UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA

"SISTEMA COMPUTACIONAL PARA LA EVALUACIÓN DE ALGORITMOS DE PLANIFICACIÓN DE TRABAJOS EN UN TALLER DE FLUJO HÍBRIDO"

TESIS

QUE PARA OBTENER EL GRADO DE
Maestro en Ciencias

PRESENTA

Rainier Romero Parra

DIRECTOR

Dra. Larisa Burtseva

Mexicali, B. C. **Diciembre de 2007**

RESUMEN

SISTEMA COMPUTACIONAL PARA LA EVALUACIÓN DE ALGORITMOS DE PLANIFICACIÓN DE TRABAJOS EN UN TALLER DE FLUJO HÍBRIDO

La publicación de numerosos algoritmos para la planificación de trabajos requiere conocer cual es el mejor. La evaluación de un algoritmo se realiza a través de un experimento computacional. En la tesis, se propone el sistema computacional PLARETF que permite: analizar, calibrar y comparar algoritmos que resuelven problemas de planificación de trabajos. El diseño del sistema se basa en los parámetros de experimentación recibidos a través del análisis de recientes publicaciones en el área. El sistema permite contener una colección de algoritmos, que utilizan una entrada genérica para problemas con distintas restricciones del modelo. Los resultados obtenidos por cada algoritmo se entregan en un formato especial de salida para su análisis posterior.

La aprobación del sistema se demuestra en un experimento computacional. Se desarrolla una metodología para la utilización del sistema, la cual se expone mediante un ejemplo de la calibración de un algoritmo con métodos estadísticos de análisis de varianza, y la comparación de dos algoritmos con respecto a sus resultados.

ABSTRACT

COMPUTER SYSTEM FOR THE EVALUATION OF ALGORITHMS FOR SCHEDULING JOBS IN A HYBRID FLOWSHOP

The publication of numerous algorithms for planning work requires knowing which is best. The evaluation of an algorithm is performed via a computational experiment. In the thesis, the computer system PLARETF is proposed to analyze, measure and compare algorithms that solve scheduling jobs problems. The system design is based on the parameters of experimentation received through analysis of recent publications in the area. The system allows to contain a collection of algorithms, which use a generic entry for problems with various restrictions of the model. The results obtained by each algorithm are delivered in a special format output for subsequent analysis.

The approval of the system is demonstrated in a computing experiment. It develops a methodology for using the system exposed through an example of a calibration algorithm with statistical methods of analysis of variance, and comparison of two algorithms with regarding its results.

Agradecimiento

Dedicatoria

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

1 INTRODUCCIÓN

1.1 Definición del problema

La planificación de los trabajos se enfoca en la asignación de actividades a recursos escasos con el objetivo de optimizar una o más medidas del resultado. Dependiendo de la situación, los recursos toman distintas formas, como máquinas en una planta de producción o ensamblaje, CPU, memoria y dispositivos de Entrada/Salida en un sistema computacional, pistas de un aeropuerto, etc. Como actividades se entienden distintas operaciones tecnológicas en manufactura, ejecución de un programa computacional, aterrizaje y despegue en un aeropuerto, etc. El objetivo de la planificación es la optimización de un criterio: la minimización de la fecha máxima de terminación de un pedido en manufactura, minimización de trabajos terminados después de alguna fecha límite, minimización del total de la tardanza ponderada de trabajos, entre otros. Este tipo de problemas tiene naturaleza combinatoria.

Los problemas de optimización combinatoria se extienden a distintas áreas como el comercio, la economía, la industria, la ingeniería o la medicina. A menudo estos problemas son difíciles de resolver. Muchos pertenecen a la clase $\mathcal{NP}$-duros, lo que implica que no existe algoritmo conocido que los resuelva en un tiempo polinomial. La cantidad de problemas de este tipo sigue en aumento, por lo tanto se requiere el desarrollo de nuevos algoritmos y métodos para solucionarlos.

La problemática de la planificación de los trabajos apareció entre las investigaciones hace muchos años. Se han propuesto soluciones que van desde los diagramas, que Henry L. Gantt desarrolló a principios del siglo XX, hasta las técnicas metaheurísticas recientes. La formación de la teoría vio sus orígenes en los años 50, con

la publicación del primer algoritmo exacto, denominado "Algoritmo de Johnson para dos máquinas" (Johnson, 1954). A partir de las investigaciones científicas en esta área surgieron numerosos algoritmos, tanto exactos como aproximados para distintos problemas provenientes de la práctica.

Con la publicación de algoritmos, que resuelven el mismo problema con distintos métodos, surgió el problema de su comparación entre sí con el propósito de escoger el más exacto y eficiente, es decir, aquel que se acerca más a la solución óptima y es más rápido. Existen dos caminos para evaluar la cercanía de la solución obtenida por un algoritmo al óptimo. Estos son: 1) el análisis teórico, el cual consiste en la obtención de la razón entre el resultado proporcionado por el algoritmo y el óptimo y 2) un experimento computacional. La segunda opción es la técnica común en consecuencia de la disponibilidad de potentes sistemas computacionales capaces de solucionar problemas de alta complejidad computacional, a los cuales pertenece la mayoría de algoritmos para la planificación de los trabajos.

Para obtener conclusiones válidas acerca de la calidad de un algoritmo, su comportamiento debe ser analizado y comparado con otros algoritmos en condiciones homogéneas, como: sistemas de recursos comparables, restricciones iguales de los modelos, el mismo rango y la distribución de datos de entrada, de preferencia, los mismos datos de entrada, etc. Además es necesario tomar en cuenta la capacidad del sistema computacional que realiza el experimento. Un experimento bien diseñado permite obtener las características del algoritmo, tanto absolutas, como relativas, y evaluar numéricamente la significancia de los factores que intervienen en el modelo.

La realización de un experimento requiere la creación de un sistema computacional para procesar los algoritmos a comparar. Este sistema debe poseer ciertas características, como un generador específico de datos pseudoaleatorios, el formato generalizado de datos de entrada y salida, independencia relativa de los algoritmos implementados en el sistema como módulos internos.

El problema que resuelve la presente Tesis de Maestría es la creación de tal sistema denominado PLARETF (Planificador de Recursos en Talleres de Flujo) y su aprobación mediante la implementación de dos algoritmos que resuelven el mismo problema: un

algoritmo para la calibración, y evaluación de su exactitud y complejidad a través de la comparación con un algoritmo de referencia.

1.2 Objetivos

Objetivo general

Diseñar y desarrollar un sistema computacional para implementación de algoritmos de planificación de los trabajos en un Taller de Flujo Híbrido (TFH) que permita evaluar la exactitud y complejidad de estos a través del análisis estadístico comparativo.

Objetivos particulares

Crear el sistema computacional que permita implementar distintos algoritmos para un TFH basándolo en dos algoritmos: uno para análisis y otro como referencia.

Diseñar y realizar el experimento computacional para la calibración del algoritmo bajo estudio y la evaluación de su exactitud y complejidad relativa con respecto al algoritmo de referencia para verificar el funcionamiento del sistema computacional.

1.3 Metodología

La metodología utilizada en esta investigación incluye:

- El estudio de métodos para solucionar el problema de TFH.
- El diseño de la estructura del sistema computacional destinado a la evaluación de la exactitud y complejidad de algoritmos de planificación de trabajos en un TFH, tomando en cuenta:
    - La revisión de métodos de evaluación de la exactitud y complejidad de algoritmos.
    - El ajuste del formato de los datos de entrada conforme a un patrón de datos.
    - La definición de estructura de datos de salida.
    - La implementación de dos algoritmos para la revisión del funcionamiento del sistema: un algoritmo heurístico para un TFH con dos etapas (TFH2) y un algoritmo genético como el algoritmo de referencia.

- La elección e implementación de algoritmos de asignación de los trabajos a las máquinas para el algoritmo heurístico.
- La generación de datos de salida.

- El diseño y desarrollo del experimento computacional para análisis comparativo estadístico de dos algoritmos bajo estudio, incluyendo:
  - El análisis de metodologías de experimentación usadas para la evaluación de algoritmos.
  - La elección de variables del experimento y sus rangos.
  - La elección de los factores, los cuales influyen en los resultados del algoritmo heurístico.
  - La calibración del algoritmo heurístico y la elección de los mejores valores de los factores que influyen al resultado.
  - La comparación del algoritmo heurístico con el algoritmo de referencia y la extracción de las conclusiones acerca de exactitud y eficiencia de los algoritmos.
- La extracción de conclusiones acerca del funcionamiento del sistema.

El esquema general de la metodología de investigación se muestra en la Figura 1.1.

Figura 1.1. Metodología de la investigación

1.4 Esquema general de la Tesis

El capítulo 2 contiene las definiciones de los términos y notaciones básicas de la Teoría de Planificación (*Scheduling Theory*). Se presentan varias clasificaciones de sus modelos. Se describe una síntesis de la Teoría de Complejidad, la cual proporciona herramientas analíticas para descubrir la dificultad intrínseca de los problemas. A continuación se enseñan varios modelos de TFH. Además, se citan algunas investigaciones enfocadas en el tema. Posteriormente se describen los métodos para la planificación de trabajos en un TFH, acompañándolos con el análisis bibliográfico: diversas heurísticas y metaheurísticas, entre las cuales se encuentran algoritmos constructivos, búsqueda local, recocido simulado, algoritmos evolutivos y colonia de hormigas. Posteriormente se analice el concepto de dicotomía y se describe el modo de su aplicación para la optimización combinatoria. El capítulo se finaliza con la descripción de un algoritmo heurístico para minimización de la fecha máxima en un TFH con dos etapas (TFH2) denominado M+m, y las conclusiones acerca de complejidad del problema bajo estudio y métodos de su solución.

El capítulo 3 describe el diseño de un sistema computacional para la comparación de algoritmos. El capítulo comienza con el análisis bibliográfico de los métodos de evaluación de la calidad de algoritmos desarrollados por distintos autores y parámetros de los experimentos utilizados. Se describe el esquema general y las estructuras del sistema computacional PLARETF para el análisis y comparación de algoritmos que resuelven problemas de TFH y se explican cada una de las partes que lo componen. El sistema se desarrolla para el análisis de algoritmos de la planificación de tareas en los TFH2. El capítulo se termina con las conclusiones acerca de la estructura del sistema computacional.

En el capítulo 4 se describe la aprobación del sistema PLARETF en el experimento computacional para la calibración del algoritmo heurístico M+m y evaluación de su exactitud y eficiencia con respecto al algoritmo de referencia. En un principio se presentan los parámetros de los algoritmos a evaluar y un algoritmo original para la fusión de subpermutaciones, recibidas en el proceso de ejecución del algoritmo heurístico M+m en una permutación. Posteriormente se exponen los parámetros del experimento computacional. Se analicen los factores que tienen el efecto directo al resultado del algoritmo M+m, se elije el diseño del experimento y se realiza su calibración estadística. Se

compara el algoritmo M+m con el algoritmo de referencia y se hacen las conclusiones acerca la exactitud y eficiencia de los algoritmos implementados.

El capítulo 5 contiene las conclusiones obtenidas en la investigación y desarrollo del trabajo.

Al final se incluyen las referencias utilizadas, seguidas de los anexos que complementan.

2 EL PROBLEMA DE TALLER DE FLUJO HÍBRIDO Y MÉTODOS DE RESOLUCIÓN

2.1 Conceptos previos

2.1.1 Notaciones y definiciones

Los términos y las notaciones usados en la presente tesis corresponden a los términos y las notaciones de las fuentes básicas de la Teoría de Planificación (Pinedo, 2002).

De acuerdo a la terminología de la Teoría de Planificación, las actividades se definen como los trabajos y los recursos de cualquier tipo como las máquinas. Se supone que el número de trabajos y máquinas es conocido de antemano.

Un problema de planificación de trabajos en un sistema de recursos (*scheduling problem*) viene determinado por:

a) el número y tiempos de trabajos a procesar;
b) el número y tipo de máquinas disponibles;
c) el patrón de flujo de los trabajos en las máquinas;
d) el criterio (uno o varios) de optimización con el que se evalúa una secuencia de producción.

Un conjunto J de trabajos (*jobs*), $J = \{1,...,n\}$ es necesario procesar sucesivamente en un conjunto m de etapas, $M = \{1,...,m\}$. Cada etapa i, $i = 1,...,m$, tiene m_i máquinas paralelas capaces de ejecutar los trabajos, $|m_i| \geq 1$. Cada trabajo debe pasar por todas las etapas y debe procesarse exactamente en una máquina por etapa.

Todo trabajo j, $j \in J$, tiene su ruta tecnológica en las máquinas, la cual incluye m operaciones (tareas) sucesivas: $\{O_{1j}, O_{2j}, ...O_{ij}, ..., O_{mj}\}$. La ruta de un trabajo j en las máquinas implica una relación de precedencia entre las operaciones. Es decir, el orden de procesamiento de m operaciones para el trabajo j es el siguiente: $O_{1j} \prec O_{2j} \prec ... \prec O_{mj}$, $\forall j$, $j = \overline{1,n}$. La operación O_{ij}, una vez comenzada en la máquina, no se interrumpe hasta terminarse.

Una máquina ejecuta una sola operación. La misma operación se asigna a cualquiera de las máquinas, si estas son convenientes para tal operación. De este modo, las máquinas son unidas en m grupos correspondientemente a la operación que realizan, con m_i máquinas en cada grupo, $i = \overline{1,m}$, y con el total de $\sum_{i=1}^{m} m_i$ máquinas. Formalmente, las m operaciones sucesivas en la realización del trabajo j representan etapas, las cuales se asocian con m grupos de máquinas. En tal caso, O_{ij} denota la operación realizada en la etapa i para el trabajo j.

El tiempo de procesamiento del trabajo j, $j \in J$, en una de las máquinas de la etapa i se denota por p_j^i. En caso de que la operación O_{ij} del trabajo j es asignada a la máquina l de la etapa i, $l \in \{m_i\}$, el tiempo de procesamiento se denota como p_{lj}^i, $i = \overline{1,m}$, $l = \overline{1,m_i}$. Los tiempos de procesamiento de todos trabajos son conocidos de antemano.

En 1979, Graham propuso para la descripción de un problema la tripleta $\alpha \mid \beta \mid \gamma$, donde:

α especifica el ambiente de recursos (o máquinas) y patrón de flujo;

β describe las propiedades de los trabajos;

γ establece el criterio de optimización.

A continuación se presentan las opciones más frecuentes para estos campos.

Opciones para α

(1) **una máquina** (*Single machine*). Este es el caso más simple entre todos los sistemas de recursos y un caso particular para sistemas más complejos.

***m* máquinas:**

(*Jm*) **Taller de Trabajo** (*Jobshop*). En un Taller de Trabajo con *m* máquinas cada trabajo tiene la ruta predeterminada a seguir por las máquinas. Es admisible si un trabajo llega a la misma máquina más de una vez y no llegar ninguna vez a algunas máquinas.

(*Om*) **Taller Abierto** (*Openshop*). En un Taller Abierto con *m* máquinas todo trabajo se procesa exactamente una vez en cada máquina, pero las rutas no son importantes.

***m* máquinas en paralelo:**

(*Pm*) **Máquinas Idénticas en Paralelo** (*Parallel and Identical Machines*). Son *m* máquinas idénticas en paralelo. Trabajo *j* se procesa en cualquiera de *m* máquinas.

(*Qm*) **Máquinas Uniformes** (*Uniform Machines*). Son *m* máquinas paralelas con velocidades diferentes. La velocidad de la máquina *i* se denota por v_i. El tiempo p_{ij} de procesamiento del trabajo *j* en la máquina *i* es p_j / v_i. En caso de máquinas idénticas $v_i = 1,\ \forall i = \overline{1,m}$.

(*Rm*) **Máquinas No Relacionadas** (*Unrelated machines*). Son *m* máquinas en paralelo, cada máquina posee velocidad diferente. La máquina *i* procesa el trabajo *j* con la velocidad v_{ij}. El tiempo p_{ij} de procesamiento del trabajo *j* en la máquina *i* es p_j / v_{ij}.

***m* grupos de máquinas:**

(*Fm*) **Taller de Flujo** (simple) (*Flowshop*, *Flowline*). En este Taller de Flujo, las *m* máquinas son ordenadas linealmente y los trabajos siguen la misma ruta: desde la primera máquina hasta la última.

(*FFm*) **Taller de Flujo Flexible** (*Flexible Flowshop*). Es la generalización de un Taller de Flujo simple. Cada trabajo debe pasar por *m* etapas. Al menos una etapa tiene máquinas paralelas idénticas.

(*FHm*) **Taller de Flujo Híbrido** (*Hybrid Flowshop*). Es la generalización de un Taller de Flujo Flexible (TFF). Las máquinas paralelas en al menos una etapa no son idénticas.

En la Figura 2.1 se muestra una clasificación de los modelos de recursos de acuerdo al tipo de flujo de trabajos por las máquinas.

