-Sánchez H, Frausto-Solís J.(2002) MPSA: A Methodology to Parallelize Simulated Annealing and Its Application to the Traveling Salesman Problem. MICAI 2002: 89-97.

Sanvicente-Sánchez, H., (1997). Recocido simulado:optimizacion combinatoria. Estado del arte. Instituto Tecnológico y de Estudios Superiores de Monterrey, (campus Morelos, México. 72pp.

Sanvicente-Sánchez, H.,(1998). Recocido simulado paralelo. Propuesta de tesis doctoral Instituto Tecnológico y de Estudios Superiores de Monterrey, (campus Morelos, México. 38pp.

Schwefel P. (1981) Numerical Optimization of computer models. Chchester: Wiley, 1981.

Shi, G. (1997) A Genetic Algorithm Applied to a Classic Job-Shop Scheduling Problem, International Journal of Systems Science, Vol. 28(1), pp. 25-32.

Souissi A., Kacem I. et Chu C.. (2004) New lower bound for minimizing total tardiness on a single machine with sequence dependent setup time. PMS'04, April 26-28, Nancy (France), 2004.

Stutzle T, Hoos HH. (2000) MAX-MIN ant system. Future Gener Comp Sys 2000;Vol.16: pp. 889–914.

Talbi E.G. (2002). A Taxonomy of Hybrid Metaheuristics. Journal of Heuristics, Vol.8(5): pp.541-564, 2002.

Taillard, É. (1994) Parallel Taboo Search Techniques for the Job-Shop Scheduling Problem. ORSA Journal on Computing, Vol16(2), pp.108-117.

Tamaki H., Ono T., Murao H. and Kitamura S. (2001) "Modeling and Genetic Solution of a Class of Flexible Job-Shop Scheduling Problems", IEEE International Conference on Automation, 2001, pp. 343-350.

Tamaki, H. and Nishikawa, Y. (1992) A Parallelled Genetic Algorithm Based on a Neighbourhood Model and its Application to the JobShop Scheduling, in Männer, R. and Manderick, B. (eds) PPSN'2 Proceedings of the 2nd International Workshop on Parallel Problem Solving from Nature, Brussels, Belgium, pp. 573-582.

Thomsen, S. (1997) Meta-heuristics Combined with Branch & Bound, Technical Report, Copenhagen Business School, Copenhagen, Denmark (in Danish).

Van der S. Zwaan and Marques. C. (1999) Ant colony optimisation for job shop scheduling. In *Proceedings of the Third Workshop on Genetic Algorithms and Artificial Life, GAAL 99*,1999. Noureddine Liouane1, Ihsen Saad2,3, Slim Hammadi2, Pierre Borne2 1ATSI : Ecole Nationale d'Ingénieurs de Monastir,rue Ibn El Jazzar, 5019 Monastir, Tunisie.

Voudouris C. and Tsang E. (1999). Guided Local Search. European Journal of Operational Research, Vol.113(2):pp.469-499, 1999.

Wilkinson B and Allen M.(2004) Parallel Programming: Techniques and Aplications Using Networked Workstations and Parallel Computers (2nd Ed.). Prentice-Hall Inc. 2004, ISBN 0131405632.

Xia W. and Wu Z.(2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Computers and Industrial Engineering,(2005) Vol. 48:pp.409-425

Yamada, T., Rosen, B. E. and Nakano, R. (1994) A Simulated Annealing Approach to Job-Shop Scheduling using Critical Block Transition Operators, IEEE ICNN'94 International Conference on Neural Networks, Orlando, Florida, 26-29 June, vol 6, pp. 4687-4692.

Yamada T. R and Nakano.(1992) A Genetic Algorithm Aplicable to Large-Scale Job-Shop Instante Solving from nature 2, North-Holland, Ámsterdam 281-290, 1992

Yamada, T. and Nakano, R. (1995a) Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search, MIC'95 Meta-heuristics International Conference, Hilton, Breckenridge, Colorado, USA, July 22-26, pp. 344-349.

Yamada, T. and Nakano, R. (1995b) A Genetic Algorithm with Multi-Step Crossover for Job-Shop Scheduling Problems, GALESIA'95 Proceedings of the Int. Conf. on GAs in Eng. Sys., pp. 146-151.

Yamada, T. and Nakano, R. (1996a) Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search, Meta-heuristics: Theory and Applications, Kluwer Academic Publishers, MA, USA, pp. 237-248.

Yamada, T. and Nakano, R. (1996b) Scheduling by Genetic Local Search with Multi-Step Crossover, PPSN'IV Fourth International Conference on Parallel Problem Solving from Nature, Berlin, Germany, Sept 22-26, pp. 960-969.

Yang DL, Chern MS. (2000)Two-machine flowshop group scheduling problem. Comput Oper Res 2000;Vol. 27:pp.975–85.

Zalzala P. J., and Flemming. Zalzala, A.M.S. (1997) (Ali M.S.),ed. Genetic algorithms in engineering systems /Edited by A.M.S. Institution of Electrical Engineers, London, 1997.

Zribi, N. Kacem, I. El Kamel, A. Borne, P. (2004): Optimization by phases for the flexible job-shop scheduling problem.LAGIS, Ecole Centrale de Lille, Villeneuve d'Ascq, France. 20-23 July 2004.Vol: 3, pp: 1889- 1895. ISBN: 0-7803-8873-9

APÉNDICE 1

Metodología MPESA para la paralelización de Recocido Simulado

Cualquier implementación que siga los pasos establecidos en la metodología MPSA, generará un algoritmo tipo recocido simulado paralelo, que construirá las cadenas de Markov del algoritmo en forma paralela. MPSA establece que cualquier algoritmo tipo recocido simulado paralelo tiene el mismo esquema de enfriamiento que el algoritmo secuencial. Sin embargo, para llevar a cabo la paralelización del ciclo de temperaturas, por lo menos la temperatura inicial, C_1; la función de enfriamiento, $f(C_k)$, y la temperatura final, C_f, del esquema de enfriamiento deben ser conocidas con anterioridad a la implementación del algoritmo. Ellos pueden ser tomados de la literatura o determinados mediante un proceso de sintonización.

Para el Kernel MPSA cada procesador envía su solución final después de generar una cadena de Markov de la longitud establecida, a todos los procesadores que están trabajando a temperaturas inferiores a la suya. Cuando un procesador recibe una solución, la evalúa a través de un criterio de aceptación o rechazo; si es que es aceptada el procesador reinicia la cadena de Markov que está construyendo con esta nueva solución, si es rechazada continua de manera normal, como se muestra en la figura A1-1.