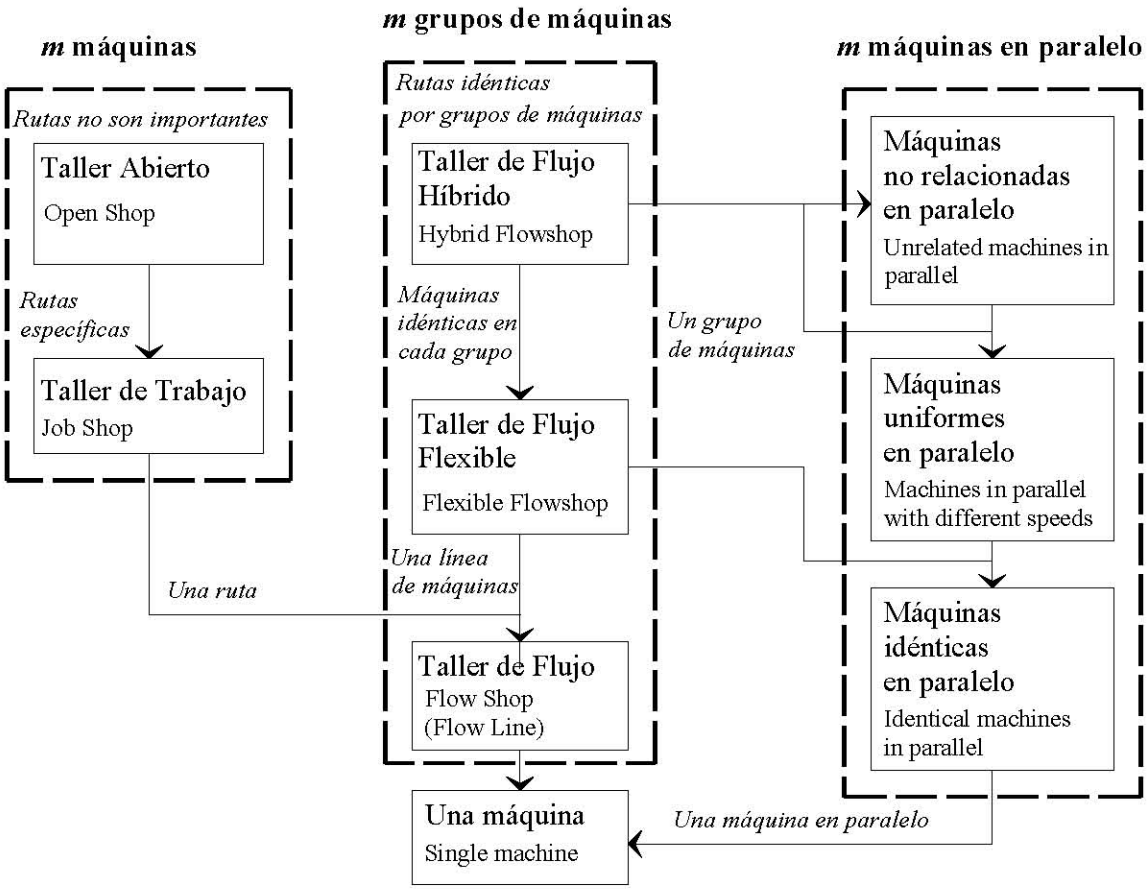

Figura 2.1. Clasificación de los modelos de recursos

Opciones para el β :

$\varnothing$ Sin propiedades de tarea (vacío)

M_j Restricciones en uso de máquinas (*machine eligibility constraints*). Se utiliza cuando el sistema de recursos contiene máquinas paralelos (Pm). La presencia de la opción M_j significa que no todas las m máquinas son capaces de procesar el trabajo j. El conjunto M_j denota el conjunto de máquinas capaces de procesar el trabajo j. Si el campo β no contiene M_j, entonces cualquiera de las m máquinas es capaz procesar el trabajo j.

prmp o *pmtn*	Se permite la interrupción durante la ejecución de operaciones. Una operación O_{ij}, una vez que comienza en una máquina, se interrumpe, si otra tarea diferente con la preferencia (*preemption*) más alta se introduce en la máquina. El tiempo ya procesado no se pierde. El tiempo restante se procesa en la misma o en otra máquina.
prmu	La solución se busca entre las permutaciones. Es más conveniente para FS. Las tareas se procesan en cada máquina de acuerdo a la disciplina *First In First Out* (FIFO), lo que implica que el orden (*permutation*) en el cual los trabajos pasan por la primera máquina, se mantiene en todas las máquinas.
r_j	Para todo trabajo j existen instantes o fechas de entrada r_j (*release dates*), cuando el trabajo este listo para procesarse en las máquinas. Ninguna operación del trabajo se inicia antes de la fecha de entrada. Se supone que $r_j = 0$, $\forall j$. Es decir, todos los trabajos están libres para procesarse en el momento 0.
d_j	Significa un compromiso en las fechas de finalización (*due date*) del trabajo j. Se permite una finalización posterior, aunque con una penalización. Las fechas de finalización son independientes para cada trabajo.
$\overline{d}$	La fecha de finalización obligada (*deadlines*) supone un compromiso firme de finalización para todos los trabajos.
w_j	El peso (*weight*) o prioridad de un trabajo j refleja la importancia del trabajo j con respecto a otros trabajos del conjunto. El peso se establece en función del costo o beneficio del trabajo, tipo de cliente, etc.
$\beta \in \{p_{ij} = 1; p_{ij} \in \{0,1\}; p_{ij} = p_{ij}(t), ...\}$	Las especificaciones del tiempo de ejecución de operaciones: $p_{ij} = 1$: si todos los tiempos de proceso de los

trabajos son iguales a 1; o $p_{ij} = p$: si todos los tiempos de proceso de los trabajos son iguales a un p.

Objetivo (criterio) γ

Sean medidas de ejecución de trabajos individuales:

C_j es la fecha (tiempo) en que finaliza el trabajo j en el taller (*completion time of job j*).

L_j es la holgura del trabajo frente a su fecha de finalización (*lateness*, *deadline*), $L_j = C_j - d_j$:

- $L_j = 0$ el trabajo termina a tiempo (*on-time*),
- $L_j < 0$ el trabajo ha terminado antes de lo previsto (*early*)
- $L_j > 0$ el trabajo termina después de la fecha máxima y se dice que se ha retrasado (*tardy*).

T_j es el retraso (*tardiness*) de un trabajo j, $T_j = \max\{L_j, 0\}$.

E_j es el adelanto (*earliness*) de un trabajo j, como $E_j = \max\{-L_j, 0\}$.

Algunas funciones a minimizar:

$C_{\max} = \max_{1 \le j \le n} C_j$ máxima fecha de finalización (*makespan).* Este criterio de optimización es el más utilizado en la literatura.

$L_{\max} = \max_{1 \le j \le n} L_j$ máxima holgura (*maximum lateness*) o máximo incumplimiento de la fecha de finalización.

$\sum w_j C_j$ total ponderado de tiempos de terminación de tarea.

$\sum w_j (1 - e^{-rC_j})$ total ponderado con descuento de C_j

2.1.2 Modelos estáticos y estocásticos

Los problemas de planificación de trabajos se describen mediante modelos estáticos (determinísticos) o estocásticos (dinámicos, aleatorios).

En modelos estáticos la solución se busca teniendo la información completa, conocida de antemano. A estos pertenecen los procesos de manufactura automatizada, donde las duraciones de operaciones se conocen por experiencia; gestión de la producción por robots; algunos problemas de gestión de transporte, etc.

En modelos estocásticos el proceso de toma de decisión suele tener varias partes, de acuerdo con la llegada (o no llegada) de nueva información. La necesidad en planificación de trabajos para modelos estocásticos, surge en procesos aleatorios, cuando un evento sucede con algún valor de probabilidad, y sus características no se conocen de antemano. Estos modelos son comunes en sistemas de tiempo real, como un *Grid* computacional, redes de información o telecomunicaciones.

Los algoritmos para asignación de órdenes de ejecución de trabajos a las máquinas que utilizan modelos estocásticos de la planificación reciben el nombre de algoritmos *on-line*, a diferencia de algoritmos *off-line* para los modelos estáticos.

Los modelos de la planificación de trabajos, que se emplean a continuación, son estáticos.

2.1.3 Complejidad computacional

La Teoría de la Complejidad se desarrollo para el estudio de la dificultad intrínseca de los algoritmos (Rosen, 2004). La complejidad de un algoritmo, está dada por el máximo número de pasos computacionales necesarios para obtener la solución. Esto a su vez requiere la definición de un paso computacional. Para definir un paso computacional se usa un modelo estándar de cómputo denominado "La maquina de Turing".

El número de pasos computacionales a menudo es simplemente el máximo número de operaciones que el algoritmo necesita para llegar a la solución. Por lo general, este número es aproximado.

Por ejemplo, el análisis cuidadoso de un algoritmo muestra que el máximo número de operaciones necesarias para obtener la solución de un problema en el peor caso se evalúa como la función $T(n)\ \ 5n^3+100n^2+1500$, que representa una función de tercer orden. A pesar de que para pequeños valores de n los dos últimos términos tienen un impacto más grande en el número de operaciones, estos no son de interés para problemas en gran escala.

Incluso el coeficiente del primer término (el 5) no es importante. Para valores grandes de n el primer término tiene el mayor impacto en el número máximo de operaciones requeridas. En tal caso se dice que el algoritmo tiene la complejidad $O(n^3)$. Tal función es una asintótica, cuyo valor siempre se encuentra por encima de la complejidad exacta del algoritmo, a partir de algún valor n_0.

Un algoritmo, cuya función asintótica representa un polinomio del tamaño n del problema es llamado algoritmo de tiempo polinomial. El algoritmo de tiempo polinomial contrasta con un algoritmo exponencial $O(2^n)$ o factorial $O(n!)$, donde el número de operaciones se evalúa con una función exponencial o factorial del tamaño del problema.

En la teoría de complejidad se hace una distinción entre problemas de optimización y problemas de decisión. La cuestión radica en que un problema de decisión requiere solo la respuesta "si" o "no". A cada problema de optimización se le asocia un problema de decisión.

Por ejemplo, en el problema $Fm||C_{\max}$ se busca el mínimo del $C_{\max}$ (*makespan).* Este es el problema de optimización. El problema de decisión que le corresponde es: ¿Existe un orden de trabajos en las máquinas con el valor de *makespan* menor a un z dado?

Se observa claramente que el problema de optimización y el problema de decisión se encuentran fuertemente ligados. Si existe un algoritmo polinomial para solucionar un problema de decisión, entonces también existe un algoritmo polinomial para solucionar un problema de optimización y viceversa.

Un concepto fundamental en Teoría de Complejidad es la "reducción del problema". Se busca un procedimiento que en un tiempo polinomial reduce el problema de decisión de interés a un problema de decisión reconocido.

Se dice que un problema P se reduce a un problema P' si para cualquier instancia de P se construye una instancia P' equivalente, en un tiempo polinomial. La reducción polinomial de P a P' se denota por $P \propto P'$. Si se conoce que no existe un algoritmo de tiempo polinomial para un problema P, entonces tampoco existe un algoritmo de tiempo polinomial para P'.

Se definen clases $\mathcal{P}$ y $\mathcal{NP}$ de algoritmos.

Clase $\mathcal{P}$: la clase $\mathcal{P}$ (*polinomial*) contiene aquellos problemas de decisión para los cuales existe un algoritmo en la máquina de Turing que conduce a la respuesta correcta "si" o "no" en un número de pasos restringidos por un polinomio en la longitud de la codificación. La definición de la clase $\mathcal{P}$ está basada en el tiempo que le toma a una máquina de Turing para resolver un problema de decisión.

Clase $\mathcal{NP}$: la clase $\mathcal{NP}$ (*nondeterministic polinomial*) contiene aquellos problemas de decisión, para los cuales la respuesta correcta se verifica por una máquina de Turing en un número de pasos restringido por un polinomio en la longitud de la cadena de entrada.

Por ejemplo, el problema de decisión asociado al problema de optimización $F3||C_{\max}$ es considerado. Una solución es una permutación de trabajos, a la cual corresponde un cierto valor de *makespan* que además es menor a un valor z dado. Para comprobar que la respuesta correcta es "si", el algoritmo toma la permutación y calcula el *makespan* con esta permutación para mostrar si este es no mayor que la constante z. Se supone que el algoritmo es polinomial. En este caso sólo se afirma que el problema de decisión pertenece a la clase $\mathcal{NP}$.

Según la definición, la clase $\mathcal{P}$ es una subclase de $\mathcal{NP}$.

Una de las cuestiones más importantes en la teoría de complejidad ¿$\mathcal{P} = \mathcal{NP}$? sigue abierta hasta el momento. Si $\mathcal{P}$ fuese igual a $\mathcal{NP}$, entonces existirían algoritmos de tiempo polinomial para todos los problemas de clase $\mathcal{NP}$ (Kelley, 1995).

Clase $\mathcal{NP}$-completo: es una clase de problemas que pertenecen a la clase $\mathcal{NP}$ y a los cuales se reduce cualquier problema de la clase $\mathcal{NP}$. Para los problemas $\mathcal{NP}$-completos hasta ahora no existen algoritmos polinomiales, de otro lado, al encontrar la solución se verifica rápidamente que esta es correcta. De encontrarse una solución en tiempo polinomial para un problema $\mathcal{NP}$-completo, todos los problemas $\mathcal{NP}$ también tendrían una solución en tiempo polinomial.

Clase $\mathcal{NP}$-duro: son los más difíciles entre problemas $\mathcal{NP}$-completos.

No todos los problemas de la clase $\mathcal{NP}$-duro son igualmente difíciles. Algunos problemas son más complejos que otros. Por ejemplo: es posible que un problema $\mathcal{NP}$-duro se soluciona en tiempo polinomial como una función del tamaño de entrada con la codificación unaria, mientras que no se soluciona en tiempo polinomial como una función

de tamaño de entrada con la codificación binaria. Para otros problemas, no existe un algoritmo de tiempo polinomial bajo la codificación unaria o binaria. El problema en la primera clase es usualmente conocido como $\mathcal{NP}$-duro en el sentido ordinario o simplemente $\mathcal{NP}$-duro. Los algoritmos para este tipo de problemas son conocidos como algoritmos pseudopolinomiales, los cuales realmente son exponenciales pero calculables para tamaños de entrada no muy grandes. La segunda clase de problemas es usualmente conocida como $\mathcal{NP}$-duro en sentido estricto.

2.2 Taller de Flujo Híbrido con dos etapas

Un TFH tiene mucho en común con un TFF, por tanto hay muchas confusiones entre términos. Si en un sistema de recursos al menos una máquina en un grupo es distinta de otras por su velocidad u otras capacidades, este hecho refiere tal modelo a los TFH, mientras cualquier máquina del grupo ejecuta cualquier trabajo durante el mismo tiempo, lo que es característico para un TFF. Por tanto, las metodologías para investigación de los TFF en muchos casos son aplicables para los TFH y viceversa.

Se considera un sistema de recursos con dos grupos de máquinas. Supongamos que las máquinas en cada grupo son idénticas, pues poseen capacidades y velocidades iguales. Es necesario procesar el conjunto $J = \{1, 2, ..., n\}$ de trabajos. Cada trabajo j, $j = \overline{1, n}$, consiste en dos operaciones: O_{1j} y O_{2j}. La primera operación O_{1j} se ejecuta por una de las máquinas del primer grupo durante p_j^1 unidades de tiempo. Después de su terminación se realiza la operación O_{2j} en una de las máquinas del segundo grupo durante tiempo p_j^2. Cualquier máquina del primer grupo es capaz de realizar la primera operación del trabajo j. Análogamente, cualquier máquina del segundo grupo es capaz de realizar su segunda operación. Las interrupciones de operaciones no se permiten. Todos los trabajos están listos para su ejecución en el momento 0, lo que significa que los instantes o fechas de entrada son $r_j = 0$, $\forall j = \overline{1, n}$. Es necesario asignar el orden de ejecución de los trabajos a las máquinas de tal manera que el tiempo de ejecución de todos los trabajos sea mínimo.

El modelo de tal proceso corresponde a un TFF con dos etapas, donde las etapas se asocian con los grupos de máquinas. Para el algoritmo heuristico, cuyo comportamiento se investiga en la presente tesis, el problema se descompone en dos subproblemas:

1) Asignación de trabajos a las máquinas.

2) Ordenación de los trabajos en cada máquina.

Resolver el problema de asignación significa indicar la máquina m_{1i} del primer grupo, $i \in \{1,...,m_1\}$, y la máquina m_{2i} del segundo grupo, $i \in \{1,...,m_2\}$ para la ejecución de las operaciones de cada trabajo j. Una vez ejecutado, el procesamiento de asignación transforma el modelo de un TFF en un caso de TFH con elegibilidad de las máquinas. De tal manera, los algoritmos investigados en el sistema computacional, son aplicables tanto para un TFH, como para su caso particular, un TFF, después de un procesamiento de asignación de trabajos a las máquinas.

Este problema de TFH2 se describe a través de la fórmula $FH2,(Rm_i)_{i=1}^{(2)} \mid prmu, M_j \mid C_{\max}$, que indica un sistema de recursos con dos grupos de máquinas no relacionadas en paralelo, con m_1 máquinas en el primer grupo y con m_2 máquinas en el segundo. Se requiere asignar tal orden de los trabajos en las máquinas, el cual proporciona el tiempo mínimo de su ejecución (mínimo de $C_{\max}$). La solución se busca entre las permutaciones de trabajos.

El sistema de recursos en un TFH2 toma una de tres posibles configuraciones:

1) una máquina en la primera etapa y varias máquinas paralelas en la segunda: $FH2,(R1^{(1)}, Rm_2^{(2)}) \mid prmu \mid C_{\max}$, (Figura 2.2); (2.1)

2) varias máquinas paralelas en ambas etapas (caso general):

$FH2,(Rm_i)_{i=1}^{(2)} \mid prmu \mid C_{\max}$ (Figura 2.3); (2.2)

3) varias máquinas paralelas en la primera etapa una máquina en la segunda:

$FH2,(Rm_1^{(1)}, R1^{(2)}) \mid prmu \mid C_{\max}$ (Figura 2.4). (2.3)

En la tesis se investigan dos algoritmos para resolución del problema (2.2), el cual es el más general entre las tres casos mencionados.

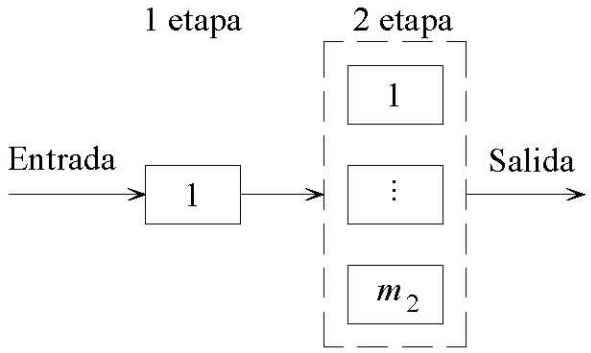

Figura 2.2. TFH2, una máquina en la primera etapa y m_2 máquinas en la segunda

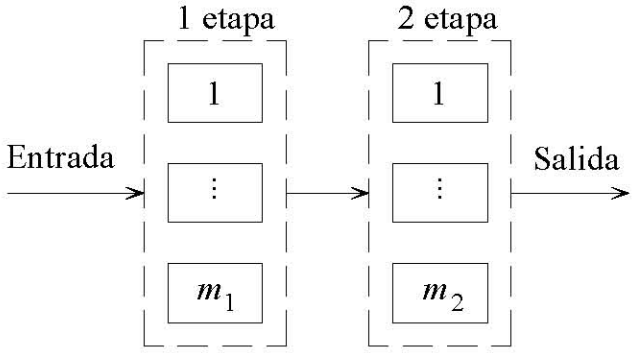

Figura 2.3. TFH2, m_1 máquinas en la primera etapa y m_2 máquinas en la segunda

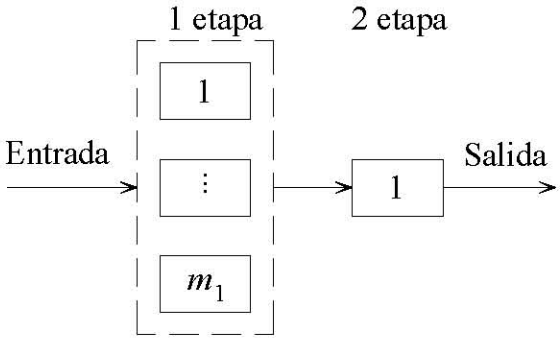

Figura 2.4. TFH2, m_1 máquinas en la primera etapa y 1 máquina en la segunda

El caso particular de un TFH2, cuando $m_1 = m_2 = 1$, es conocido como el problema de Johnson para dos máquinas. La regla de Johnson encuentra la solución óptima para un conjunto determinado de trabajos, los cuales se procesan sucesivamente en dos máquinas

(Johnson, 1954). Numerosas variaciones de la regla se usan para resolución de distintos problemas en la planificación de trabajos. En el apartado 2.5 se describe un algoritmo heurístico para resolver el problema (2.2), basado en la regla de Johnson (Burtseva et. al., 2005).