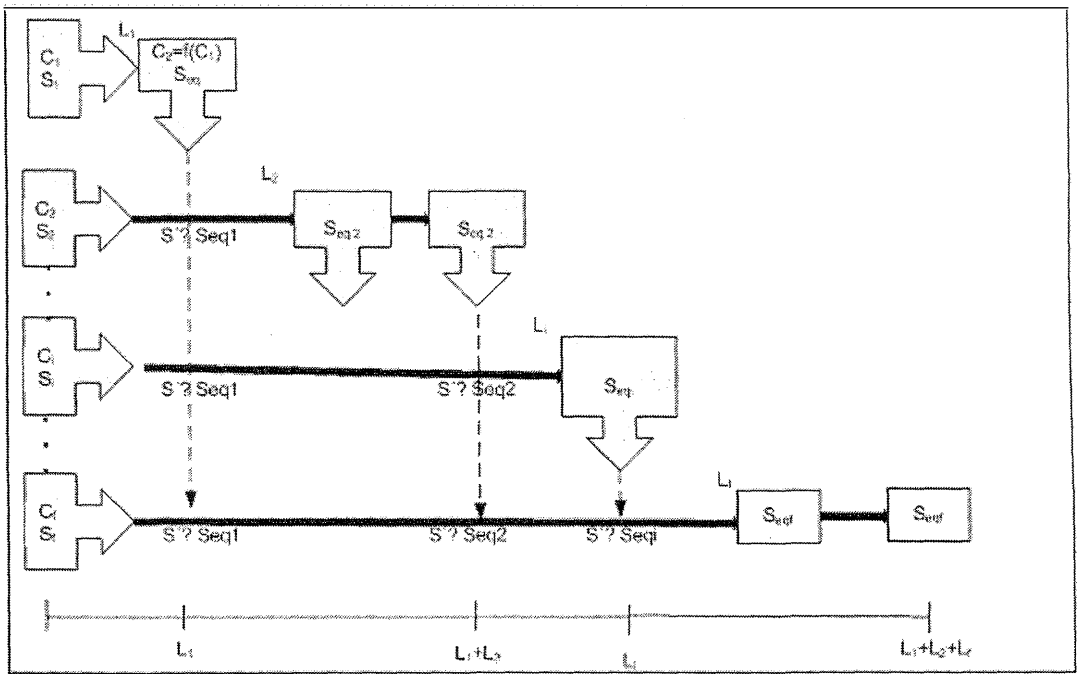

Figura A1-1. Esquema de interacción entre procesadores en el Kernel de MPSA

En el Kernel de MPSA un procesador no puede dar por finalizado su trabajo sino hasta que hayan terminado todos los procesadores trabajando a temperatura superior pues requiere de las soluciones de éstos para saber si debe reiniciar y rehacer su cadena de Markov. En la Figura A1-1 se observa que el procesador trabajando a temperatura C□ reinicio su cadena de Markov tomando ahora como solución inicial la enviada por el procesador trabajando a C□, por lo que en lugar de terminar al tiempo TIEMPO (L□) terminó al tiempo TIEMPO (L□+L□),de esta forma los procesadores trabajando a temperaturas inferiores necesitaron esperar a que éste terminara su proceso para saber si deberían de reiniciar sus cadenas, caso que se cumplió con el procesador trabajando a temperatura Cf, de manera que el tiempo final de ejecución del algoritmo fue TIEMPO(L□+L□+Lf). De esta forma el tiempo de ejecución de un algoritmo MPSA es el requerido para ejecutar la secuencia de cadenas de Markov de longitud máxima que se establece durante el proceso:

$$MPSA_{tiempo} = TIEMPO(MÁX.(Long(i))), \qquad \text{para todo } i \in \Phi$$

Donde Φ es el conjunto de secuencias de cadenas de Markov establecidas durante el proceso y Long es una función que determina la longitud de una secuencia.

En MPSA la comunicación debe ser asíncrona, de manera que los procesadores se mantienen trabajando sin perder tiempo esperando por retroalimentación, ellos simplemente toman ésta al momento en que llega. Una excepción es cuando un procesador ha terminado su cadena y alguno o algunos procesadores están aún trabajando a temperaturas superior. En este caso, el primer procesador envía su solución a todos los procesadores trabajando a temperaturas inferiores y espera para recibir la solución de alguno de los procesadores trabajando a temperaturas superiores. De esta forma, MPSA no descarta el hecho de que pueda ser establecida una longitud fija, L, de cadena de Markov, simplemente los procesadores terminarán de construir sus cadenas y esperarán por las soluciones de los procesadores trabajando a temperaturas superiores, para determinar si deben rehacer o no su cadena de Markov.

Algunos criterios de evaluación que podrían usarse cuando una solución es recibida son: a)si ésta es mejor que su solución de inicio, b) si ésta es mejor que su solución actual, c) si el costo de ésta es mejor que el costo promedio, etc. Por otro lado algunos de los criterios de mejoría o aceptación o rechazo de una solución podrían ser: a) aquélla que sea más probable según la función de distribución de probabilidades que se está ajustando, b) aquélla que posean un valor de costo superior, c) la aplicación del criterio de aceptación del algoritmo tipo recocido simulado que se esté paralizando, d) una combinación de los criterios anteriores dependiendo de la temperatura, etc.

A través de este esquema de comunicación e interacción entre procesadores es difícil evaluar el tiempo de ejecución de un algoritmo MPSA, pues depende de los reinicios que se establezcan. Sin embargo, a semejanza del análisis de algoritmos sí se puede establecer un peor y mejor caso.

De la figura A1-1 se observa claramente que el peor caso es establecido cuando todos y cada uno de los procesadores (excepto aquel trabajando a temperatura C□) tienen que reiniciar al termino del trabajo del procesador a la temperatura inmediata superior. Esto producirá una secuencia de cadenas de Markov igual a la del algoritmo secuencial. Debido a que no se tiene una sobrecarga de comunicación significativa el tiempo se ejecución de cualquier algoritmo paralelo implementado a través de MPSA para el peor caso debería ser el mismo que el de la versión secuencial.

El mejor caso para un algoritmo paralelo implementado a través de MPSA es cuando ningún procesador requiere reiniciar la construcción de su cadena de Markov, entonces el tiempo de ejecución es únicamente el último ciclo de temperatura.

Ahora bien, el mejor caso únicamente puede darse cuando todos los procesadores o cadenas de Markov inician con soluciones adecuadas que le permitan rechazar todas las soluciones que lleguen de las temperaturas superiores. De esta forma, la determinación de la configuración inicial para cada cadena de Markov en la paralelización del ciclo de temperaturas propuesto por el kernel de MPSA representa una oportunidad de desarrollo de nuevos algoritmos que permitan reducir el tiempo de procesamiento o incrementar la eficacia del proceso.

En general un algoritmo paralelizado a través de MPSA se ejecutará más rápidamente que su versión secuencial [Sanvicente-Sánchez y Frausto-Solís, 2002], pero el SpeedUp y la eficacia paralela del proceso podrán ser incrementados mediante una adecuada determinación de los valores de configuración inicial en cada cadena de Markov.

La determinación de la configuración inicial para cada cadena de Markov intuitivamente puede ser efectuada de las siguientes formas: a) una misma configuración para todos los procesadores (entrada individual), b) diferentes configuraciones, una por procesador (entrada poblacional), c) configuración o configuraciones seleccionadas mediante algún proceso especial, etc.

A semejanza del análisis de eficiencia realizado anteriormente, un análisis similar puede ser efectuado para el trabajo computacional de un algoritmo ATSA secuencial y un algoritmo MPSA paralelo.