El problema de minimización de $C_{\max}$ en un TFH2 pertenece a los $\mathcal{NP}$-duros. Gupta en 1988 mostró que es $\mathcal{NP}$-completo si el número de máquinas en una de las etapas es mayor que uno. Para la comprobación se usan las suposiciones que simplifican el modelo: las máquinas en cada etapa son idénticas, además $m_1 \leqq m_2$ y tiempo de procesamiento en la primera máquina es cero para todo trabajo. En tal caso, el problema se reduce hasta la planificación de trabajos en m_2 máquinas idénticas. Es equivalente a encontrar la solución del problema m_2-PARTICIÓN, acerca del cual es conocido que es $\mathcal{NP}$-completo (Garey, 1979). Gupta comprueba que el problema de TFH2 es $\mathcal{NP}$-duro para cualquier valor arbitrario de m_1 y m_2, mientras $\max(m_1, m_2) > 1$. Por lo tanto, sólo algunos casos especiales suelen encontrar las soluciones óptimas con algoritmos polinomiales o pseudopolinomiales. La solución al problema en general se busca con métodos aproximados. Las más exitosas técnicas se presentan en el apartado 2.3.

Existen numerosas técnicas que aportan soluciones satisfactorias para el problema de minimización de *makespan* para un TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda. A continuación se analizan algunas importantes publicaciones (Tabla 2.1).

Una de las primeras publicaciones se debe a Sriskandarajah y Sethi en 1989 que posteriormente abordan el problema con el mismo número de máquinas en las dos etapas. Utilizan una variante del algoritmo de Johnson aplicada a la descomposición del problema en M subproblemas de Taller de Flujo.

Tabla 2.1. Algoritmos revisados para un TFH2

Año	Autores	Problema	Método

1988	Gupta	$FH2,(Rm_1^{(1)},R1^{(2)})\mid prmu \mid C_{\max}$	Algoritmo aproximado
1989	Sriskandarajah y Sethi	$FH2,(Rm)_{i=1}^{(2)} \mid prmu \mid C_{\max}$	Variación de Johnson, descomposiciones en M subproblemas de TF2.
1991	Gupta y Tunc	$FH2,(R1^{(1)},Rm_2^{(2)}),m_2>n \mid prmu \mid C_{\max}$ tamaño pequeño del problemas	Dos heurísticas, ramificación y acotamiento
2002	Riane, Artiba y Elmaghraby	$FH2,(R1^{(1)},R2^{(2)})\mid prmu \mid C_{\max}$	Programación dinámica, variación de Johnson, heurística GRASP
2004	Allaoui y Artiba	$FH2,(R1^{(1)},Rm_2^{(2)})\mid prmu, M_j \mid C_{\max}$	Ramificación y acotamiento
2005	Guirchoun, Martineau y Billaut	$FH2,(R1^{(1)},Q2^{(2)})\mid prmu \mid C_{\max}$	Algoritmo óptimo
2005	Xie y Wang	$FH2,(Rm)_{i=1}^{(2)})\mid no\ \ prmp,M_j \mid C_{\max}$	Heurística

Desde principios de los 90 hasta nuestros días la atención de investigadores se ha centrado en técnicas más avanzadas, ejemplos de estas son el recocido simulado, búsqueda tabú, búsqueda local iterativa, algoritmos genéticos, GRASP (*Greedy Randomized Adaptive Search Procedures*) y otros.

En 1991, Gupta y Tunc extienden el trabajo de Gupta (1991) comprobando obtención de resultados óptimos para un caso particular, en el cual el número de máquinas en la segunda etapa es mayor que el número de trabajos. En caso contrario, se plantean dos heurísticas y un algoritmo de ramificación y acotamiento para abordar problemas de tamaño pequeño.

En 2002, Riane, Artiba y Elmaghraby tratan el problema con una máquina en la primera etapa y dos diferentes máquinas en la segunda para minimizar el tiempo de ejecución de trabajos. Formulan un algoritmo de programación dinámica, adoptan enfoques de Johnson e incluyen una heurística del tipo GRASP.

En 2005, Guirchoun, Martineau y Billaut investigan el problema de TFH en el contexto de un servidor y dos máquinas paralelas. Se asume el tiempo de procesamiento igual a una unidad sin tiempos de espera y el objetivo es minimizar el total de los tiempos para completar todos los trabajos. Se demuestra que el algoritmo propuesto es óptimo en el caso planteado.

Xie y Wang también en 2005, analizan la complejidad y aproximación del TFH2 para el caso no reanudable, cuando se interrumpen las operaciones en las máquinas. Se contemplan restricciones determinísticas de disponibilidad. Proveen algoritmos aproximados con límites finitos de desempeño en el peor caso para algunas variaciones del problema.

Por último, Allaoui y Artiba en 2006 (Allaoui & Artiba, 2006) tratan el problema de TFH2, cuando hay una máquina en la primera etapa y m máquinas en la segunda. Se asume que cada máquina esta sujeta a por lo menos un periodo de no disponibilidad y todos los trabajos no se reanudan si son interrumpidos. Proponen un algoritmo, el cual se basa en el método de ramificación y acotamiento. Se calcula el peor caso entre tres heurísticas: algoritmo LIST, algoritmo LPT y heurística H, propuestos por Lee - Vairaktarakis (1994).

2.3 Métodos de planificación de los trabajos en un Taller de Flujo Híbrido

A continuación se describe una serie de métodos para la resolución de problemas de optimización combinatoria aplicables a la planificación de la producción.

Los algoritmos se clasifican en dos tipos principales: algoritmos constructivos y algoritmos de búsqueda local. A los algoritmos constructivos pertenecen el algoritmo de Johnson y sus variaciones, así como algoritmos voraces, por ejemplo, algoritmo de Dejkstra entre otros.

Conforme a su exactitud, los algoritmos se clasifican en exactos y aproximados. Los algoritmos exactos buscan una solución óptima, la cual es un óptimo global. Ejemplos de algoritmos de este tipo son algoritmo de Johnson, Búsqueda con Retroceso (*backtracking*), Ramificación y Acotamiento (*branch and bound*) (Land, 1960), Programación Dinámica. A pesar de su gran desarrollo, estos no han demostrado un buen rendimiento ante muchos problemas, lo que ha llevado a la búsqueda de algoritmos aproximados que proporcionan resultados de calidad.

Para problemas de optimización difíciles, de gran tamaño, son aplicables métodos basados en búsqueda local. Ejemplos de algoritmos de búsqueda local son recocido simulado y búsqueda tabú.

Los últimos diez años, han tenido un desarrollo considerable en los métodos inspirados en la evolución biológica, como algoritmos genéticos, algoritmos de la colonia de hormigas. A continuación se presenta la breve descripción de los métodos mencionados.

2.3.1 Algoritmo de Johnson

El algoritmo de Johnson, encuentra la solución óptima de problema $F2 \| C_{\max}$ (Handbook, 2004). La obra de Johnson ha sido reconocida como el origen en la investigación de la planificación (*scheduling*). Los trabajos en la secuencia óptima se ubican según la regla de Johnson[1].

La regla de Johnson determina que un trabajo j precede en la secuencia a un trabajo k si:

$$\min\{p_{1j}, p_{2k}\} < \min\{p_{1k}, p_{2j}\}.$$

La solución se encuentra entre las permutaciones. A continuación se presenta el algoritmo clásico de Johnson:

Paso 1. Hacer *Ini* = 1, *Fin* = *n*. Crear una lista l de trabajos ya secuenciados y una lista de trabajos no secuenciados l'. Inicializar $l = \varnothing$ y $l' = \{1,...,n\}$.

[1] En varias fuentes bibliográficas la regla recibe el nombre de "la condición de Bellman-Johnson" (Pinedo, 2002).

Paso 2. Entre todos los trabajos de l', buscar el trabajo con el mínimo tiempo de procesamiento en la máquina 1 y el trabajo con el mínimo tiempo de procesamiento en la máquina 2. Hacer $\min_1 = \min_{j \in l'}(p_{1j})$, $h_1 = l'(j)$ y $\min_2 = \min_{k \in l'}(p_{2k})$, $h_2 = l'(k)$.

Paso 3. Si $\min_1 < \min_2$, entonces ir al Paso 4. Si $\min_1 > \min_2$, entonces ir al Paso 5. Si $\min_1 = \min_2$, ir de manera aleatoria al Paso 4 o 5.

Paso 4. Insertar el trabajo h_1 en la posición Ini de l, $l(Ini) = h_1$. Hacer $Ini = Ini + 1$. Ir al Paso 6.

Paso 5. Insertar el trabajo h_2 en la posición Fin de l, $l(Fin) = h_2$. Hacer $Fin = Fin - 1$. Ir al Paso 7.

Paso 6. Eliminar h_1 de l'. Si $l' = \varnothing$, entonces fin. En el caso contrario ir al Paso 2.

Paso 7. Eliminar h_2 de l'. Si $l' = \varnothing$, entonces fin. En el caso contrario ir al Paso 2.

Ejemplo

Se busca mínimo valor de $C_{\max}$ para los siguientes datos iniciales:

Trabajo j:	1	2	3	4	5	6	7	8
p_{1j}:	5	2	1	7	6	3	7	5
p_{2j}:	2	6	2	5	6	7	2	1

La solución óptima:

Trabajo j:	3	2	6	5	4	7	1	8
p_{1j}:	1	2	3	6	7	7	5	5
p_{2j}:	2	6	7	6	5	2	2	1
Fin en M1	1	3	6	12	19	26	31	36
Fin en M2	3	9	16	22	27	29	33	37

$C_{\max} = 37$.

La regla de Johnson no es aplicable para el número de máquinas $m > 2$ de forma directa. Sin embargo, existen casos particulares con 3 o más máquinas donde esta se usa. Se

han desarrollado muchas modificaciones del algoritmo para distintos modelos de talleres. En el apartado 2.5.2 se presenta una modificación del algoritmo para solución de un problema de TFH2.

Como el problema $F2||C_{\max}$ es resoluble de modo eficiente, uno de los caminos para 3 y más máquinas sucesivas, es reducir el caso hacia $F2||C_{\max}$ y buscar la solución entre las permutaciones. En su obra original, Johnson (1954) indica un modelo, donde se encuentra la solución óptima de $F3||C_{\max}$. La condición es: los tiempos a procesar en la segunda máquina son uniformemente más cortos con respecto a la primera, y de igual forma, en la tercera máquina con respecto a la segunda:

a) Si $\min_j \{t_{1j}\} \geq \max_j \{t_{2j}\}$, o

b) Si $\min_j \{t_{3j}\} \geq \max_j \{t_{2j}\}$,

Tal problema se resuelve con la regla de Johnson mediante la resolución del problema auxiliar de dos máquinas con tiempos

$$P_{1j} = p_{1j} + p_{2j}; \qquad P_{2i} = p_{2i} + p_{3i}.$$

Autores, como Cambell, Dudek y Smith (1970), aplican la regla de Johnson para m máquinas sucesivas creando un problema con dos máquinas "virtuales". Una manera simple de construir tal problema virtual es definido como p'_{1j} y p'_{2j} a los tiempos en las dos máquinas del problema virtual que se calculan como:

$$p'_{1j} = \sum_{i=1}^{\left\lceil \frac{m}{2} \right\rceil} p_{ij}, \quad p'_{2j} = \sum_{i=\left\lceil \frac{m}{2} \right\rceil + 1}^{m} p_{ij}, \quad j = (1,...,n).$$

Resolviendo este nuevo problema con la regla de Johnson se obtiene la secuencia de los trabajos para el problema original, aunque en este caso la solución recibida se desvíe considerablemente de la solución óptima.

2.3.2 Algoritmos voraces

Un algoritmo voraz (*greedy*) optimiza la selección hecha en cada paso sin considerar las elecciones que corresponden a pasos anteriores (Brassard & Bratley, 1996). Son los algoritmos constructivos, los cuales generan resultados desde cero agregando componentes a la solución en cada paso de ejecución. Estos algoritmos obtienen una solución razonablemente buena en un tiempo corto, sin embargo no garantizan la optimalidad de la solución.

Los algoritmos voraces se aplican a problemas de optimización en los cuales se cumple el llamado principio de optimalidad: "En una secuencia óptima de decisiones, cualquier subsecuencia debe ser también óptima para el subproblema que resuelve". Para algunos de estos problemas, es sencillo encontrar una secuencia de decisiones óptima sin cometer errores: serán los problemas que hemos resuelto mediante algoritmos voraces. A estos pertenece el algoritmo de Dejkstra para solución del problema del vendedor viajero, uno de los problemas más generales de optimización (Rosen, 2004).

Una estrategia común es el uso de un algoritmo voraz como punto de partida para la búsqueda local.

2.3.3 Método general de la búsqueda local

Búsqueda local es una metaheurística que resuelve problemas de optimización difíciles. Se formula como la obtención del máximo de un criterio entre varios candidatos.

El algoritmo de búsqueda local se mueve de solución a solución en el espacio de sus candidatos hasta que se encuentra el óptimo o hasta un tiempo límite determinado.

Algunos problemas de optimización donde se aplica la búsqueda local son:

- El problema de la cobertura de vértices, cuya solución es la cobertura de vértices en un grafo. El objetivo es encontrar una solución con el número mínimo de vértices.
- El problema de vendedor viajero, cuya solución es un ciclo cerrado que contiene todos los nodos del grafo. El objetivo es minimizar la longitud total del ciclo.
- El problema de satisfactibilidad booleana, cuya solución es una asignación de valores de verdad a las variables. El objetivo es maximizar el número de cláusulas que satisfacen a la asignación.

La mayoría de los problemas se formulan en términos del espacio de búsqueda y el criterio. Por ejemplo, para el problema del vendedor viajero, la solución es un ciclo de vértices. El criterio a optimizar es la longitud del ciclo, al cual corresponde una combinación de aristas.

Un algoritmo de búsqueda local comienza con una solución candidata y luego se mueve iterativamente a una solución vecina. La relación de vecindad debe ser definida en el espacio de búsqueda. Por ejemplo, la vecindad de una cobertura de vértices es otra cobertura de vértices que sólo se diferencia por un nodo.

En el problema de satisfactibilidad booleana, los vecinos de una asignación de verdad son por lo general los que se diferencian por la asignación de una sola variable.

Típicamente, cada solución candidata tiene más de una solución vecina. La elección de hacia donde debe moverse es tomada usando sólo la información sobre las soluciones en la vecindad de la solución actual, de ahí el nombre de búsqueda local.

Cuando la elección de la solución vecina es hecha tomando aquella que optimiza el criterio, la metaheurística actúa de acuerdo al proceso de escalada (*climbing*). La búsqueda local se termina cuando llega a un límite de tiempo o cuando la mejor solución encontrada por el algoritmo no se ha mejorado en un número dado de pasos.

Los algoritmos de búsqueda local no necesariamente encuentran la solución óptima. Esto ocurre en el caso de terminación debido a la imposibilidad de mejorar la solución, cuando la solución óptima esta lejos de la vecindad de las soluciones cruzadas por los algoritmos, o cuando se ha terminado el tiempo límite de búsqueda.

Los algoritmos de búsqueda local son aplicados a numerosos problemas computacionales difíciles, a los cuales pertenecen los problemas de planificación de trabajos. Los ejemplos del algoritmo de búsqueda local aplicables en esta área son WalkSAT (Bart, et al., 1996) y "2–opt" (Croes, 1958).

Búsqueda Local:

Inicio

s = solución inicial

While *s* no es óptimo local

IF $s' \in N(s),\ f(s') < f(s)$ **THEN**

se encontró una mejor solución *s'* en la vecindad de *s*,

$s \leftarrow s'$

Return *s*

2.3.4 Recocido simulado

El nombre del método "recocido simulado" proviene de configuraciones de baja energía de materiales magnéticos desordenados, clasificados bajo el termino de "vidrio de espín". La interpretación numérica de estas configuraciones se utiliza en problemas de optimización, en el sentido de que "el vidrio de espín" de un vaso de giro presenta varios "valles" de profundidades desiguales. Kirkpatrick (1983) e independientemente Cerny (1985) propusieron tomar la técnica experimental de recocido usado por los metalurgistas como un punto de partida para obtener un "buen" estado sólido "ordenado", de energía mínima (evitando las estructuras "meta estables", la característica del mínimo local de energía).

El método de recocido simulado invierte el proceso de recocido para la solución de un problema de optimización: La función objetivo del problema, similar a la energía de un material (E), entonces se minimiza a través de la introducción de una "temperatura ficticia" (T), en este caso, un parámetro controlable del algoritmo.

El recocido explota el algoritmo de Nicholas Metrópolis (1953), el cual permite describir el comportamiento de un sistema termodinámico en "equilibrio" a cierta temperatura. Comenzando con una configuración, el sistema se sujeta a una modificación elemental por ejemplo, uno reacomoda un componente, o uno intercambia dos

componentes. Si esta transformación proporciona una disminución en la función objetivo ("la energía") del sistema, entonces es aceptada. Por otra parte, si la transformación causa un incremento en energía (δE) de la función objetivo, se acepta, pero con una probabilidad $e^{-\frac{\delta E}{T}}$. Este proceso se repite de forma iterativa, manteniendo la temperatura constante, hasta que el balance termodinámico sea alcanzado. Luego la temperatura es aminorada, antes de implementar una serie nueva de transformaciones: La ley de disminución por las etapas de la temperatura es a menudo empírica, algo así como el criterio de terminación de programa.

Cuándo es aplicado al problema de la colocación de componentes, el recocido simulado dirige una transformación de desorden a orden, lo cual es representado de una manera gráfica en la Figura 2.5.

Las desventajas del recocido simulado recaen en el "ajuste", tal como la gerencia de la disminución de la temperatura. El usuario debe tener el conocimiento acerca de "buenos" ajustes para evitar que el tiempo de cómputo sea muy significativo. Esto provocó la paralelización del método. Por otra parte, el método de recocido simulado tiene la ventaja de ser flexible con relación a los cambios del problema y es fácil de implementar. Este ha dado resultados excelentes para problemas de gran tamaño.

El diagrama de flujo del algoritmo recocido simulado es presentado en la Figura 2.6.