Para un algoritmo tipo recocido simulado secuencial (ATSA), en la Figura A1-2 se estableció que los valores L□ del eje inferior indican que se ha generado una cadena de Markov de longitud L□, esto es, se han efectuado L□ ensayos en la construcción de la cadena de Markov í. Dado que en cada ensayo se efectúa una evaluación de la función objetivo y si se considera que el trabajo computacional de un algoritmo ATSA recae principalmente en la evaluación de dicha función, entonces el trabajo computacional de un algoritmo ATSA puede ser calculado como el número de evaluaciones de la función objetivo, o más específicamente como el número de ensayos realizados por un algoritmo ATSA, esto es:

$$ATSA_{trabajo} = \sum_{i=1}^{f} L_i$$

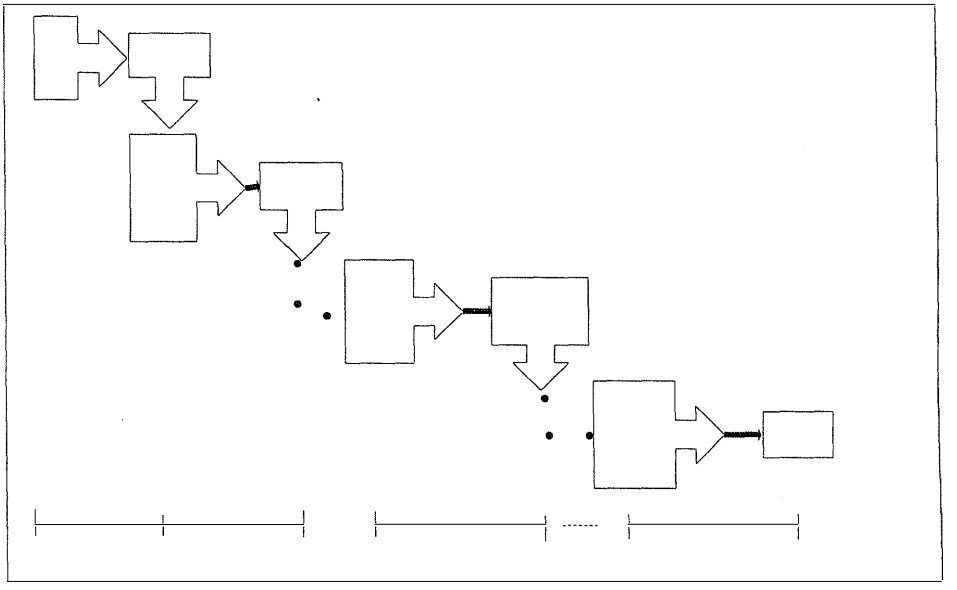

Figura A1-2. Esquematización de un algoritmo tipo recocido simulado (ATSA)

Para un algoritmo MPSA en trabajo computacional depende de los reinicios que se establezcan a lo largo del proceso (ver Figura A1-1). De esta forma, el caso de

menor trabajo computacional es cuando ningún procesador reinicia y entonces el trabajo computacional es igual al algoritmo ATSA secuencial. El caso de mayor trabajo computacional es cuando todos y cada uno de los procesadores tienen que reiniciar al momento de recibir una solución de un procesador a temperatura superior. El trabajo computacional para este último caso es establecido por la ecuación:

$$\sum_{i=1}^{f} \sum_{j=1}^{i} L_j$$

Del análisis anterior se determina que el trabajo computacional de un algoritmo MPSA es en general mayor que el del algoritmo ATSA secuencial, sin embargo la paralelización hace que los tiempos de ejecución del algoritmo MPSA sean en general menores o a lo mucho equivalentes a los del algoritmo ATSA secuencial.

PARALELIZACIÓN VIRTUAL Y NIVELES DE PARALELISMO

El kernel de MPSA establece que cada procesador ejecute únicamente una tarea y que ésta genere una cadena de Markov homogénea a la temperatura así asignada, a esta tarea se le llamará tarea constructora de una cadena de Markov (TCCM). Ahora bien, el kernel de MPSA al paralelizar el ciclo de temperaturas requiere que existan tantos procesadores como ciclos de temperatura se efectúen en el algoritmo secuencial según el esquema de enfriamiento, lo cual permite una paralelización masiva, basado en la convergencia asintótica del algoritmo. Sin embargo, si no se tienen suficientes procesadores esto no puede ser efectuado de manera real y debe recurrirse a un paralelismo virtual [Sanvicente-Sánchez y Frausto-Solís, 2000 y 2002].

Una paralelización virtual puede ser establecida ya sea usando una máquina paralela o una red de área local (LAN por sus siglas en inglés) con sólo unos cuantos procesadores de la siguiente manera:

- Se establece el número de tareas TCCM requeridas por el proceso de recocido, el cual a su vez definirá el número de procesadores necesarios para implementar el kernel de MPSA.
- Se establece el número de procesadores con que se cuenta, ya sea en la máquina paralela o en la LAN.
- Se asigna una tarea TCCM a cada procesador siguiendo el orden descendente de la secuencia de temperaturas.
- Se obtiene el número de tareas TCCM pendientes de ser asignada.
- Se asigna nuevamente una tarea TCCM a cada procesador siguiendo el orden de la primera asignación.
- Se repiten los dos pasos anteriores hasta que ya no queden más tareas TCCM pendiente de ser asignadas.

La Figura A1-3 muestra un ejemplo de una LAN, la cual está ejecutando un algoritmo paralelo tipo recocido simulado implementado a través de MPSA. En la Figura A1-3 se muestra que cada procesador está corriendo tres tareas y que cada tarea está construyendo una cadena de Markov, esto es, cada procesador ejecuta tres tareas constructoras de una cadena de Markov (TCCM).

Una paralelización virtual reduce la eficiencia debido a que las tareas corriendo en un único procesador compiten por los recursos de éste (tiempo de CPU, memoria, espacio en disco, etc.). Además, una paralelización virtual dentro de una LAN necesita tener en cuenta la velocidad de comunicación en la LAN y la carga en la red. Estos dos últimos aspectos generalmente son reducidos en una máquina paralela, pues desde su diseño se considera el contar con canales de comunicación de alta velocidad (High Speed Comunication Bus) y generalmente son máquinas de uso dedicado.

Si un proceso T puede ser dividido en dos tareas paralelas (T1, T2) y si se ejecutan al mismo tiempo en un solo procesador, el tiempo de ejecución de T1 y T2 es igual al tiempo de ejecución del proceso secuencial T más un ligero incremento en tiempo por el manejo del sistema operativo y la comunicación.

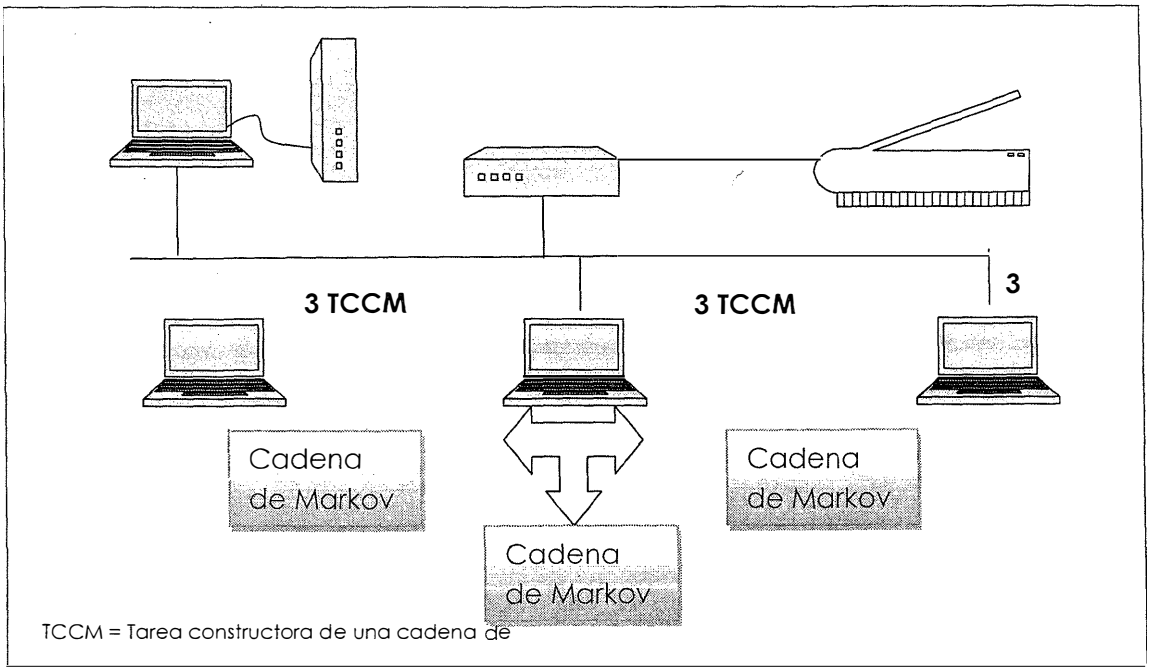

Figura A1-3. Paralelización virtual dentro de una LAN.

Para MPSA una solución a la competencia de recursos del paralelismo virtual es obtener niveles de paralelismo que le permitan establecer la cantidad de trabajo a efectuarse en paralelo [Sanvicente-Sánchez y Frausto-Solís, 2002]. Esto es hecho agrupando las tareas TCCM dentro de una sola tarea llamada grupo de tareas constructoras de una cadena de Markov (GTCCM). Esto significa que en este esquema cada grupo de tareas constructoras de una cadena de Markov ejecuta una porción parcial de la secuencia total de cadenas de Markov en el proceso de recocido. La longitud de la secuencia total de cadenas de Markov en el proceso de recocido es igual al número de temperaturas del esquema de enfriamiento.

Una forma de generar tareas GTCCM de tamaño similar en el sentido de construir igual número de cadena de Markov en cada procesador es: si se tiene un esquema de enfriamiento con C temperaturas, pero únicamente se dispone de P procesadores en lugar de los C procesadores requeridos por el kernel de MPSA (P < C), se debe dividir el número de cadenas de Markov entre el número de procesadores para obtener P secuencias parciales de longitud aproximada N =

int(C / P) de la secuencia total de cadenas de Markov (las secuencias parciales deben tener una longitud entera por lo que el residuo debe ser distribuido equitativamente entre las primeras o las últimas secuencias parciales). Esto es, la secuencia total del recocido es dividida en P secuencias parciales, donde cada secuencia parcial es asignada a un procesador.

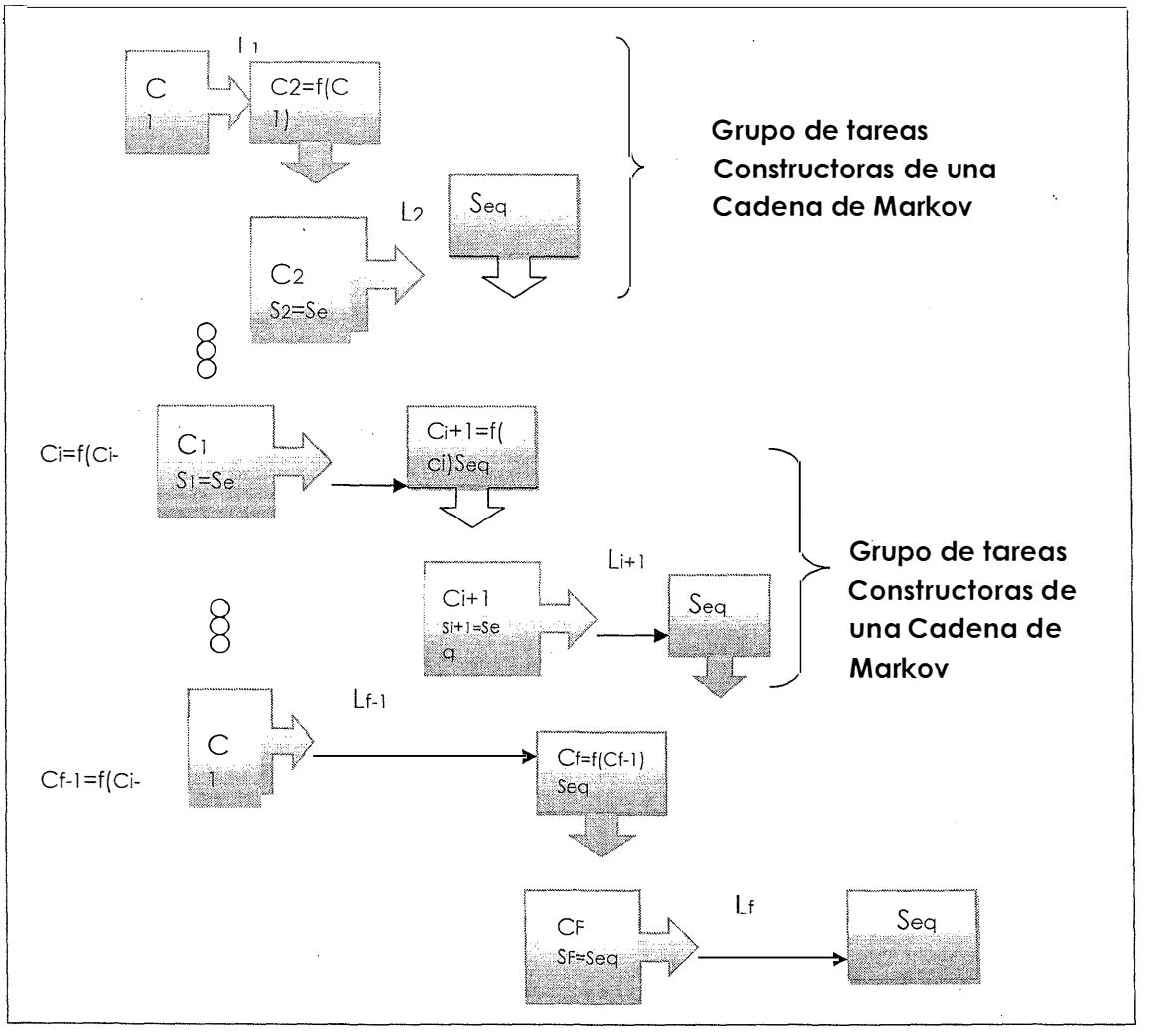

Figura A1-4. Niveles de paralelismo y grupo de tareas constructoras de una cadena de Markov.