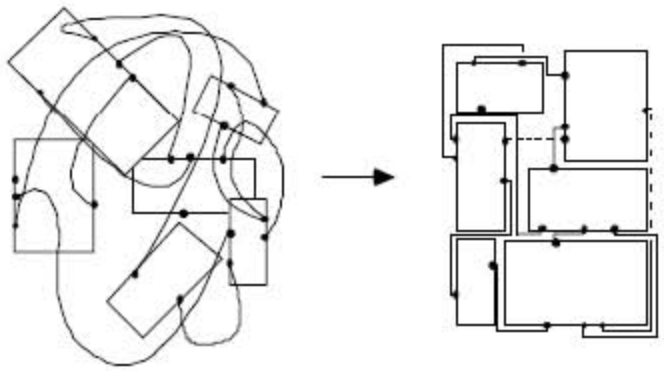

Figura 2.5. Transformación de desorden (a) a orden (b)

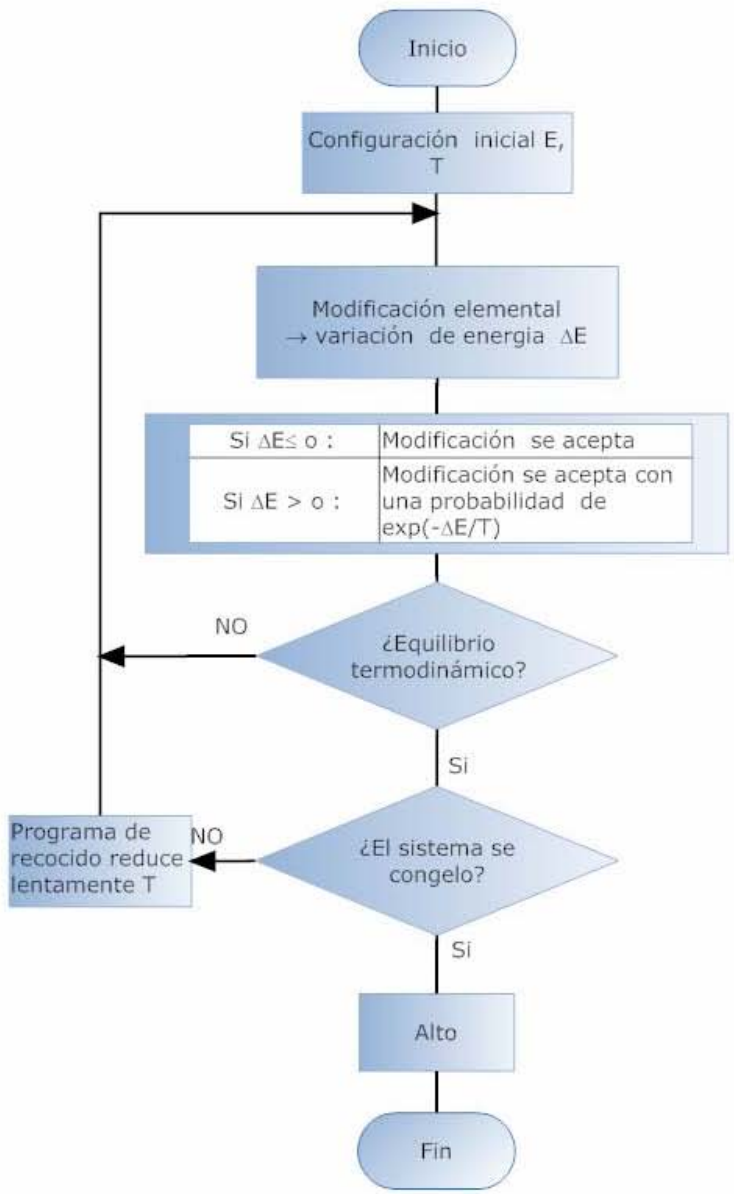

Figura 2.6. Diagrama de flujo del algoritmo recocido simulado

2.3.5 Búsqueda tabú

El método de búsqueda con tabú, o simplemente la búsqueda tabú, o método tabú fue formalizado en 1986 por Glover (1986). Su característica principal se basa en el uso de mecanismos inspirados por la memoria humana. El método tabú toma, de este punto de vista, un camino opuesto al recocido simulado, el cual no utiliza memoria en absoluto, y así es incompetente para aprender la experiencia del pasado. Por otra parte, el modelado de la

memoria introduce grados múltiples de libertad en un riguroso análisis matemático del método (Glover & Laguna, 1997).

El principio del método tabú es simple: Al igual que el recocido simulado, el método tabú al mismo tiempo funciona con sólo una "configuración actual" (al comienzo, cualquier solución), la cual se transforma durante iteraciones sucesivas. En cada iteración, el mecanismo de cambio de una configuración, llamada *s*, para la siguiente, llamada *t*, contiene dos etapas:

1. Se construye el conjunto de los vecinos en *s*. Esto es el conjunto $V(s)$ de las configuraciones accesibles en sólo un movimiento elemental en *s*. Si este conjunto es demasiado grande, se aplica una técnica de reducción del tamaño, por ejemplo, se utiliza una lista de candidatos, o se extrae al azar un subconjunto de vecinos de tamaño fijo.

2. Se evalúa la función objetivo f del problema para cada configuración en $V(s)$. La configuración t en $V(s)$ es aquella, donde f toma el valor mínimo. La configuración t es adoptada aun si es peor que *s*, es decir, $f(t) > f(s)$. Debido a esta característica el método tabú evita que f caiga en un mínimo local.

La desventaja del método es el riesgo significativo para regresar a una configuración ya retenida en el tiempo de una iteración anterior, lo cual genera un ciclo. Para evitar este fenómeno, en cada iteración se requiere la actualización y explotación excesiva de la lista de movimientos tabú, la "lista tabú". Esta lista, contiene m movimientos $(t \to s)$, los cuales son contrarios al los últimos m movimientos $(s \to t)$ llevados a cabo. El diagrama de flujo de este algoritmo conocido como "tabú simple" se muestra en la Figura 2.7.

El algoritmo modela una forma elemental de memoria, la memoria breve de las soluciones visitadas recientemente. Dos mecanismos adicionales, la denominada intensificación y la diversificación, son implementados para equipar el algoritmo con una larga memoria de término. La intensificación es destinada para la exploración de ciertas áreas fuera del espacio de solución. La diversificación es al contrario la reorientación periódica de la búsqueda de un óptimo hacia áreas rara vez visitadas hasta el momento.

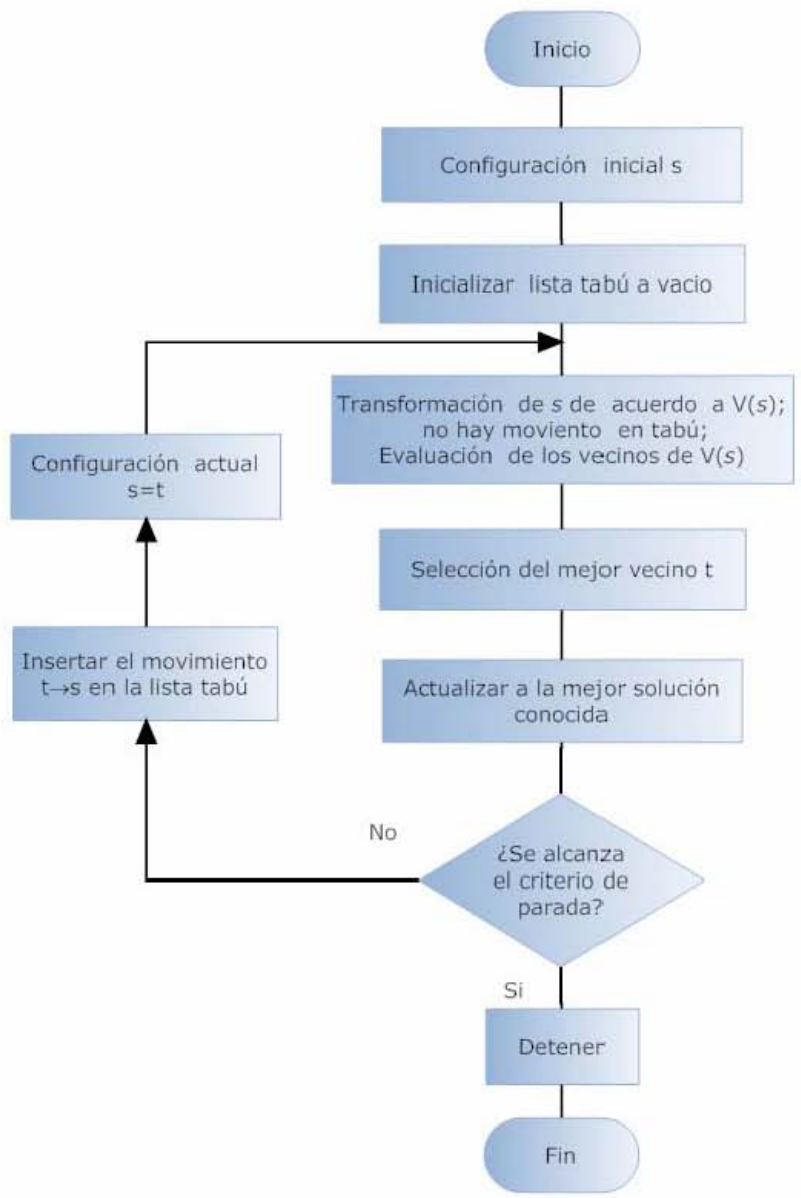

Figura 2.7. Diagrama de flujo del algoritmo tabú simple

Para ciertos problemas de optimización combinatoria, el método tabú proporciona resultados excelentes (Chelouah & Siarry, 2000b), (Siarry & Berthiau, 1997). Además, en su forma básica, el método cuenta con menos parámetros de ajuste que el recocido simulado. Sin embargo, los diversos mecanismos adicionales, como la intensificación y la diversificación, agregan complejidad al método.

2.3.6 Algoritmos evolutivos

Los algoritmos evolutivos son técnicas de búsqueda, inspiradas en la evolución biológica de las especies; aparecieron a fines de los 1950's (Fraser, 1957). Entre varias investigaciones, como (Holland, 1962), (Fogel et al., 1966), (Rechenberg, 1965), (Renders & Flasse, 1996), los algoritmos genéticos son ciertamente el ejemplo más conocido, después de la publicación del famoso libro "*Genetic Algorithms in Search, Optimization and Machine learning*" del autor D.E. Goldberg, en 1989. Los métodos evolutivos inicialmente despertaron un interés limitado, por su costo significativo de ejecución. Sin embargo, han tenido un desarrollo considerable (Chelouah & Siarry, 2000a).

En principio, un algoritmo evolutivo se describe en forma simple. Un conjunto N apunta a un espacio aleatorio de búsqueda, el cual constituye la población inicial. Cada individuo x de la población tiene un cierto valor de adaptabilidad. Un algoritmo evolutivo consiste en evolución sucesiva, por generaciones, manteniendo el tamaño constante de población. El objetivo es mejorar la adaptabilidad de los individuos. Tal resultado se obtiene simulando los dos mecanismos principales que controlan la evolución de los seres vivos, según la teoría de Charles Darwin:

- La selección, que soporta la reproducción y sobrevivencia de los individuos más adecuados.
- La reproducción, que permite la recombinación y las variaciones de las características heredadas de padres a hijos, para formar descendencia con potencialidades nuevas.

En la práctica, para los individuos de una población se selecciona una representación. Así para problemas combinatorios un individuo representa una lista de enteros, para problemas numéricos en espacios continuos es un vector de números reales, para problemas boléanos es una cadena de dígitos binarios. Estas representaciones se combinan en estructuras complejas.

La transformación de una generación a la siguiente pasa por cuatro fases: selección, reproducción (o la variación), evaluación de adaptabilidad y reemplazo.

En la fase de selección se escogen los individuos que toman parte de la reproducción. Cuando los individuos cuentan con una gran adaptabilidad, su probabilidad de ser seleccionados es mayor.

Los individuos seleccionados se hacen disponibles para la reproducción. Ésta consiste en la aplicación de operadores de variación, como cruzamiento y mutación, a copias de los individuos seleccionados anteriormente para generar nuevos individuos. El cruzamiento produce uno o dos descendientes de dos padres, y la mutación produce a un nuevo individuo de sólo un individuo. En la Figura 2.8 se muestra un ejemplo de los operadores de cruzamiento y de mutación, en el caso de individuos representados por cadenas de 8-bits un ejemplo.

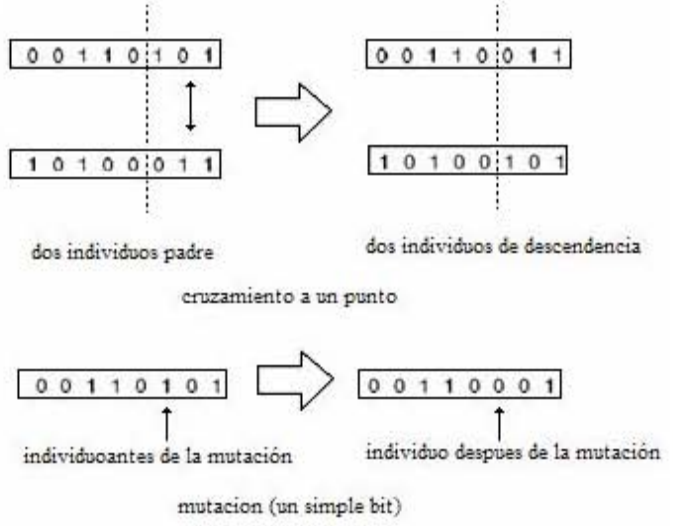

Figura 2.8. Ejemplo de los operadores de cruzamiento y de mutación

La adaptabilidad de los nuevos individuos se calcula en la fase de evaluación.

Finalmente, la fase del reemplazo consiste en seleccionar a los miembros de las generaciones nuevas. Por ejemplo, se reemplazan los individuos con la menor adaptabilidad.

El algoritmo se termina después de un cierto número de generaciones, de acuerdo con un criterio de terminación especificado por el usuario. El esquema de un algoritmo evolutivo se representa en la Figura 2.9.

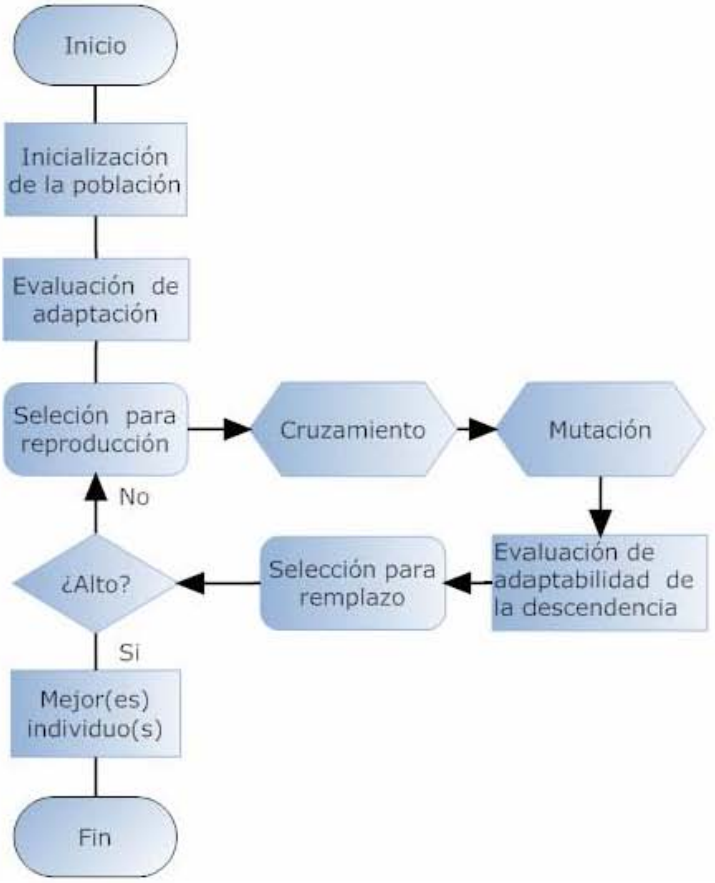

Figura 2.9. Esquema de un algoritmo evolutivo

2.3.7 Algoritmos de colonia de hormigas

El método de colonia de hormigas (*Ant's colony*), propuesto por A. Colorni, M. Dorigo y V. Maniezzo (1992), simula la capacidad colectiva de los miembros de la colonia de hormigas, los cuales poseen facultades limitadas, para solucionar ciertos problemas. Las hormigas en busca de comida fuera del hormiguero siempre siguen el mismo camino, y este camino es el más corto posible como el resultado de un tipo de comunicación indirecta, llamada "*stigmergy*". Cada hormiga deposita una sustancia química (feromonas) a lo largo de su camino. Todos los miembros de la colonia perciben esta sustancia y preferentemente dirigen su paseo hacia las áreas más "olorosas". Si el camino se bloquea fortuitamente por un obstáculo, la capacidad colectiva de las hormigas encuentra rápidamente el camino más corto (Figura 2.10).

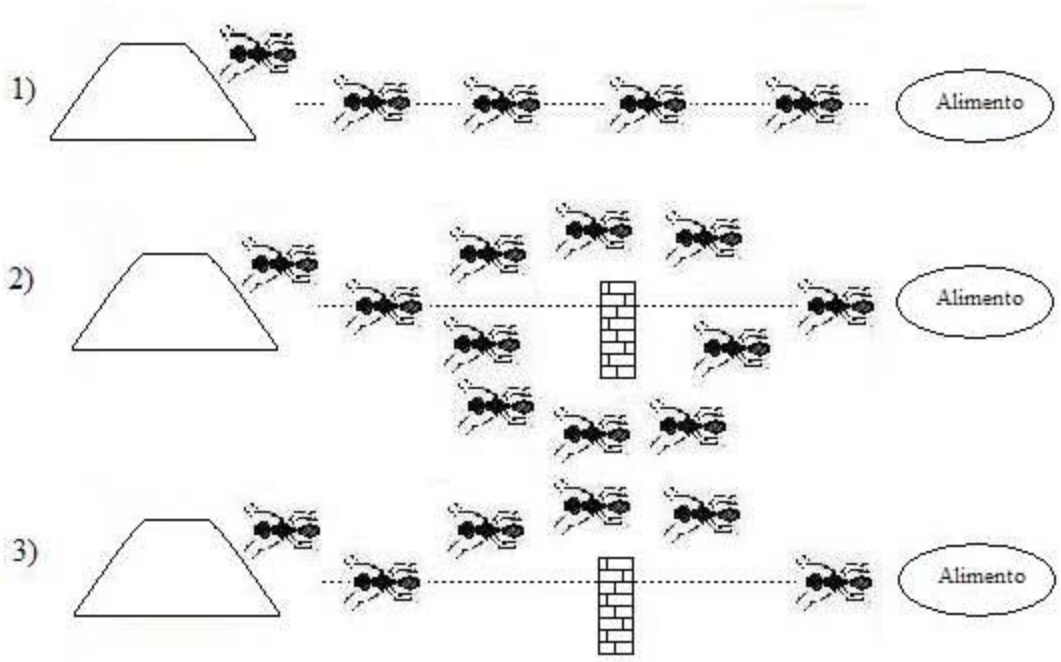

Figura 2.10. Capacidad de una colonia de hormigas para encontrar el camino más corto
Las hormigas siguen un camino entre el hormiguero y una fuente de alimento.
1) Superación de un obstáculo en el camino; las hormigas eligen girar a la derecha o la izquierda, con probabilidades iguales; la feromona es ingresada más rápidamente en el camino más corto.
2) Todas las hormigas escogen el camino más corto.