Cada procesador en lugar de abrir N tareas TCCM y efectuar un paralelismo virtual que compita por los recursos, agrupa éstas en una sola tarea GTCCM y aprovechando que el GTCCM está conformado por una secuencia parcial del recocido, ejecuta las tareas TCCM del grupo en forma secuencial. De esta forma cada tarea GTCCM debe conocer su solución de inicio, su temperatura inicial, su temperatura final, la función de decremento de temperatura y las longitudes de las cadenas de Markov a construir en su secuencia parcial.

A través de MPSA, cada tarea GTCCM construye su secuencia parcial de cadenas de Markov y envía la solución obtenida al final de ésta (ver Figura A1-4). Cuando una tarea GTCCM necesita reiniciar, ésta construye toda la secuencia parcial de nuevo, como lo establece el kernel de MPSA. Además, un procesador no puede dar por finalizado su trabajo sino hasta que todos los procesadores trabajando a una temperatura superior hayan terminado.

Los niveles de paralelismo ayudan a evitar la competencia por los recursos de la máquina, pues abren únicamente una tarea GTCCM por procesador, pero reducen el paralelismo.

De esta forma el nivel de paralelismo depende del número de tareas GTCCM abiertas. Por ejemplo cuando se abre sólo una tarea GTCCM, la secuencia total de cadenas de Markov del proceso de recocido será asignada a esta tarea y se ejecutará un algoritmo secuencial. Cuando el número de tareas GTCCM abiertas es igual al número de ciclos de temperatura, se obtiene el máximo paralelismo, pues cada tarea GTCCM construirá una sola cadena de Markov, como el kernel de MPSA lo establece, en este caso cada tarea GTCCM es equivalente a una tarea constructora de una cadena de Markov (TCCM).

Los GTCCM pueden ser generados de las siguientes dos formas principales: a) de igual longitud a través de la simple división del número de temperaturas que se establece en el esquema de enfriamiento entre el número de procesadores a usar y b) de longitud variable, donde el usuario indica como se agrupan las tareas TCCM, por ejemplo efectuando una mayor agrupación a altas temperaturas que a bajas temperaturas. La generación de las tareas GTCCM de longitud variable puede ser muy variada y abre al igual que la determinación de la configuración de entrada para cada procesador y de los criterios de evaluación, aceptación o rechazo, cuando una solución es recibida, una línea de investigación que permite la generación de más de un algoritmo MPSA.

Una combinación de estos conceptos, paralelismo virtual y niveles de paralelismo, puede ser efectuada para incrementar la paralelización y explorar las capacidades

de un algoritmo implementado a través de MPSA [Sanvicente-Sánchez y Frausto-Solís, 2002].

Implementación de MPSA

MPSA es una metodología que paraleliza el ciclo de temperaturas de algoritmos tipo recocido simulado que efectúan recocidos secuenciales.

Por recocido secuencial se entenderá la generación (por un solo procesador) de una serie de cadenas de Markov construidas una después de la otra, para una secuencia descendente de un parámetro o temperatura, {Ck}, que controla la forma de la función de distribución estacionaria de cada cadena de Markov, y donde la solución inicial de una cadena es igual a la solución de salida de la cadena inmediata anterior.

MPSA puede ser aplicada a algoritmos tipo recocido simulado secuenciales sin importar las funciones de generación, G(c), y aceptación, A(c), empleadas en la construcción de la cadena de Markov. Además, MPSA puede ser aplicada a los algoritmos paralelos que realicen recocidos secuenciales, para ayudar a efectuar una paralelización masiva del proceso. MPSA es aplicable directamente a los algoritmos Partición de la estructura de Datos (PED), Recocidos Independientes Paralelos (RIP) que ejecutan algoritmos SA secuenciales independientes, dentro de un esquema de pseudo-paralelización. También puede aplicarse a los algoritmos Ensayos de Markov Paralelos (EMP) y Árboles Especulativos (AE) que paralelizan el trabajo del ciclo interno constructor de la cadena de Markov.

La aplicación de MPSA a algoritmos paralelos debe efectuarse exclusivamente cuando se desea un paralelismo masivo que no puede ser efectuado con el algoritmo paralelo original pues se produce un fuerte deterioro en el SpeedUp y la eficiencia paralela o quizá en la calidad de la solución a encontrar con el incremento de los procesadores.

Supóngase que se tiene un algoritmo tipo recocido simulado secuencial que genera N cadenas de Markov de longitud fija L, entonces el tiempo de ejecución de dicho algoritmo

$$t_{sec} = N\ L$$

Ahora bien, supóngase que dicho algoritmo es paralelizado con un algoritmo de árbol especulativo (AE) que corre sobre P1 procesadores en un tiempo tAE produciendo un SpeedUp igual a K. a partir de la definición de SpeedUp se tiene:

$$K = t_{sec} / t_{AE}$$

Entonces

$$t_{AE} = t_{sec}/K = N\ L / K = N\ L'$$

donde $L' = L / K$

Ahora supóngase que sobre este algoritmo paralelo AE se aplica MPSA abriendo P_2 tareas GTCCM que se ejecutan en un tiempo t_{MPSA} produciendo un SpeedUp igual a M, entonces se tiene que

$$M = t_{AE} / t_{MPSA} = N\ L' / t_{MPSA}$$

$$t_{MPSA}\ N\ L' / M = (N\ L) / (M\ K) = t_{sec} / (M\ K)$$

El SpeedUp del algoritmo doblemente paralelizado es entonces (M K), lo cual en primera instancia indica un incremento en la aceleración del algoritmo. Sin embargo, hay que tener en cuenta que el número de procesadores empleados en esta doble paralelización es la multiplicación de P_1 por P_2 y que la eficiencia paralela del proceso igualmente es E_{AE} por E_{MPSA} lo cual indica en términos reales que se tiene un deterioro en el trabajo en paralelo del algoritmo doblemente paralelizado. Ahora bien, en la literatura se reporta que el algoritmo AE posee una gran sobrecarga en la comunicación [Greening, 1990] la cual aumenta conforme el número de procesadores crece lo que reduce rápidamente la eficiencia del

algoritmo AE evitando su paralelismo masivo. De esta forma, una doble paralelización podría ayudar a reducir la caída de la eficiencia del algoritmo AE y efectuar una paralelización masiva. Otros algoritmos reportados en la literatura con una alta sobrecarga en la comunicación que impiden una paralelización masiva son los EMP sobre todo en los ciclos de la temperatura.

Los pasos para paralelizar un algoritmo secuencial general tipo recocido simulado a través de MPSA son los siguientes:

- Analizar el código o pseudocódigo general del algoritmo tipo recocido simulado.
- Esquematizar el algoritmo tipo recocido simulado a través de su ciclo de temperaturas, simplificando el ciclo interno o constructor de la cadena de Markov por una línea recta (los procesos o comunicaciones existentes para construir una cadena de Markov no son alterados por MPSA).
- Esquematizar el algoritmo tipo recocido simulado a través del kernel de MPSA (iniciando todos los ciclos de temperatura al mismo tiempo y estableciendo la comunicación de MPSA)
- Esquematizar el algoritmo tipo recocido simulado a través de niveles de paralelismo Determinar si el número de niveles de paralelismo es:
    - Fijo (siempre se van a abrir el mismo número de tareas GTCCM)
    - Variable (establecido por el usuario, por ejemplo dependiendo del número de procesadores disponibles)
- Establecer la longitud de los GTCCM cómo:
    - Igual
    - Variable
- Si las longitudes de los GTCCM son variables como va a ser determinada:
    - Fijados con anterioridad a través de porcentajes.
    - Establecidos por el usuario interactivamente
- Determinar el tipo de entrada del algoritmo:
    - Individual.

- Poblacional (una entrada se considera poblacional cuando se tienen dos o más soluciones diferentes).

- Si la entrada es individual determinar cómo se producirá ésta:
  - Aleatoria.
  - Recocido simulado recortado.
  - A través de un procesador de selección especial (otro algoritmo heurístico o de búsqueda local).
- Si la entrada es poblacional determinar cómo se obtendrá ésta:
  - Cada procesador efectúa su propia selección aleatoria.
  - Población a través de un muestreo representativo y extracción aleatoria a partir de la población.
  - Población a través de un muestreo representativo y extracción Monte Carlo a partir de la función de distribución al inicio de cada cadena. Comenzando de la temperatura mayor a la menor.
  - Muestreo aleatorio Monte Carlo para cada temperatura y selección de la solución más probable en cada muestreo según la función de distribución al inicio de cada cadena.
  - Muestreo aleatorio Monte Carlo para cada temperatura y selección usando la función de distribución al inicio de cada cadena de la solución más probable a partir de la unión del muestreo propio y los muestreos de los procesadores trabajando a temperaturas superiores.
  - Selección a partir de un reconocido simulado recortado.
  - A través de procesos de selección especiales (otros algoritmos heurísticos).
- Determinar si la entrada del algoritmo va ser en línea o fuera de línea:
  - En línea como un proceso previo al recocido y con procesos en paralelo en el caso de entrada poblacional.
  - Fuera de línea con un proceso de sintonización y asignación mediante un archivo de datos.

- Determinar el criterio de evaluación de contra que compara una solución de llegada. Entre los criterios se consideran:
    - Contra la solución actual del procesador.
    - Contra la solución de inicio de la cadena o tarea GTCCM.
    - Costo de la solución contra costo promedio de la cadena siendo construida.
- Determinar los criterios de aceptación o rechazo de una solución de llegada, pudiendo considerar:
    - Mayor probabilidad según la función de distribución de probabilidades al inicio de cada cadena de Markov.
    - Mayor probabilidad según la función de distribución estacionaria de cada cadena de Markov.
    - Mejor costo.
    - Aplicación del criterio de aceptación del algoritmo tipo recocido simulado que se esté paralelizado.
    - Una combinación de los criterios anteriores dependiendo de la temperatura.
- Si el impacto principal del algoritmo que se está paralelizando es el esquema de enfriamiento, obtener éste (temperatura inicial, función de decremento de temperatura, temperatura final o criterio de paro, longitud de la cadena de Markov o criterio de equilibrio) y analizarlo para determinar si es compatible con MPSA. Esto es, poder establecer de antemano la secuencia de temperaturas del proceso, ya sea mediante sintonización o por conocimiento previo de la temperatura inicial, la temperatura final y la función de enfriamiento con todos sus parámetros.
- Si es un algoritmo que no posee un esquema de enfriamiento específico establecerle un esquema de enfriamiento compatible con MPSA (el Capítulo 5 presenta un nuevo esquema de enfriamiento que es totalmente compatible con MPSA).
- Llevar a cabo la codificación o el pseudocódigo del algoritmo general tipo recocido simulado paralelizado a través de MPSA.

Cabe recordar que dependiendo de la entrada asignada al algoritmo MPSA y de las reglas de aceptación o rechazo de una solución enviada por otro procesador dependerá el SpeedUp y la eficiencia paralela que pueda alcanzarse en el proceso. Así como el hecho de que cualquier combinación de tipos de entrada con reglas de aceptación o rechazo producirá un nuevo algoritmo paralelo tipo MPSA.

Los pasos para paralelizar un algoritmo específico tipo recocido simulado existente a través de MPSA son los siguientes:

- Dado que el algoritmo existe, analizar el código o el pseudocódigo del algoritmo.
- Determinar pseudocódigo y el código del algoritmo general tipo recocido simulado.
- Paralelizar pseudocódigo y el código del algoritmo general a través de MPSA, considerando lo más adecuado para la determinación de su entrada y para la interacción entre cadenas de Markov según las características propias del problema específico que se está resolviendo.
- Codificar o implementar el algoritmo específico paralelizado a través de MPSA.

Los pasos para crear un algoritmo paralelo específico a través de MPSA son:

- A partir de un algoritmo general tipo recocido simulado llevar a cabo su paralelización a través del kernel de MPSA o de los niveles de paralelismo de MPSA.
- Determinar el tipo de entrada que se desea y su forma de obtenerla considerando las características propias del problema que se está resolviendo.
- Determinar el tipo de evaluación que se desea considerando las características propias del problema a resolver.
- Codificar o implementar el algoritmo paralelo específico creado.

Cualquier algoritmo general tipo MPSA puede a su vez ser implementado dentro de un tipo de problema específico y resolver cualquier instancia de éste, aunque las reglas para determinar la entrada y de aceptación o rechazo sean generales.

Notación de algoritmos MPSA

MPSA es una metodología que permite la paralelización de algoritmos tipo recocido simulado, los cuales a su vez pueden ser generales o específicos para solucionar un problema. Además, MPSA requiere de algunas especificaciones como son: el tipo de entrada sobre la que se va a trabajar, la forma como se generará esa entrada, y los criterios de evaluación o rechazo de una solución durante la interacción de los procesadores. Finalmente, al momento de ejecutar un algoritmo tipo recocido simulado se requiere conocer la instancia que se está resolviendo y los valores de los parámetros de sintonización.

Como una forma rápida de informar los detalles principales de una algoritmo desarrollado a través de MPSA se crea una notación que especifica éstos en el nombre del algoritmo. La notación posee una estructura especial integrada de campos separados por una diagonal.

Dado que MPSA es la metodología a partir de la cual se está paralelizando un algoritmo tipo recocido, el nombre de dicho algoritmo debe ser prefijado con un campo conteniendo la palabra MPSA. De esta manera si se está paralelizado un algoritmo SA, su nombre cambiará á:

MPSA/SA

Si lo que se está generando es un algoritmo general, la palabra GENERAL deberá anexarse a través de un campo sufijo, pero si lo que se está desarrollando es un algoritmo específico para un problema, el nombre del problema se anexará en dicho campo sufijo, por ejemplo:

MPSA/SA/GENRAL o MPSA/SA/TSP

Hasta aquí se considera que se tiene un nombre que informa qué algoritmo fue paralelizado a través de MPSA, por lo que los campos mostrados hasta el momento se establecen como obligatorios. Sin embargo, MPSA puede trabajar con dos tipos de entrada (individual y poblacional) lo cual también podría ser informado opcionalmente en el nombre, por ejemplo:

MPSA/SA/TSP/Individual o MPSA/TSP/Poblacional

El uso de las mayúsculas y minúsculas indica que esa es una información opcional.