A. Dorigo (1997) propuso un algoritmo para la solución del problema del vendedor viajero basado en este método. Desde este trabajo de investigación, el método se ha extendido a muchos problemas de optimización combinatoria.

El método de la colonia de hormigas tiene varias características interesantes, como:

- El paralelismo intrínseco;
- La flexibilidad (una colonia de hormigas se adapta a las modificaciones del ambiente);
- La robustez (una colonia está lista para mantener su actividad aun si algunos individuos fallan);
- La descentralización (una colonia no obedece una autoridad centralizada);
- La autoorganización (una colonia encuentra por sí mismo una solución, la cual no se conoce de antemano).

2.4 Aplicación del método de dicotomía para la optimización combinatoria

2.4.1 Método

Dicotomía es un método numérico, el cual por lo regular se aplica en matemáticas para la búsqueda de raíces de ecuaciones algebraicas. Representa la división sucesiva de un intervalo inicial en dos partes con el propósito de localizar la raíz en un intervalo de longitud mínima posible. La idea de división en dos partes se aplica exitosamente como un esquema externo para formular criterios en algoritmos de optimización combinatoria, en particular, en la planificación de la producción (Yaurima et al., 2006).

El método se basa en el Teorema de los Valores Intermedios (Burden et al., 1985), el cual establece que toda función continua f en un intervalo cerrado $[a,b]$ toma los valores entre $f(a)$ y $f(b)$. Esto es, que todo valor entre $f(a)$ y $f(b)$ es la imagen de al menos un valor en el intervalo $[a,b]$. En caso de que $f(a)$ y $f(b)$ tengan signos opuestos, pues $f(a)\cdot f(b)<0$, el valor cero sería un valor intermedio entre $f(a)$ y $f(b)$, por lo que con certeza existe un m^*, tal que $m\in[a,b]$ en $[a,b]$ que cumple con $f(m^*)=0$. De esta forma, se asegura la existencia de al menos una solución de la ecuación $f(x)=0$.

El método de dicotomía es iterativo y consiste en lo siguiente:

Se requiere la continuidad de la función $f(x)$ en el intervalo $[a,b]$. En el paso i se verifica la condición $f(a_i)\cdot f(b_i)<0$. Se calcula el punto medio m_i del intervalo $[a_i,b_i]$. A continuación se calcula $f(m_i)$. Si $f(m_i)=0$, la raíz buscada es encontrada. En caso contrario, se busca el extremo, sea $f(a_i)$ o $f(b_i)$, con el cual $f(m_i)$ tiene signo opuesto. Se redefine el intervalo $[a_i,b_i]$ como $[a_i,m_i]$ o $[m_i,b_i]$ según se haya determinado, en cuál de estos intervalos ocurre un cambio de signo. Con este se continúa sucesivamente encerrando la solución en un intervalo cada vez dos veces más pequeño, hasta alcanzar la precisión deseada. La Figura 2.11 ilustra el procedimiento descrito.

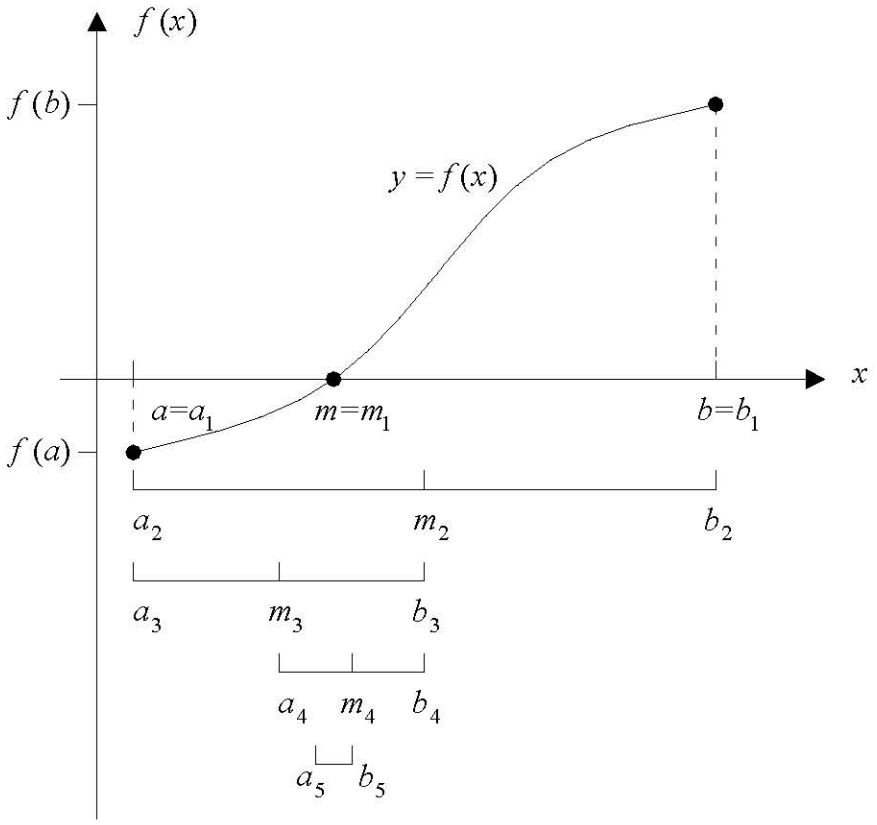

Figura 2.11. Esquema de funcionamiento del método de dicotomía

2.4.2 Convergencia del método de dicotomía

El procedimiento de la dicotomía se aplica en el método de búsqueda binaria, el cual representa un método común en la programación (Rosen, 2004).

La búsqueda binaria consiste en lo siguiente: Dado un valor x. Es necesario determinar ¿se encuentra entre los elementos de un conjunto ordenado $\{a_1, a_2, ..., a_n\}$ un elemento igual a x? El arreglo de elementos en orden creciente (no-decrecimiento en caso de la existencia de elementos con valores iguales) significa una estrategia de estrechamiento del área de ubicación del elemento buscado.

El método de dicotomía representa una generalización de la búsqueda binaria. A continuación se analiza la complejidad del método de dicotomía.

Sea dada una funcional $C(x)$ [2], cuyo dominio es un conjunto finito P de elementos x. Es necesario encontrar tal elemento x^* que

$$C(x^*) = \min_{x \in P} C(x).$$

Supongamos que los límites para $C(x)$ son conocidos de antemano:

$$C^0_{\min} < C(x^*) \le C^0_{\max}.$$

Ahora para la búsqueda del elemento x^* es necesario resolver un C_1-problema.

C_1-**problema**: entre todo elemento del conjunto P encontrar un elemento $x_1 \in P$, tal que $C(x_1) \le C_1$, donde $C_1 = \dfrac{C^0_{\min} + C^0_{\max}}{2}$.

Si tal elemento existe, entonces ponemos:

$$C^1_{\min} = C^0_{\min},$$

$$C^1_{\max} = C_1.$$

En el caso contrario:

$$C^1_{\min} = C_1,$$

$$C^1_{\max} = C^0_{\max}.$$

Ahora $C^1_{\min} < C(x^*) \le C^1_{\max}$.

La iteración j de la búsqueda de x^*, incluye:

1) El calculo $C_j = \dfrac{C^{j-1}_{\min} + C^{j-1}_{\max}}{2}$;
2) La determinación del elemento x_j, tal que $C(x_j) \le C_j$ (en el caso si el C_j-problema es resoluble).
3) La determinación de los límites $C^j_{\min}$ y $C^j_{\max}$ que cumplen las condiciones:

---

[2] Una **funcional** es una función que toma funciones como su argumento; es decir, una función cuyo dominio es un conjunto de funciones.

$$C^j_{\min} = \begin{cases} C^{j-1}_{\min}, & si \quad C_j - \text{problema} \quad \text{tiene} \quad \text{una} \quad \text{solución} \\ C_j, & si \quad C_j - \text{problema} \quad \text{no} \quad \text{tiene} \quad \text{una} \quad \text{solución} \end{cases},$$

$$C^j_{\max} = \begin{cases} C_j, & si \quad C_j - \text{problema} \quad \text{tiene} \quad \text{una} \quad \text{solución} \\ C^{j-1}_{\max}, & si \quad C_j - \text{problema} \quad \text{no} \quad \text{tiene} \quad \text{una} \quad \text{solución} \end{cases}.$$

El intervalo que incluye el elemento buscado se estrecha en cada iteración dos veces. Por otra parte, el número de iteraciones es finito, puesto que el conjunto de valores de la funcional es finito.

Si la funcional toma valores enteros, el proceso termina cuando

$\lfloor C^j_{\max} \rfloor = \lfloor C^j_{\min} \rfloor + 1$.

Número de iteraciones, necesarios para encontrar el mínimo de $C(x)$ es no mayor que $-\lfloor -\log \rho \rfloor$, $\rho = C^0_{\max} - C^0_{\min}$.

2.4.3 Análisis de complejidad

La complejidad temporal del método de dicotomía, se basa en los Teoremas 2.1 y 2.2 para la búsqueda binaria (Aho et al, 1988).

Teorema 2.1. Búsqueda del elemento a_m, el cual es igual a x, en un conjunto de números $a_1 < a_2 < ... < a_n$ requiere $O(\log n)$ operaciones elementales.

La demostración del teorema consiste en la imposibilidad de construir un algoritmo con la complejidad menor que $O(\log n)$.

Teorema 2.2. No existe un algoritmo, el cual requería en número de operaciones menor que el algoritmo de la búsqueda binaria.

La complejidad temporal de algoritmos, basados en el método de dicotomía, depende de la longitud del intervalo $(C^0_{\min}, C^0_{\max}]$, al cual pertenece el valor $C(x^*)$, y de la complejidad temporal del C_j-problema. Por tanto, para minimizar el número de operaciones calculatorias, es muy importante encontrar los más exactos límites superior e inferior de la funcional a minimizar.

Por lo regular, la determinación de $C^0_{\max}$ no es un problema grave: por un límite superior se toma $C(x^0)$ de una solución admisible x^0. Para la mayoría de problemas clásicos en la Teoría de Planificación, cuyos parámetros tienen valores enteros no negativos de Z_0^+, la determinación de un límite inferior $C^0_{\min}$ tampoco requiere esfuerzos grandes.

Las dificultades principales relacionadas con el uso del método de dicotomía provienen de la complejidad del C_j-problema, el cual debe ser formulado como un problema de decisión: Para un valor fijo C ¿existe tal solución x, que $C(x) \le C$?

Si el problema de decisión se resuelve en un tiempo polinomial, entonces el problema de optimización correspondiente también se resuelve en un tiempo polinomial.

Como un ejemplo se formula el problema de empaquetamiento conocido en la optimización combinatoria. Este problema y sus casos particulares tienen muchas aplicaciones en la práctica como, por ejemplo, la asignación de los trabajos a las máquinas paralelas idénticas.

Problema de empaquetamiento:

Para un número entero positivo $C \in Z^+$ ¿existe tal partición del conjunto G de números $p_j \in Z^+$, $j = \overline{1,n}$, en m disjuntos J_i, $i = \overline{1,m}$, tales que

$$\max_{1 \le i \le m} \sum_{p_j \in J_i} p_j \le C \ ?$$

Supongamos que existe un algoritmo polinomial para la resolución del C-problema, el cual requiere el tiempo $t(s)$ para el tamaño de entrada s. Se conoce que para $m = 0$ y valores dados de n, p_j, $j = \overline{1,n}$, no existe una solución $C(x^*)$ (Rosen, 2004). Cuando $m \ge 1$, el valor mínimo $C(x^*)$ y el límite C satisfacen las desigualdades:

$$C(x^*) \le C \le \sum_{j=1}^{n} p_j ,$$

de lo cual sigue que

$$C \le 2^{\log \sum_{j=1}^{n} p_j} .$$

Todos los números p_j en la cadena de entrada del problema de optimización se representan en forma binaria. Por lo tanto,

$$\sum_{j=1}^{n} \log p_j \le s,$$

ó bien,

$$2^{\log \prod_{j=1}^{n} p_j} \le 2^s.$$

Entonces,

$$C \le 2^{\log \sum_{j=1}^{n} p_j} \le 2^{\log \prod_{j=1}^{n} p_j} \le 2^s.$$

Por tanto, para $\rho = C - C(x^*)$, se cumplen las desigualdades:

$$-\lfloor -\log \rho \rfloor \le \log C \le s,$$

y un algoritmo para la resolución del problema de optimización tiene la complejidad temporal $O(t'(s))$, donde $t'(s) = st(s)$. Como $t(s)$ es un polinomio, entonces $st(s)$ también lo es.

Con esta razón se recibe la conclusión de que la complejidad temporal de un problema de optimización combinatoria en general se determina por el tiempo de la obtención de la respuesta "si" o "no" del C-problema. Independientemente de la complejidad temporal del algoritmo para la resolución del C-problema, sus pasos son correctos, puesto que estos

a) construyen la solución x, la cual cumple la desigualdad $C(x) \le C$, si x existe;

b) no encuentran x si x no existe.

A continuación se muestra una adaptación del método de dicotomía para la resolución de problemas en optimización combinatoria, específicamente para la planificación de los trabajos.

2.5 Algoritmo heurístico para minimización de la fecha máxima en un Taller de Flujo Híbrido con dos etapas

2.5.1 Modelo

Se considera un sistema de recursos de dos etapas. Las máquinas de cada etapa son identicas entre sí. El problema de la minimización de *makespan* en tal sistema de recursos corresponde a un TFF, cuya fórmula es $FF(Pm_i)_{i=1}^{2} \mid prmu \mid C_{\max}$. Capacidades y velocidades de las máquinas en una etapa son iguales.

La asignación de los trabajos a las máquinas convierte un TFF a un TFH con elegibilidad de las máquinas.

En un TFH2, cuyas etapas contienen m_1 y m_2 máquinas paralelas respectivamente, se procesa un conjunto determinado de trabajos $J = \{j \mid 1 \le j \le n\}$. Para el trabajo j está definida la ruta tecnológica $i = \{i_1, i_2\}$, como una sucesión ordenada de dos operaciones, $i_1 \in \{m_1\}$, $i_2 \in \{m_2\}$. El tiempo de ejecución de operaciones del trabajo j es $p_{ij} = \{p_{ij}^1, p_{ij}^2\}$.

El criterio de optimización es la minimización de la fecha cuando el último trabajo abandona el taller. Esto es, la minimización de $C_{\max}$. A continuación se presentan las implicaciones del modelo:

- El número de máquinas es $m_1 \ge 1$, $m_2 \ge 1$; y el número de trabajos es $n > m_1$ y $n > m_2$.
- Todos los trabajos se encuentran disponibles para su ejecución en el momento 0, $\therefore r_j = 0$, $\forall j \in J$.
- Las máquinas son disponibles desde el momento cero.
- Cada máquina procesa no más de un trabajo al mismo tiempo.
- Cada trabajo se procesa al mismo tiempo no más que en una máquina.
- La interrupción de operaciones no es permitida. Una vez que comienza una operación de un trabajo j en una máquina i, esta debe procesarse hasta su finalización.

- Los trabajos (no las operaciones) son independientes entre si, es decir, no hay relaciones de precedencia de ningún tipo entre los trabajos.
- Los tiempos de cambio de partida son incluidos en tiempos de ejecución de operaciones o insignificantes, y no dependen de la secuencia de los trabajos.
- Se asume una capacidad de almacenamiento ilimitada entre las máquinas. Nunca un trabajo queda bloqueado entre dos máquinas.
- Los tiempos p_{ij} de procesamiento de un trabajo j según la ruta tecnológica i se conocen de antemano, son determinísticos y permanecen constantes durante todo el proceso.
- Los tiempos de transporte de los trabajos entre las máquinas son despreciables.

La determinación de la ruta tecnológica para todo trabajo j supone que con respecto a la primera etapa el conjunto de trabajos $J = \{j \mid 1 \le j \le n\}$ ahora se representa como una partición del conjunto J por m_1 disjuntos J_k, tal que $J = \bigcup_{k=1}^{m_1} J_k$, $J_k \neq \varnothing$, $J_\mu \bigcap J_\nu = \varnothing$, $\mu \neq \nu$. Aquí J_k es un subconjunto de los trabajos, cuya primera operación ejecuta la máquina k, $k \in \{1,2,...,m_1\}$.

Analógicamente, con respecto a la segunda etapa, el conjunto J se divide por m_2 disjuntos J_l, tal que $J = \bigcup_{l=1}^{m_2} J_l$, $J_l \neq \varnothing$, $J_\mu \bigcap J_\nu = \varnothing$, $\mu \neq \nu$. Aquí J_l es un subconjunto de los trabajos, cuya segunda operación ejecuta la máquina l, $l \in \{1,2,...,m_2\}$. En el resultado, la segunda operación de todo trabajo $j = (p_j^1, p_j^2)$, $j = \overline{1,n}$, se sujeta a uno de los m_2 procesadores del segundo nivel.

En el caso trivial, cuando $m_1 = m_2 = 1$, el problema considerado representa un Taller de Flujo simple conocido como el problema de Johnson $F2 \,||\, C_{\max}$. Es de interés, cuando el número de máquinas al menos en un grupo es mayor que uno. Estos casos se clasifican en la categoría de $FH2, (Rm^{(i)})_{i=1}^{2} \,||\, C_{\max}$ (Pinedo, 2002).

El trabajo $j \in J$, $j = \overline{1,n}$, se identifica con un par ordenado (p_{Ij}^1, p_{ij}^2), donde p_{Ij}^1 es la duración de la primera operación del trabajo j procesada en la máquina I de la primera etapa, $I = \overline{1,m_1}$; p_{ij}^2 es la duración de su segunda operación procesada en la máquina i de la segunda etapa, $i = \overline{1,m_2}$.

Sea G_{Ii} el conjunto de trabajos con la misma ruta tecnológica , $I = \overline{1,m_1}$, $i = \overline{1,m_2}$.

Se forma el conjunto G_I, $I = \overline{1,m_1}$, de todos los trabajos, cuyas primeras operaciones realiza la máquina I de la primera etapa, y se calcula $A_I = \sum_{j \in G_I} p_{Ij}^1$. De igual forma, Q_i es el conjunto de trabajos, cuyas segundas operaciones realiza la máquina i de la segunda etapa, $i = \overline{1,m_2}$. Sea $B_i = \sum_{j \in Q_i} p_{ij}^2$. Entonces, $G_I = \bigcup_i G_{Ii}$, $G_{Ii} \cap G_{Il} = \varnothing$, $i \neq l$, $i,l \in \{1,2,...,m_2\}$, $I = \overline{1,m_1}$; $Q_i = \bigcup_I G_{Ii}$, $Q_{Ii} \cap Q_{Li} = \varnothing$, $I \neq L$, $I,L \in \{1,2,...,m_1\}$, $i = \overline{1,m_1}$. De tal manera, $\bigcup_I Q_i = \bigcup_i G_i$.

2.5.2 Evaluación de límites

Sea Ω una solución válida del problema. Entonces $C(\Omega)$ es el tiempo de procesamiento de los n trabajos, y el problema formulado anteriormente consiste en minimizar $C(\Omega)$ para el conjunto de soluciones válidas Ω. A continuación se evalúan los límites superior e inferior de la funcional buscada.

Por el límite inferior $C_{\min}$ de $C(\Omega^*)$ se toma:

$$C_{\min} = \max[\max_{1 \le I \le m_1}(A_I + \min_{J_j \in G_I} p_{ij}^1), \max_{1 \le i \le m_2}(B_i + \min_{J_j \in Q_i} p_{ij}^2), \max_{1 \le j \le n}(p_{Ij}^1 + p_{ij}^2)] - 1.$$

Como el límite superior $C_{\max}$ se utiliza la longitud de una solución válida Ω'.

El valor de $C(\Omega')$ se determina mediante el siguiente algoritmo que representa una modificación del algoritmo de Johnson (Burtseva et. al., 2005).

Paso 1. G es la lista de los elementos $j = (p_{Ij}^1, p_{ij}^2)$, ordenados según el decrecimiento de los valores p_{ij}^2; $A_I = 0$, $I = \overline{1,m_1}$; $F_i = 0$, $i = \overline{1,m_2}$; $j = 0$.

Paso 2. $j = j + 1$; para el elemento $j = (p_{Ij}^1, p_{ij}^2)$ poner $A_I = A_I + p_j^I$.

Paso 3. Si $A_I \geq F_i$, entonces $F_i = A_I + q_j^i$, de lo contrario, $F_i = F_i + q_j^i$.

Paso 4. Si $j < n$, entonces el Paso 2.

Paso 5. Encontrar $C(\Omega') = \max(F_i \mid 1 \leq i \leq m_2)$.

Paso 6. Si $C(\Omega') = C_{\min} + 1$, entonces la solución Ω' es óptima.

La complejidad del Paso 1 del algoritmo se evalúa mediante la complejidad del procedimiento de la ordenación y es igual a $O(n \log_2 n)$. La ejecución de los Pasos 2-4 requiere el tiempo $O(n)$. La complejidad del Paso 5 es $O(n)$. Entonces, la complejidad del procedimiento del cálculo de $C(\Omega')$ es $O(n \log n)$.

2.5.3 Algoritmo

Primero se describen las ideas, en las cuales se basa el algoritmo de la búsqueda de la solución Ω^0.

Dado que las máquinas de la primera etapa funcionan de manera independiente la una de la otra, entonces las primeras operaciones de los trabajos se ejecutarán durante el tiempo $A_K = \max(A_I \mid 1 \leq I \leq m_1)$. En cualquier solución válida, incluyendo la óptima, el trabajo, el cual finaliza la ejecución de las primeras operaciones, pertenece al conjunto G_K. El mismo trabajo también es el último en la sucesión de los trabajos procesados en la máquina correspondiente de la segunda etapa. Al encontrar el trabajo para la última posición en la sucesión de los trabajos para la máquina K de la solución óptima, se resta de A_K la duración de primera operación de aquel trabajo y se determina un nuevo valor de A_K. Este trabajo se ubica en la última posición de trabajos para la máquina correspondiente de segunda etapa. En el caso contrario, el tiempo de su ocupación, al menos, no se cambiará.

De nuevo, se determina $\max(A_I \mid 1 \leq I \leq m_1)$, después se busca el conjunto G_L, $L = \{1, 2, ..., m_1\}$, el cual contiene este trabajo. El procedimiento se repite hasta que los valores de A_I, $I = \overline{1, m_1}$, igualen al cero.

La dificultad principal en la búsqueda de la solución óptima Ω^* consiste en escoger un trabajo entre los procesados en la misma máquina de la segunda etapa.

Según el esquema dicotómico, el algoritmo contiene un número finito de iteraciones. La iteración s del algoritmo para un límite C^s construye una solución Ω^s, la cual cumple la desigualdad $C(\Omega^s) \le C^s$.

El algoritmo finaliza cuando resulta imposible mediante sus operaciones construir una solución Ω^{s+1} para algún valor $C^{s+1} < C^s$. En este caso, como la solución Ω^0 se escoge la solución Ω^s construida en la iteración anterior.

El algoritmo funciona de la siguiente manera: Primero, para un valor fijo C^s se intenta encontrar la solución Ω^s mediante la creación de m_1 permutaciones π_I^s. La permutación π_I^s establece el orden del procesamiento del conjunto de trabajos G_I en la máquina I del primer grupo en la iteración s $I = \overline{1, m_1}$. Además, las acciones del algoritmo determinan las m_2 permutaciones Ψ_i^s. La permutación Ψ_i^s corresponde a la sucesión del procesamiento de los trabajos en la máquina i del segundo grupo, $I = \overline{1, m_2}$.

En el caso de un éxito, el procedimiento se repite para el valor C^{s+1}. En caso de un fracaso, el algoritmo se termina. Su resultado es la solución obtenida para el valor anterior C^{s-1}.

A continuación se presenta un algoritmo eficiente que actúa dentro del esquema del método de dicotomía. En cada iteración s el algoritmo forma las permutaciones de solución del problema $C^s = C(\Omega')$. La organización de los cálculos de acuerdo al método de dicotomía permite evaluar la cercanía de la solución al óptimo mediante la longitud del intervalo obtenido en la iteración s.

Algoritmo que minimiza $C_{\max}$ en un TFH2 con rutas tecnológicas fijas (M+m):

Paso 1. $s = 0$, $C^s = C(\Omega')$.

Paso 2. $s = s + 1$, $C^s = (C_{\min} + C^s)/2$.

Paso 3. G_I es un conjunto con n_I elementos $j=(p_{Ij}^1, p_{ij}^2)$, $G_I^s = G_I$, $A_I^s = \sum_{j\in G_I} p_{Ij}^1$, $I=\overline{1,m_1}$; $B_i = 0$, $i=\overline{1,m_2}$.

Paso 4. Se busca tal conjunto G_L^s, que $A_L^s = \max(A_I^s \,|\, 1 \le I \le m_1)$; $l = n_L$.

Paso 5. Si $A_L^s = 0$, entonces está construida la solución Ω^s de la longitud $C(\Omega^s) \le C^s$; ir al Paso 2.

Paso 6. Si para cada elemento $j=(p_{Ij}^1, p_{ij}^2) \in G_L^s$, $j=\overline{1,l}$, $i \in \{1,2,...,m_2\}$, se cumple la desigualdad

$$C^s < A_L^s + p_{ij}^1 + B_i,\ j \in G_{Li}, \tag{3}$$

entonces ir al Paso 9.

Paso 7. Entre todos componentes del conjunto G_L^s, que no cumplen la desigualdad (3), se determina un elemento $l=(p_{Ll}^1, p_{il}^2) \in G_{Li}$ con el valor máximo de p_{Ll}^1. Este elemento se coloca en la posición l en la permutación π_L^s; $G_L^s = G_L^s \setminus \{l\}$; $A_L^s = A_L^s - p_{Ll}^1$; $B_i = B_i + p_l^i$; $l = l-1$, $n_L = l$; ir al Paso 4.

Paso 8. $s = s+1$, $C^s = (C_{\min} + C^s)/2$.

Paso 9. Q_i es el conjunto con n_i elementos $j=(p_{Ij}^1, p_{ij}^2)$, $Q_i^s = Q_i$, $B_i^s = \sum_{J_j \in Q_i} p_{ij}^1$, $A_I = 0$, $I=\overline{1,m_1}$; $l_i = 0$, $i=\overline{1,m_2}$.

Paso 10. Se busca tal conjunto Q_k^s, que $B_k^s = \max(B_i^s \,|\, 1 \le i \le m_2)$, $l = n_k - l_k$.

Paso 11. Si $B_k^s = 0$, entonces está construida la solución Ω^s de la longitud $C(\Omega^s) \le C^s$, ir al Paso 8.

Paso 12. Si para cada elemento $j=(p_{Ij}^1, p_{kj}^2)$, $j=\overline{1,l}$, $I \in \{1,2,...,m_1\}$, se cumple la desigualdad

$$C^s < B_k^s + p_{Ij}^1 + A_I,\ J_j \in Q_{Ik}, \tag{4}$$

entonces, fin: está construida la solución Ω^0 de una longitud que no supera C^{s-1}, $\Omega^0 = \Omega^{s-1}$.

Paso 13. Entre todos elementos del conjunto Q_k^s, que no cumplen la desigualdad (2), se determina el elemento $l=(p_{Il}^1, p_{kl}^2)\in G_{ik}$ con el valor máximo de p_{kl}^2. Este elemento se coloca en la posición l_k+1 de la permutación Ψ_k^s; $Q_k^s=Q_k^s\setminus\{j_l\}$; $B_k^s=B_k^s-p_{kl}^2$; $A_I=A_I+p_{Il}^1$; $l_k=l_k+1$; ir al Paso 10.

La complejidad temporal del algoritmo depende del número de iteraciones y operaciones elementales en cada iteración.

En el peor caso, cuando $m_1=1$ o $m_2=1$, los Pasos 6, 7, 12, 13 del algoritmo se repiten no más que n veces y requieren no más que $\sum_{j=1}^{n} j+\sum_{j=1}^{n-1} j$ operaciones de comparación. El número de iteraciones no supera un valor $1-[-\log_2\rho]$, donde $\rho=C_{\max}-C_{\min}$. Por lo tanto, la complejidad del algoritmo es $O(n^2)$.

A continuación se muestra un ejemplo.

Sean $M=2$, $m=3$, $n=6$;

$j_1=(p_{11}^1, p_{11}^2)$, $p_{11}^1=3$, $p_{11}^2=5$;

$j_2=(p_{12}^1, p_{12}^2)$, $p_{12}^1=1$, $p_{12}^2=1$;

$j_3=(p_{13}^1, p_{23}^2)$, $p_{13}^1=5$, $p_{23}^2=1$;

$j_4=(p_{14}^1, p_{34}^2)$, $p_{14}^1=2$, $p_{34}^2=3$;

$j_5=(p_{25}^1, p_{15}^2)$, $p_{25}^1=2$, $p_{15}^2=1$;

$j_6=(p_{26}^1, p_{26}^2)$, $p_{26}^1=9$, $p_{26}^2=5$.

Entonces,

$G_1=\{j_1, j_2, j_3, j_4\}$, $G_2=\{j_5, j_6\}$;

$Q_1=\{j_1, j_2, j_5\}$, $Q_2=\{j_3, j_6\}$, $Q_3=\{j_4\}$;

$A_1 = 3+1+5+2 = 11,\ A_2 = 2+9 = 11;$

$B_1 = 5+1+1 = 7,\ B_2 = 1+5 = 6,\ B_3 = 3.$

Se busca el límite inferior:

$C_{\min} = \max[\max(11+1, 11+1), \max(7+1, 6+5, 3+2),$
$\max(3+5, 1+1, 5+1, 3+2, 2+1, 9+5)] - 1 = 13.$

Sea ya construida una solución válida de longitud 17. Se ejecuta el algoritmo para construir la solución Ω^1. El límite superior es $C^1 = 15$.

Los valores necesarios para siguiente iteración son:

$G_1^1 = G_1,\ G_2^1 = G_2;\ A_1^1 = A_2^1 = 11;$

$B_1 = B_2 = B_3 = 0.$

Se busca $\max(A_1^1, A_2^1) = 11$. Puesto que $A_1^1 = A_2^1$, entonces se escoge cualquiera de los conjuntos: G_1^1 o G_2^1, por ejemplo, G_1^1. Se asigna $l = n_1 = 4$ y se verifica el cumplimiento de n_1 desigualdades (3):

$15 < 11+5+0,\quad j_1 \in G_{11} = \{j_1, j_2\},$

$15 > 11+1+0,\quad j_2 \in G_{11} = \{j_1, j_2\},$

$15 > 11+1+0,\quad j_3 \in G_{12} = \{j_3\},$

$15 > 11+3+0,\quad j_4 \in G_{13} = \{j_4\}.$

Puesto que $p_{12}^1 = 1,\ p_{13}^1 = 5,\ p_{14}^1 = 2$, entonces en la última posición de permutación π_1^1 se ubica el elemento $j_3 = (p_{13}^1, p_{23}^2)$.

En el Paso 7 del algoritmo se obtiene:

$G_1^1 = \{J_1, J_2, J_4\}.\ A_1^1 = 11-5 = 6,\ B_2 = p_{23}^1 = 1,\ l = 3,\ n_1 = 3.$

Ahora $\max(A_1^1, A_2^1) = \max(6, 11) = A_2^1 = 11,\ l = n_2 = 2.$

Son dos desigualdades:

$15 > 11+1+0,\quad J_5 \in G_{21} = \{j_5\},$

$15 < 11+5+1, \quad j_6 \in G_{22} = \{j_6\}.$

Se coloca el elemento $j_5 = (p^1_{25}, p^2_{15})$ en la última posición de la permutación π^1_2; $G^1_2 = \{j_6\}$, $A^1_2 = 11-2 = 9$, $B_1 = 1$, $l = 1$, $n_2 = 1$.

Otra vez se busca

$\max(A^1_1, A^1_2) = \max(6,9) = A^1_2 = 9$

y se verifica el cumplimiento de la desigualdad (1) para el elemento j_6:

$15 = 9+5+1, \quad j_6 \in G_{22} = \{j_6\}.$

De tal manera, el elemento $j_6 = (p^1_{26}, p^2_{26})$ se ubica en la permutación π^1_2 antes del elemento

$j_5 = (p^1_{25}, p^2_{15})$; $G^1_2 = \varnothing$, $A^1_2 = 0$.

$B_2 = 1+5 = 6$, $l = 0$, $n_2 = 0$.

Se repite el Paso 4 del algoritmo:

$\max(A^1_1, A^1_2) = \max(6,0) = A^1_1 = 6.$

Para los tres elementos restantes j_1, j_2, j_4 del conjunto G^1_1 se tienen las siguientes desigualdades:

$15 > 6+5+1, \quad j_1 \in G_{11},$

$15 > 6+1+1, \quad j_2 \in G_{11},$

$15 > 6+3+0, \quad j_4 \in G_{13}.$

De acuerdo con el Paso 7, el elemento $j_1 = (p^1_{11}, p^2_{11})$, para el cual

$\max(p^1_{11}, p^1_{12}, p^1_{14}) = \max(3,1,2) = 3,$

se coloca en la tercera posición de la permutación π^1_1.

Repitiendo las operaciones analógicas, se llega a la solución Ω^1, cuya longitud no supera 15. En la solución Ω^1, el orden de procesamiento de los trabajos en la máquina 1 del primer grupo es (2, 4, 1, 3), y en la máquina 2 del mismo grupo es (6, 5).

Se fija el nuevo límite $C^2 = 14$ y volvemos a repetir los Pasos 3-7. De nuevo, en la última posición en π_1^2 se ubica el elemento $j_3 = (p_{13}^1, p_{23}^2)$, y el elemento $j_5 = (p_{25}^1, p_{15}^2)$ se asigna a la segunda posición de la permutación π_2^2. Se busca $\max(A_1^2, A_2^2) = A_2^2 = 9$ y se ejecuta el Paso 6:

$14 < 9+5+1, \quad j_6 \in G_{22} = \{j_6\}$.

En este caso, es necesario ir al Paso 9.

Se tienen

$Q_1^2 = Q_1,\ Q_2^2 = Q_2,\ Q_3^2 = Q_3, B_1^2 = 7,$

$B_2^2 = 6,\ B_3^2 = 3,\ A_1 = A_2 = 0,$

$l_1 = l_2 = l_3 = 0$.

Ahora

$\max(B_1^2, B_2^2, B_3^2) = \max(7,6,3) = B_1^2$.

Se asigna $l = n_1 - l_1 = 3 - 0 = 3$ y se verifica el cumplimiento de las desigualdades (2) para los elementos del conjunto $Q_1^2 = \{j_1, j_2, j_5\}$:

$14 > 7+3+0, \quad j_1 \in G_{11},$

$14 > 7+1+0, \quad j_2 \in G_{11},$

$14 > 7+2+0, \quad j_5 \in G_{21}$.

A la primera posición de la permutación Ψ_1^2 se asigna el elemento j_1, dado que

$p_{11}^2 = \max(p_{11}^2, p_{12}^2, p_{15}^2) = \max(5,1,1) = 5$.

Ahora

$Q_1^2 = \{j_2, j_5\},\ B_1^2 = 7 - 5 = 2,\ A_1 = 3,\ l_1 = 1$.

Se busca

$\max(B_1^2, B_2^2, B_3^2) = \max(2,6,3) = B_2^2 = 6;\ l = 2 - 0 = 2$.

Para los elementos del conjunto $Q_2^2 = \{j_3, j_6\}$ se tienen las siguientes expresiones:

$14 = 6+5+3, \quad j_3 \in G_{12},$

$14 < 6+9+0, \quad j_6 \in G_{22}$.

De tal manera, el elemento $j_3 = (p_{13}^1, p_{13}^2)$ se ubica en la primera posición de la permutación Ψ_2^2,

$Q_2^2 = \{j_6\}$, $B_2^2 = 6-1 = 5$,

$A_1 = 3+5 = 8$, $l_2 = 1$.

Otra vez,

$B_2^2 = \max(B_1^2, B_2^2, B_3^2) = \max(2,5,3) = 5$.

Se obtiene

$14 = 5+9+0$, $j_6 \in G_{22}$.

El elemento j_6 se ubica en la segunda posición de la permutación Ψ_2^2, $Q_2^2 = \varnothing$, $B_2^2 = 0$, $A_2 = 9$, $l_2 = 2$.

Ahora,

$\max(B_1^2, B_2^2, B_3^2) = \max(2,0,3) = B_3^2$, $l = 1$.