Otros campos opcionales pueden ser la forma como se genera la entrada y los criterios de evaluación o rechazo de una solución. Todos los campos opcionales se agregarán a la derecha de los campos obligatorios, sin embargo, agregar varios campos opcionales puede producir una descripción muy larga para un nombre por lo que deberían ser agregados únicamente los más importantes y en su defecto ser sustituidos mediante algún campo opcional breve que indicará un nombre particular del algoritmo o su objetivo. Por ejemplo:

MPSA/SA/TSP/ Recalentamiento

Ahora bien, cuando se ejecute dicho algoritmo algunos detalles de la ejecución como por ejemplo el nombre de la instancia solucionada, y algún parámetro propio del algoritmo puede ser indicado entre paréntesis.

APÉNDICE 2

Notas sobre OpenMP.

OpenMP es un conjunto de librerías para C y C++, regidas por las especificaciones ISO/IEC (*International Standard Organization / International Engineering Consortium*), basado en el uso de directivas para ambientes paralelos.

Especificación de 1998. Versión 2.0 (f90) en 2000, C. C++ en 2002. versión 2.5 (Fortran+C/C++) en 2005. Acordado por una serie de fabricantes de hardware y software y usuarios.

- API destinado a máquinas de memoria compartida.
- Entra en el programa fuente esencialmente en forma de directivas de compilación.
- Bindings para F77, f90, C y C++.
- Permite una paralelización "incremental".

OpenMP tiene soporte para diferentes sistemas operativos como UNIX, Linux y Windows.

Un programa escrito con OpenMP inicia su ejecución, en un solo hilo activo, llamado maestro. El hilo maestro ejecuta una región de código serial antes de que la primera construcción paralela se ejecute. Bajo el API de OpenMP la construcción paralela se obtiene mediante directivas paralelas. Cuando se encuentra una región de código paralelo, el hilo maestro crea (*fork*) hilos adicionales, convirtiéndose en el líder del grupo. El hilo maestro y los nuevos hilos ejecutan de forma concurrente la sección paralela (realizan trabajo compartido). Unirse al grupo (*join*) es el procedimiento donde al finalizar la ejecución de la región de código paralelo los hilos adicionales se suspenden o se liberan, y el hilo maestro retoma el control de la ejecución. Este método se conoce como **fork & join**.

Un lenguaje de programación en paralelo debe tener tres aspectos fundamentales para llevar a cabo la programación en paralelo:

- Ejecución paralela especifica
- Comunicación entre múltiples hilos
- Sincronización entre los hilos

Toma directivas base que se acercan a la programación en paralelo. Esto consiste en una serie de directivas que pueden ser usadas en base a un lenguaje de programación tal como Fortran, C o C++.

La diferencia entre el modelo de paso de mensajes y el modelo de memoria compartida, se debe al número de procesos o hilos activos. En un modelo de paso de mensajes, durante la ejecución del programa, todos los procesos se encuentran activos. Por otra parte, en un modelo de memoria compartida, sólo existe un hilo activo al iniciar y al finalizar el programa; sin embargo, durante la ejecución del programa, la cantidad de hilos activos puede variar de forma dinámica (ver figura A2-1).

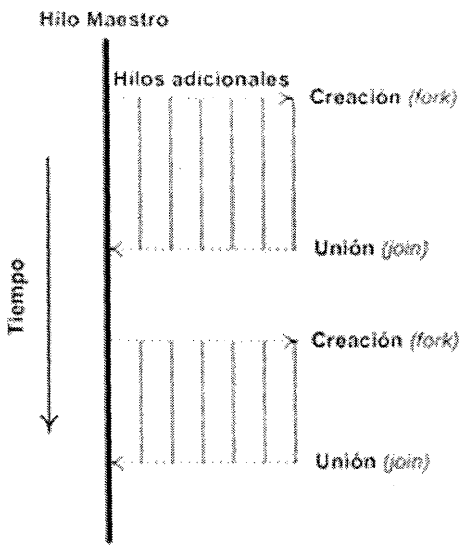

Figura A2-1 Modelo de ejecución fork & join

Modelo de Memoria de OpenMP

• Memoria compartida, consistencia relajada. Además de la memoria central compartida, cada hilo tiene una vista temporal de la memoria compartida (almacenamiento temporal privado, cacheable) y un espacio privado (threadprivate), no accesible desde otros hilos.

• Si un hilo se convierte en máster de un equipo de hilos, sus variables privadas serán visibles por su equipo a no ser que se declaren de nuevo privadas. La vista temporal de la memoria del hilo no tiene que ser siempre consistente con la memoria central. Se puede asegurar esto con FLUSH.

Estructura OpenMP

• OpenMP entra en un programa a través de directivas, con funciones propias (que pueden ser condicionalmente compiladas) y variables de entorno, que también pueden modificar el comportamiento en tiempo de ejecución.

Variables de entorno

• Establecidas en el shell que comienza la ejecución, ignoradas una vez ha comenzado ésta. Las variables siempre en mayúsculas, sus argumentos dan igual.

OMP_SCHEDULE: Controla la distribución del trabajo en los bucles. Puede ser static, dynamic o guided y admite un parámetro numérico que especifica, por ejemplo, el tamaño de bloque a repartir en una distribución estática.

• El valor por omisión es dependiente de la implementación.

• Ej.: setenv OMP_SCHEDULE static,100

OMP_NUM_THREADS: Controla el número de hilos de ejecución en los que se abre una región paralela.

OMP_DYNAMIC: Controla dinámicamente el número de hilos de ejecución en los que se expande una región paralela para maximizar el uso de recursos. Puede ser true o false. Valor por omisión dependiente de la implementación.

OMP_NESTED: true o false Controla la expansión de un hilo de ejecución en un equipo de hilos en regiones paralelas anidadas. El valor por omisión es false.

- Se pueden alterar en tiempo de ejecución los comportamientos que regulan estas variables con llamadas a la librería. Ej.: omp_set_dynamic,

omp_set_num_threads.

Directivas de OpenMP

- Región paralela: PARALLEL
- Distribución de trabajo: DO, SECTIONS, SINGLE, WORKSHARE
- Sincronización: MASTER, CRITICAL, BARRIER, ATOMIC, FLUSH, ORDERED.
- Entorno de datos: THREAD PRIVATE
- Alcance de datos: PRIVATE, SHARED, DEFAULT, FIRSTPRIVATE, LASTPRIVATE,

REDUCTION, COPYIN

Las implementaciones de OpenMP en C++ proporcionan un archivo estándar llamado ***omp.h***, este proporciona a OpenMP un tipo de definiciones y prototipos de función de biblioteca. Este archivo o librería debe ser incluido por C y C++.