Puesto que se cumple la desigualdad

$14 > 3+2+8, \quad j_4 \in G_{13}$,

entonces, el elemento j_4 es el único en la permutación Ψ_3^2. Se tienen que $Q_3^2 = \varnothing$, $B_3^2 = 0$, $A_1 = 8+2 = 10$, $l_3 = 1$.

Se encuentra

$\max(B_1^2, B_2^2, B_3^2) = \max(2,0,0) = B_1^2$; $l = 3-1 = 2$.

Para los elementos del conjunto $G_2^1 = \{j_2, j_5\}$ se obtienen las siguientes desigualdades:

$14 > 2+1+10, \quad j_2 \in G_{11}$,

$14 > 2+2+9, \quad j_5 \in G_{21}$.

Puesto que $p_{12}^2 = p_{15}^2$, entonces, en el Paso 13 se escoge cualquiera de los elementos j_2, j_5, por ejemplo j_2. El elemento j_2 se ubica en la segunda posición de Ψ_1^2. En este caso,

$Q^2 = \{j_5\}$, $B_1^2 = 2-1=1$, $A_1 = 10+1=11$, $l_1 = 2$.

Sólo resta comprobar, si es posible colocar el elemento j_5 en la posición 3 en Ψ_1^2:

$14 > 1+2+9$, $j_5 \in G_{21}$.

Se ubica el elemento j_5 en la tercera posición de la permutación Ψ_1^2; $Q_1^2 = \varnothing$, $B_1^2 = 1-1=0$, $A_2 = 11$, $l_1 = 3$.

Puesto que $\max(B_1^2, B_2^2, B_3^2) = 0$, entonces, está construida la solución Ω^0 de longitud 14. Se observa, que la solución encontrada es óptima. De acuerdo con la solución Ω^0, la máquina 1 de la segunda etapa procesa el grupo los trabajos j_1, j_2, j_5 en el orden $\Psi_1^2 = (1,2,5)$; la máquina 2 procesa los trabajos j_3 y j_6 en el orden $\Psi_2^2 = (3,6)$; $\Psi_3^2 = (4)$. La Figura 2.12 muestra gráficamente el TFH2 construido.

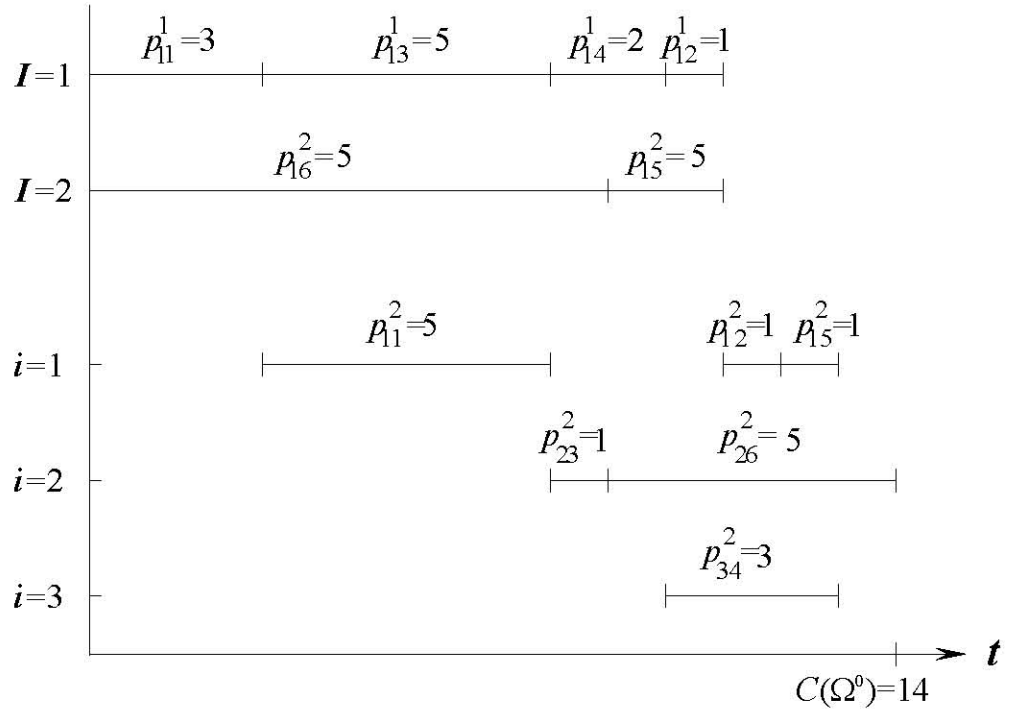

Figura 2.12. TFH2 de longitud óptima

2.6 Conclusiones del capítulo

El problema de TFH2 es $\mathcal{NP}$-Duro y por tanto no existe algoritmo polinomial que obtenga una solución óptima para el caso general. Los algoritmos exactos se desarrollan sólo para unos casos particulares. Existe una variedad de métodos para encontrar una solución aproximada: métodos de programación matemática (programación dinámica, ramificación y acotamiento), métodos heurísticos (modificaciones del algoritmo de Johnson, algoritmos voraces, etc.) y metaheurísticos (recocido simulado, búsqueda tabú, algoritmos genéticos, colonia de hormigas), así como su combinación.

En los últimos años se han hecho populares los métodos evolutivos de inteligencia artificial, los cuales son metaheurísticas y por su naturaleza representan una combinación de heurísticas con enfoque probabilístico. Estos métodos proporcionan buenos resultados, los cuales con frecuencia superan a otros tipos de algoritmos por la cercanía al óptimo. Lamentablemente, el costo de tiempo de solución es mucho más alto que en caso de los

algoritmos de heurísticas simples. La elección de un algoritmo en específico debe ser realizada después de su comparación con otros que resuelven el mismo problema.

El método de dicotomía, representa un esquema eficiente y económico para algoritmos de optimización combinatoria, en particular, para la planificación de los trabajos. Las soluciones se localizan rápidamente si los cálculos combinan el acercamiento al óptimo de acuerdo al método de dicotomía con la ejecución del algoritmo M+m en cada iteración dicotómica. La exactitud de la solución se evalua analíticamente por el tamaño del intervalo que proporciona la última iteración valida.

3 DISEÑO DE UN SISTEMA COMPUTACIONAL PARA LA COMPARACIÓN DE ALGORITMOS

3.1 Evaluación de calidad de algoritmos en sistemas computacionales

La calidad de los algoritmos se caracteriza por la cercanía de la solución al óptimo (la exactitud) y la complejidad temporal (la función asintótica). Para escoger un algoritmo entre varios que resuelven el mismo problema estos deben compararse. Los métodos principales para la comparación de algoritmos con respecto a su exactitud y complejidad son: el análisis teórico y la simulación.

Por la alta complejidad de los problemas de la planificación, su análisis teórico es difícil. El análisis teórico de un algoritmo supone la obtención de la formula de la aproximación al óptimo o limites inferior y superior más cercanos. Actualmente la mayoría de los investigadores prefieren analizar la calidad de un algoritmo a través de la simulación.

La simulación se realiza a través de un experimento computacional. Entre las publicaciones relacionadas con TFH2 se encuentran distintos diseños de experimentos.

Uno de los autores principales, J. Gupta, en su articulo *"Two-Stage, Hybrid Flowshop Scheduling Problem"* (1988) explica la forma en la que realizó el experimento computacional con el fin de evaluar un algoritmo heurístico de minimización del *makespan* en el problema de TFH2 con dos máquinas en la primera etapa y una máquina en la segunda. La eficiencia del algoritmo se demuestra de forma empírica. El algoritmo heurístico fue programado en FORTRAN. Se resolvieron 380 problemas en el rango de 5 a 100 trabajos. Los tiempos de procesamiento de los trabajos se generaron siguiendo la distribución uniforme discreta y un rango de (1, 99).

De forma similar, J. Gupta en su articulo *"Schedule for a two-stage hybrid flowshop with parallel machines at the second stage"* (1991) realizó un experimento computacional para evaluar un algoritmo heurístico que minimice el *makespan* para TFH2 con una máquina en la primera etapa y 2 máquinas en la segunda. El algoritmo fue programado en FORTRAN. Se resolvieron 760 problemas con el rango de trabajos de 5 a 100, divididos en dos conjuntos de datos. Para los tiempos de ejecución de trabajo se utilizó la distribución uniforme discreta. La variable de tiempo para el primer conjunto se generó en el rango (1, 99) para ambas etapas. Para el segundo conjunto de datos, los tiempos de procesamiento de la primera etapa se generaron en el rango (1, 49), mientras que los tiempos de procesamiento de la segunda etapa se generaron en el rango (1, 99).

Bo Chen en su artículo *"Analysis of Classes of Heuristics for Scheduling a Two-Stage Flow Shop with Parallel Machines at One Stage"* (1995) considera el modelo con m máquinas idénticas en la primera etapa y una máquina en la segunda. Selecciona tres clases de heurísticas: la clase $\mathscr{H}$ de heurísticas, basadas en la secuencia de Johnson en la máquina m, su extensión $\mathscr{H}^{-}$ y $\mathscr{G}$. En la última los trabajos se procesan en la máquina m de acuerdo a su terminación en la primera etapa. El autor encontró las formulas de aproximación de $C_{\max}$ al óptimo en dependencia de valores m y n. Realizó el análisis teórico comparativo de las clases heurísticas. Reforzó el análisis teórico con un experimento computacional. Seleccionó cuatro heurísticas que representan las tres clases mencionadas anteriormente tres valores de m (2, 5, 8) y cinco valores de n (10, 15, 30, 50, 100). Los tiempos de trabajo, a_j y b_j, $j=\overline{1,n}$, se generan de forma aleatoria según la distribución uniforme discreta en el intervalo $1 \le a_j, b_j \le 100$ para la primera categoría I_1, $1 \le a_j \le 100m$ y $1 \le b_j \le 100$ para la segunda categoría I_2. Para generar tiempos a_j de la categoría I_1 se aplica el coeficiente m, el cual equilibra la carga de las máquinas de la primera y segunda etapa. En total se generaron aproximadamente 500 problemas. El análisis empírico consistió en la estimación de la desviación entre $C_{\max}$ y el limite inferior para cada problema.

A. Guinet en su articulo *"A computational study of heuristic for two-stage flexible flowshop"* (1996) realizó la simulación del ambiente de una manufactura con dos centros de máquinas, los cuales representan dos etapas consecutivas de producción. Cada centro está

compuesto por múltiples máquinas en paralelo. En cada centro el trabajo debe ser procesado en cualquiera de las máquinas. Los trabajos son independientes, es decir no existe relación de precedencia. Se permite la espera entre etapas de un trabajo. El espacio de almacenamiento entre las etapas es ilimitado. El objetivo es minimizar el tiempo de completar los trabajos. Este problema es estrictamente $\mathcal{NP}$-duro.

A. Guinet describe tres métodos para resolver el problema y siete reglas para la fase de asignación: basada en la regla de Johnson, tres modificaciones de SPT (Short Process Time), tres modificaciones de LPT (Long Process Time). La eficiencia de las heurísticas se evalúa a través de la comparación de cada variante de la heurística con el mejor de tres límites inferiores característicos para el modelo (Gupta, 1988), (Gupta, 1991).

Realizó un experimento computacional, el cual mostró que la regla de Johnson (1954) obtiene la mejor asignación de los trabajos. En total se resolvieron 3456 problemas. Los datos de entrada fueron generados utilizando la distribución uniforme discreta. El número de trabajos a procesar se tomó de n = 50, 100, 150, 200, 250 y 300. El número de trabajos corresponde al número de parte y cada parte era solicitada una vez. Para cada centro de trabajo se seleccionó el número de máquinas, el cual podría tomar valores de 2,3 y 4 por centro. Se estudiaron 54 combinaciones de parámetros, cada uno con dos rangos de datos. Para cada problema se realizaron 32 pruebas a cada rango de datos.

Los rangos de datos corresponden a dos escenarios para el proceso de los trabajos. En el primero, los tiempos de procesamiento se generaron en el rango [10, 30]. Esto corresponde al ambiente donde los trabajos están agrupados en familias grandes de acuerdo con las similitudes de procesamiento, incluyendo tiempo de procesamiento. Este es el caso para manufactura de celulares.

En el segundo escenario, se permiten variaciones de los tiempos de procesamiento. Estos tiempos de procesamiento se toman en el rango [10, 100]. El valor de parámetro usado es realista en el sentido de que se basó en datos de un reporte de la instalación actual de la empresa de manufactura donde se realizó la prueba. Los algoritmos se programaron en Pascal y se ejecutaron en una computadora SUN3 Workstation bajo el sistema operativo UNIX. No se reportan los tiempos de CPU debido a su insignificancia en comparación con la complejidad del problema.

Ceyda Oguz en su articulo "*Heuristic Algorithms for Scheduling Multi Layer Computer Systems*" (1997) examina tres heurísticas para resolver problemas en sistemas de multiprocesadores con capas múltiples, generalmente empleados en aplicaciones de tiempo real como visión por computadora y robótica. La aplicación de proceso de visión por computadora requiere arquitecturas especiales. Muchas plataformas de cómputo, construidas para este propósito emplean múltiples capas de arreglos de procesadores, cada una dedicada para el procesamiento de una imagen. Su estudio representa un TFH2. Para cada Tarea i de Multiprocesador MPT(i,j), i=1,2,…,n, j=1,2, los tiempos de procesamiento en la etapa j, se generan con la distribución uniforme con los rangos [1,m] y [1,30], respectivamente. El número de trabajos fue seleccionado de n = 30, 50, 70, 100. El número de procesadores fue elegido de $m_1 = 2$, $m_2 = 2^k$ y $m_1 = m_2 = 2^k$, $k = 1, 2, 3, 4$. En cada diseño, correspondiente a su combinación de n y (m_1, m_2), se generaron 30 problemas. En el artículo se deducen cuatro límites inferiores. La eficiencia de las heurísticas se evalúa a través de la desviación entre la solución y el límite inferior específico que corresponde al conjunto de datos de entrada.

Ling-Huey Su en su articulo *"A hybrid two-stage flowshop with limited waiting time constraints*" (2003) considera un TFH2 formado por varios procesadores de lotes (*batch processors*) en la primera etapa y un único procesador en la segunda. Los trabajos se procesan por lotes. Cada trabajo de un lote se procesa simultáneamente. Una vez que el proceso comienza, ningún trabajo se libera del grupo de procesadores hasta que el lote completo sea procesado. Los tiempos de espera para procesarse en la segunda etapa no superan un límite superior. En el artículo se formula el modelo en términos de la programación entera mixta.

Los tiempos de procesamiento para la segunda etapa se generan de acuerdo con la distribución uniforme en el rango [1, 10]. El tiempo límite de espera para procesarse en la segunda etapa se genera según la distribución uniforme con el rango de [0, 15]. En el análisis estadístico se consideran dos factores principales, los cuales afectan a la solución: la capacidad de los procesadores de la primera etapa y tiempo de procesamiento de lotes en la primera etapa. La solución se compara con los límites inferior y superior recibidos a través del modelo de programación entera mixta.

Para verificar la calidad de la solución utilizaron la siguiente formula:

$$\text{Calidad de la solución} = \frac{\left[\left(2 \times \text{óptimo o límite inferior}\right) - \text{heurística}\right]}{\text{óptimo o límite inferior}} \times 100\%$$

Para estimar la exactitud del tiempo de ejecución, cada prueba se procesó continuamente 100000 veces y se calculó el promedio de tiempo de procesador utilizado.

El experimento fue realizado en una computadora personal con un procesador Pentium a 400 MHz. Los algoritmos heurísticos fueron programados en el lenguaje C++.

El artículo de Rubén Ruiz y Concepción Maroto *"Evaluación de heurísticas para el problema del taller de flujo"* (2003) incluye un análisis de métodos de evaluación de heurísticas para el problema $FHm2||C_{\max}$, utilizados por otros autores. Varios métodos mencionados tienen inconvenientes; no utilizan conjuntos de datos estándares y/o los sistemas de recursos utilizados no son los mismos. Lo anterior trae como consecuencia que los resultados no sean comparables ni generalizables. Las comparaciones se hacen con unos cuantos métodos y por lo regular son siempre los mismos. Los autores proponen una comparativa extensa y actualizada de heurísticas y metaheurísticas para problemas de Taller de Flujo permutacional. Se evalúan 25 heurísticas en un diseño estandarizado utilizando el conjunto de problemas más difíciles a resolver desarrollado por Éric Taillard (1993). Los parámetros principales toman valores: el número de trabajos n = 20, 50, 100, 200; el número de etapas m = 5, 10, 20. El número de máquinas por etapa esta distribuido de manera uniforme entre uno y tres máquinas por etapa. Se consideran once combinaciones de m y n (con excepción de 200×5). Para cada combinación se resuelven 10 diferentes problemas donde los tiempos de procesamiento son distribuidos de manera uniforme en el rango [1, 99].

La variable de respuesta se basa en el porcentaje de incremento de la solución heurística con respecto a la mejor solución conocida. Se calcula de acuerdo a la fórmula:

$$\%\ \text{Incremento} = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100,$$

donde Heu_{sol} es el mejor resultado obtenido por un algoritmo dado, $Best_{sol}$ es la mejor solución conocida.

El experimento fue realizado en un cluster de 4 computadoras PC/AT con procesadores Athlon XP 1600+ y 512 MB de memoria.

Los resultados del experimento se analizaron por medio del Análisis de Varianza multifactorial (ANOVA) probando tres hipótesis: la normalidad, la homogeneidad de la varianza e independencia de residuos. El nivel de confianza utilizado fue de 95%.

3.2 Esquema general del sistema computacional

En un problema de TFH intervienen varios parámetros como:

- El número de máquinas que forma el modelo.
- Tipo de máquinas (paralelas idénticas, paralelas relacionadas o no relacionadas).
- Velocidad de las máquinas.
- Distribución de las máquinas en el modelo de recursos.
- Número de etapas en el modelo.
- Número de trabajos a procesar.
- Número de operaciones que necesita cada trabajo para ser completado.
- Ruta tecnológica de los trabajos.

El sistema computacional PLARETF se utiliza para la investigar algoritmos que resuelven distintos problemas en Talleres de Flujo. La Figura 3.1 presenta el esquema general del sistema computacional para ejecutar los algoritmos de la planificación de trabajos. Tal sistema permite la implementación independiente de algoritmos y utilización de cualquier tipo de técnica, desde algoritmos constructivos hasta evolutivos. El sistema utiliza la misma entrada de datos para distintos algoritmos, siempre y cuando los algoritmos sean compatibles con el problema.