Librerías de OPENMP en C/C++

void opm_set_num_threads (int): Colocar el número de hilos para usar en un equipo.

int omp_get_num_threads (void): Regresa el número de hilos en la región de la ejecución en paralelo actual.

int opm_get_max_threads (void): Regresa el valor máximo que la llamada a omp_get_num_threads pueda regresar. Este valor cambia con la llamada a omp_set_num_threads.

int opm_get_thread_num (void): Regresa el número de hilos dentro del equipo, inclusive un valor dentro del intervalo 0 y omp_get_num_threads() -1. Hilo maestro es 0.

int omp_get_num_procs (void): Regresa el número de procesos disponibles para el programa.

void omp_set_dynamic (int): Habilita/Deshabilita arreglos dinámicos del número de hilos usados por un constructor paralelo.

int omp_get_dynamic (void): Regresa .TRUE. si los hilos dinámicos son habilitados y .FALSE. lo contrario.

int omp_int_parallel (void): Regresa .TRUE. si esta función es invocada desde adentro de la extensión dinámica de la región paralela, y .FALSE. lo contrario.

void omp_set_nested (int): Habilita/Deshabilita paralelismo anidado. Si es deshabilitado, entonces las regiones en paralelo anidadas están seriadas.

int omp_get_nested (void) : Regresa .TRUE. Si es el paralelismo anidado es habilitado y .FALSE. Lo contrario

GLOSARIO DE TÉRMINOS

Algoritmo Secuencial: Algoritmos que puede ejecutarse en un solo procesador.

Algoritmo Paralelo: Algoritmo que permiten la división de un problema en subproblemas de forma que se puedan ejecutar de forma simultánea en varios procesadores.

Algoritmo determinístico: En términos informales, es un algoritmo completamente predictivo si se conocen las entradas al mismo siempre producirá la misma salida, y la máquina interna pasará por la misma secuencia de estados.

Algoritmo no deterministico: En ciencias de la computación, es un algoritmo que con la misma entrada ofrece muchos posibles resultados. No se puede saber de antemano cuál será el resultado de la ejecución de un algoritmo no determinístico.

Algoritmo Genético: Algoritmos que se basan en un modelo abstracto de la evolución natural, tal que la calidad de las estructuras individuales al nivel mas alto son compatibles con el ambiente en el que operan, lo que en los problemas de computación corresponden a las restricciones del propio problema a resolver.

Benchmark: Es una técnica utilizada para medir el rendimiento de un sistema o componente de un sistema, frecuentemente en comparación con el cual se refiere específicamente a la acción de ejecutar un benchmark. La palabra *benchmark* es anglicismo traducible al castellano como comparativa. Para los problemas que tienen que ver con la asignación de tareas en talleres de manufactura existe un conjunto de problemas utilizados para medir el rendimiento de los algoritmos propuestos.

Búsqueda Tabú: es un procedimiento de búsqueda orientado determinista simple, el cuál restringe la búsqueda a fin de optimizar la búsqueda local a través del almacenamiento de la historia del proceso en su memoria

Colonia de Hormigas: son algoritmos inteligentes utilizados para resolver problemas de optimización en general, donde el estudio desde el punto de vista biológico de una colonia de hormigas presenta que su comportamiento es altamente estructurado y además se conoce que una hormiga tiene capacidades limitadas

Costo de operación : Duración en tiempo para una operación en particular en los problemas tipo JSSP y FJSSP.

Estructura de Vecindad: determina el sistema de las soluciones de vecindad que se pueden alcanzar de la solución actual realizando un movimiento o una transición.

Flexible Job Shop Scheduling Problem (FJSSP): Es una derivación del problema JSSP y cuya característica principal es que para una operación existe más de una máquina que la pueda procesar.

Job Shop Scheduling Problem (JSSP): Problema de asignación de tareas en talleres de manufactura, donde el principal problema es encontrar la mejor distribución de la carga de trabajo utilizando recurso que generalmente son tiempos en m máquinas para procesar n trabajos, cada uno de ellos con un cierto número de operaciones en un orden lógico de precedencia.

Makespan : Tiempo máximo de terminación (el tiempo de inicio más la duración) de la última operación asignada en una máquina en todo el proceso.

Máquinas CNC: máquinas que automáticamente realizan las operaciones requeridas de acuerdo a un set detallado de instrucciones codificadas para la máquina y su operación en lotes

Metaheurísticas : Es un método heurístico para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficiente.

Modelo de programación entera mixta (MIP): Los problemas de calendarización pueden ser resueltos de manera óptima utilizando técnicas de programación matemática y una de las formas más comunes de programación matemática para problemas tipo JSSP Y FJSSP es la programación entera mixta. El formato de la programación entera mixta es simple, ya que solamente se requiere tener una función objetivo lineal, que es minimizar el tiempo total de procesamiento de todas las operaciones dadas (Cmax), en este modelo sobresalen dos conjuntos de restricciones a considerar por su importancia, de las cuales uno de los conjuntos tiene variables enteras binarias (de decisión) que son utilizadas para implementar las restricciones disyuntivas.

Modelo de Grafos disyuntivo: Es un modelo propuesto que permite representar problemas de decisión tipo JSSP y FJSSP mediante nodos y arcos, donde los arcos conjuntivos representan las restricciones de precedencia u orden lógico en que las operaciones tienen que procesarse y los arcos disyuntivos representan el orden en que las operaciones asignadas en una máquina en particular serán procesadas (restricción de capacidad de recursos).

NP: En teoría de la complejidad computacional, NP es el acrónimo en inglés de Polinómico no determinista (Non-Deterministic Polynomila-time). Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista.

NP-duro: En teoría de la complejidad computacional, la clase de complejidad NP-Duro o NP-Hard es el conjunto de los problemas de decisión que contiene los problemas H tales que todo problema L en NP puede ser transformado polinomialmente en H.

Recocido Simulado (SA). Es un algoritmo que realiza una búsqueda aleatoria orientada, el cual fue introducido como una analogía de la física estadística del proceso simulado de una fundición de metal hasta que su energía mínima se alcanza, por lo que las configuraciones son análogas a los estados de un sólido, mientras que el costo de la función f y el parámetro de control c son equivalentes a la energía y a la temperatura respectivamente.

Setup time: Múltiples periodos de mantenimiento para algunas máquinas, que pueden corresponder a la preparación de una máquina para recibir una operación, o para intercambiar una operación y otra en la misma máquina o al finalizar los trabajos, se

adiciona un tiempo para la limpieza de las máquinas y dejarlas listas para el siguiente proceso. Para los periodos de intercambio de una operación y otra en una misma máquina, cada uno de ellos es caracterizado por un tiempo de inicio y un tiempo final del proceso mismo más el tiempo de intercambio entre operaciones sucesivas en una misma máquina

Sistemas de Manufactura Flexible (SMF): Sistemas que se componen de máquinas con diferentes capacidades para realizar operaciones en los procesos de producción.

Sintonización: Proceso que permite encontrar los valores ideales de las variables de entrada para un algoritmo a fin de que el comportamiento a fin de su rendimiento sea mejor.

Tiempo polinomial: En las ciencias computacionales, cuando el tiempo de ejecución de un algoritmo (mediante el cual se obtiene una solución al problema) es menor que un cierto valor calculado a partir del número de variables implicadas (generalmente variables de entrada) usando una fórmula polinómicas se dice que dicho problema se puede resolver en un tiempo polinómico.