Los pasos generales para cualquier algoritmo de Taller de Flujo son: Entrada de datos, incluyendo las características del modelo (*Input*), Asignación de los trabajos a las máquinas del modelo (*Assignment*), Secuenciación de los trabajos en las máquinas (*Sequencing*), Programación de tareas (*Timing*) y Salida de resultados (*Output*).

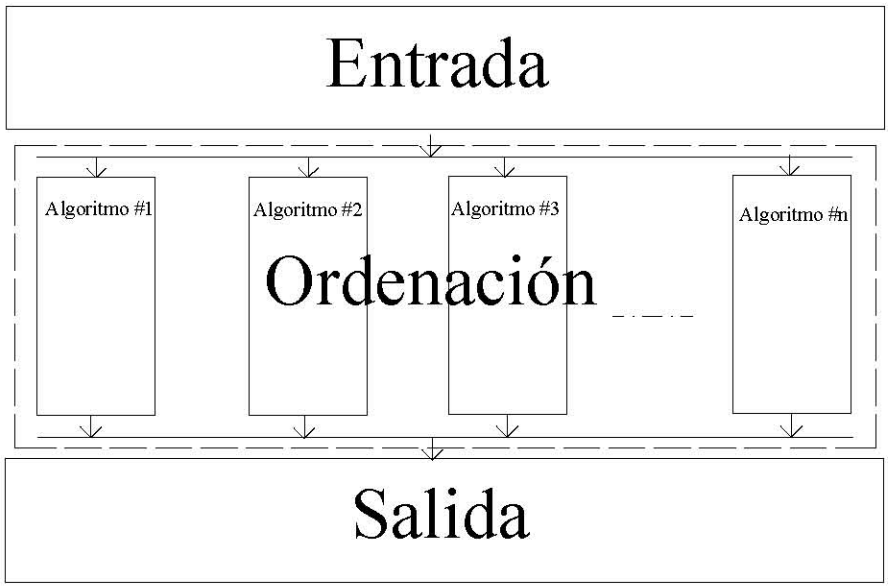

Figura 3.1. Esquema general del sistema computacional

A continuación se describen las secciones que conforman el sistema computacional (Romero et al., 2006).

Entrada: Esta sección se encarga de recibir los datos de entrada y convertirlos a estructuras genéricas, con las cuales se ejecutan los algoritmos. La entrada recibe los parámetros del problema a resolver: los trabajos y sus tiempos de procesamiento, número de máquinas, número de etapas, tiempos de ajuste, etc. Además se obtiene la configuración del sistema para manipular su comportamiento del simulador.

Ordenación: Esta sección contiene una colección de algoritmos diseñados para resolver problemas de planificación de trabajos en Talleres de Flujo (TFH, TFF, Taller Abierto, Taller de Trabajo, etc.).

Salida: Esta sección se encarga de recibir el resultado obtenido por el algoritmo y generar archivos con la información necesaria para la presentación y análisis de resultados.

El sistema PLARETF se implementó en el lenguaje de programación ANSI-C para facilitar la portabilidad con distintos sistemas operativos (como Windows, UNIX y GNU\LINUX) y plataformas de Hardware.

3.3 Entrada

La entrada del sistema está formada por archivos de datos, dos archivos de configuración del sistema y los archivos de configuración propios de los algoritmos. Para obtener la solución de un problema se requiere conocer sus características y estas deben ser ingresadas como parte de la entrada del sistema. Los datos necesarios son (Tabla 3.1):

Tabla 3.1. Notaciones implementadas en el sistema

Notación	Implementación en el sistema
m	**int m;**
n	**int n;**
k_{max}	**int kmax;**
m_1	**int *number_machines_stage;** **number_machines_stage=(int*)malloc(m*sizeof(int));**
m_2	

J_j	Matriz formada por los tiempos de ejecución de los trabajos en cada etapa;
	job_list representa J_j.
	struct job_param {
	long *exec_time;
	long release_time;
	long due_date;
	long dead_line;
	long finish_time;
	long start_time;
	};
	Lista de trabajos:
	struct job_param *job_list;
	int s;
	job_list = (struct job_param*)malloc(n*sizeof(struct job_param));
	for (s = 0; s < n; s++)
	job_list[s].exec_time =(long*)malloc(m*sizeof(long));

El sistema recibe los datos para el experimento, a través de un archivo de texto, el cual está basado en un patrón de datos obtenido por el generador de números

pseudoaleatorios introducido por Éric Taillard (1993). La Figura 3.2 muestra un archivo de datos de entrada para el problema $FH2,(RM^{(i)})_{i=1}^{2} \| C_{\max}$ con subconjuntos de máquinas por etapa $M^1 = \{1\}$, $M^2 = \{1,2\}$, y 20 trabajos. El ejemplo de un archivo completo se muestra en el ANEXO A.

```
20 3 2
1 2
  0 54  1 83  2 83
  0 15  1 71  2 71
  0 77  1 36  2 36
  0 53  1 38  2 38
  0 27  1 87  2 87
  0 76  1 91  2 91
  0 14  1 29  2 29
  0 12  1 77  2 77
  0 32  1 87  2 87
  0 68  1 94  2 94
  0 79  1  3  2  3
  0 11  1 99  2 99
  0 56  1 70  2 70
  0 99  1 60  2 60
  0  5  1 56  2 56
  0  3  1 61  2 61
  0 73  1 75  2 75
  0 47  1 14  2 14
  0 21  1 86  2 86
  0  5  1 77  2 77
SSD
M0
  0 39 46 28 43 45 34 38 34 32 47 44 48 28 32 28 45 33 47 42
 49  0 45 25 27 50 39 43 50 40 26 39 25 40 43 44 37 28 30 47
 26 45  0 29 48 37 28 48 36 40 30 39 41 26 25 41 35 41 37 31
 44 44 35  0 42 40 32 42 45 48 28 40 37 47 47 27 34 35 39 35
 38 45 38 32  0 40 39 30 47 38 33 38 35 43 28 47 43 27 37 37
 47 40 29 42 32  0 30 26 42 31 46 36 34 25 41 27 38 25 48 35
 47 43 36 50 27 47  0 42 33 25 30 48 36 37 31 32 36 50 33 34
 41 36 34 36 27 47 26  0 25 25 37 41 40 32 49 46 32 26 25 38
 42 35 28 42 36 31 41 41  0 32 45 42 46 50 34 28 29 45 36 31
 44 47 40 41 41 46 48 39 41  0 34 25 25 36 41 34 37 41 26 33
 38 50 46 37 32 33 44 25 49 26  0 49 43 41 33 50 26 35 38 46
 49 43 34 28 30 44 37 38 34 45 30  0 38 43 36 31 45 50 40 34
 27 50 43 30 34 34 42 49 49 35 25 38  0 31 48 31 25 42 49 26
 49 30 39 34 25 30 28 28 42 46 45 25 39  0 27 30 28 42 26 29
 38 47 47 25 32 46 43 43 35 49 32 49 49 33  0 35 33 31 25 47
 43 40 50 37 41 28 48 41 43 25 30 37 31 40 41  0 37 49 48 41
 28 36 30 43 43 29 25 36 33 25 43 50 33 36 29 49  0 46 45 44
 46 36 31 46 35 42 30 43 40 36 48 39 32 39 32 41 50  0 43 43
 49 34 35 42 28 33 25 43 43 31 47 42 42 31 27 37 39 39  0 42
 38 37 44 36 49 32 41 50 35 28 50 38 47 44 43 37 37 27 43  0
M1

M2
```

Figura 3.2. Archivo de datos de entrada para un problema de TFH2

La Figura 3.3 muestra la distribución de los parámetros del sistema de recursos y tiempos de procesamiento de trabajos en el archivo de entrada. Debido a que las maquinas en el modelo son no relacionadas, los tiempos de procesamiento de un trabajo en la etapa se distinguen en dependencia de la máquina.

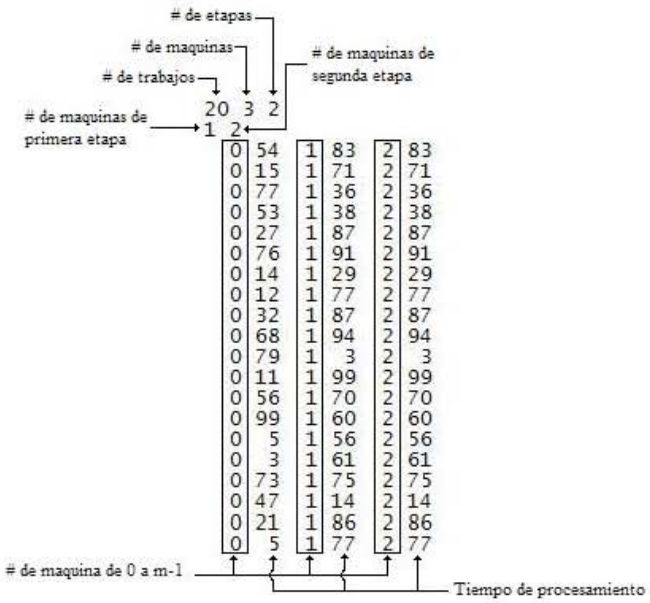

Figura 3.3. Parámetros de la primera sección del archivo de datos de entrada

El primer renglón del archivo de texto contiene tres datos:

- el número de trabajos del experimento,
- el número de máquinas disponibles,
- el número de etapas en el modelo.

El segundo renglón contiene una lista de números que indican la cantidad de máquinas por etapa. En la Figura 3.3 aparecen los datos para un problema $FH2,(RM^{(i)})_{i=1}^{2} \mid\mid C_{\max}$ con 20 trabajos, 3 máquinas y 2 etapas: para la primera etapa se utiliza una máquina y para la segunda etapa 2 máquinas, en total 3 máquinas en el primer renglón.

Después siguen 20 renglones con datos, uno por cada trabajo a realizar. Para el ejemplo de la Figura 3.3, cada renglón cuenta con seis columnas, las cuales aparecen como tres pares de datos. El primer dato en cada par es el número de máquina, el cual comienza a partir de cero (máquina 1), y el segundo es el tiempo de procesamiento del trabajo en dicha máquina.

En caso de que un trabajo no se procese en alguna máquina, este valor se pone en "-1" para indicarle al sistema que la máquina no está disponible para el trabajo.

Las máquinas (y sus tiempos de procesamiento) se agrupan de acuerdo al número de máquinas por etapa (Figura 3.4).

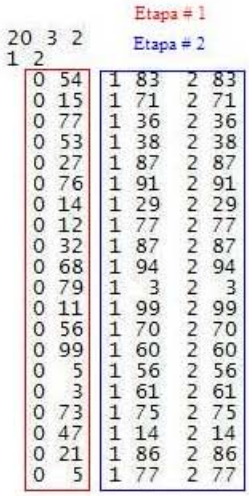

Figura 3.4. Dos grupos de máquinas de acuerdo al número de etapas

La segunda sección del archivo de entrada proporciona los tiempos de ajuste (*Setup Times*) de las máquinas dependientes de la secuencia de los trabajos. Esta sección está formada por matrices cuadradas para cada una de las máquinas del modelo, como se aprecia en la Figura 3.5[3]. El tamaño de la matriz corresponde al número de trabajos a ejecutar. Para el ejemplo se tienen 20 trabajos y 3 máquinas, por lo tanto se requiere de 3 matrices de tamaño 20x20.

En la matriz el índice del renglón corresponde al número de trabajo anterior y el índice de columna es número del trabajo actual. Por ejemplo, si se requiere introducir el trabajo número 8 en la máquina 1 (en este caso M0) y además anteriormente se procesaba el trabajo número 4, entonces el tiempo de ajuste sería de 42 unidades de tiempo (Figura 3.6). La diagonal principal de la matriz es cero ya que es el mismo trabajo.

Los archivos de configuración de PLARETF son: *config.ini* y *listacarga.ini*.

El archivo *config.ini* describe la configuración del sistema computacional en el siguiente orden (ANEXO B):

- el número de experimentos a realizar,
- el modo de ejecución del sistema (*Debug/Run*),

[3] Si el problema a resolver no utiliza tiempos de ajuste, la matriz debe llenarse con ceros.

- la selección de los algoritmos.

El archivo *listacarga.ini* contiene la lista de los nombres de archivos de datos del problema a ejecutar (ANEXO C).

La Figura 3.7 muestra la estructura de directorios del sistema.

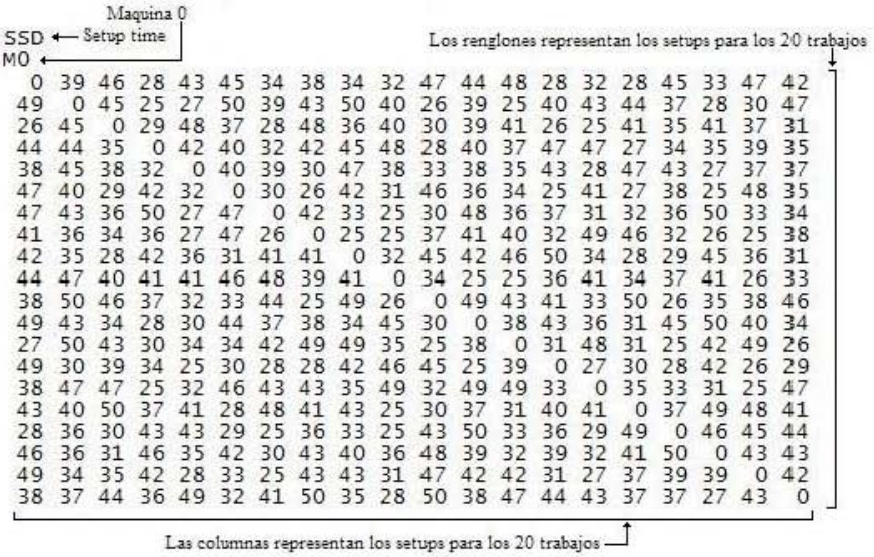

Figura 3.5. Parámetros de la segunda sección del archivo de datos de entrada

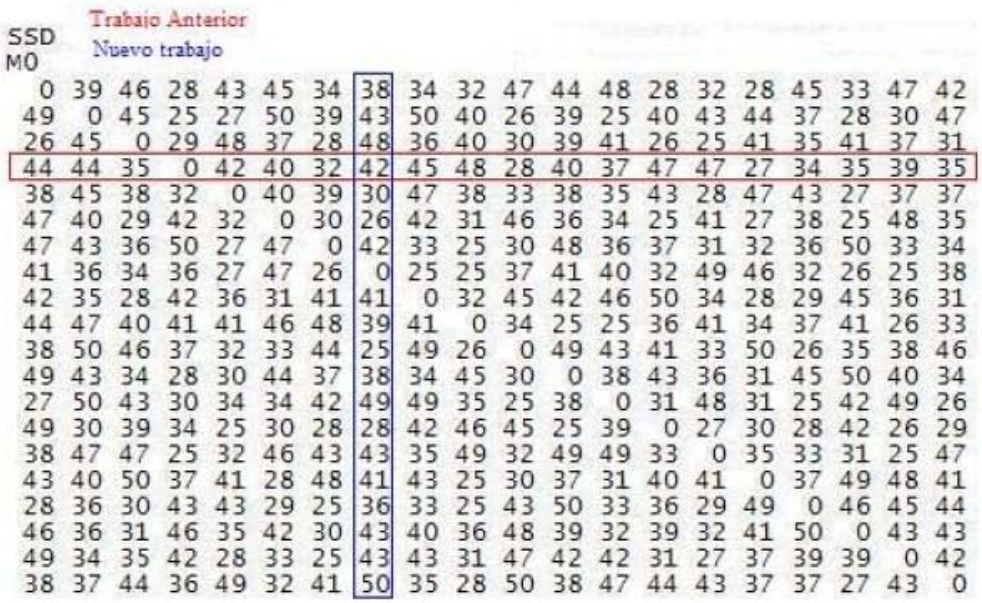

Figura 3.6. Tiempo de ajuste de la máquina M0 para el trabajo 8 después del 4

Figura 3.7. Estructura de directorios del sistema PLARETF

En la Tabla 3.2 se describe el contenido de la carpeta principal *PLARETF*.

La carpeta *LOGS* contiene los archivos de datos de problemas para experimentos (Figura 3.8).

En la capeta *Experiments* se depositan los resultados obtenidos (la salida) por el sistema (Figura 3.9).

Tabla 3.2. Contenido de la carpeta principal del sistema PLARETF

Nombre del archivo	Descripción
PLARETF.exe	Archivo ejecutable del sistema de simulación.
config.ini *listacarga.ini*	Archivos de configuración del sistema.
GA_HFS_SDST.ini *M_plus_m.ini* *uno_plus_m.ini*	Archivos de configuración de los algoritmos
path_local_directory.txt	Archivo de configuración para la salida del sistema.
Experiments	Carpeta para los archivos de salida del sistema.
LOGS	Carpeta con archivos de carga para el sistema.

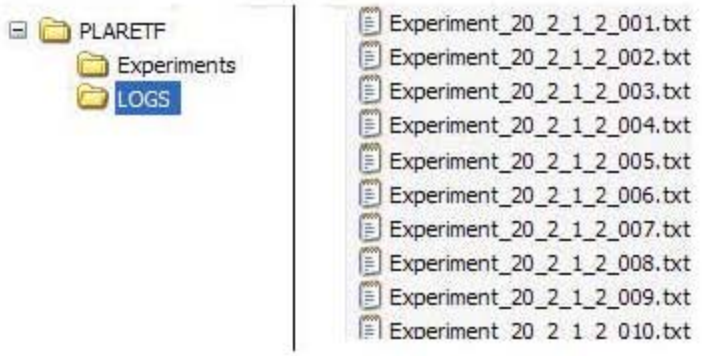

Figura 3.8. La carpeta *LOGS*

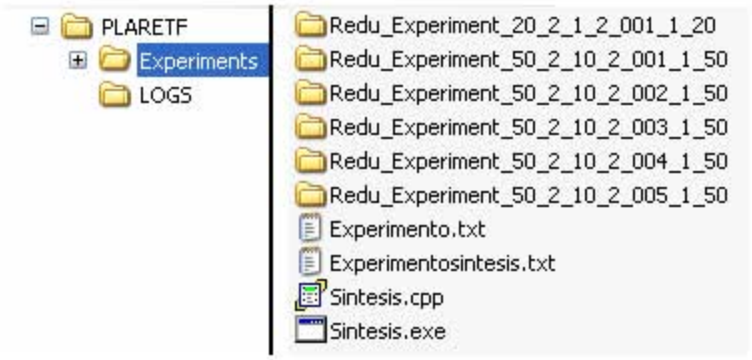

Figura 3.9. La capeta *Experiments*

3.4 Ordenación

La sección central del esquema del sistema PLARETF llamada ORDENACIÓN (Figura 3.1) está formada por una colección de algoritmos diseñados para resolver varios tipos de problemas de Taller de Flujo. El sistema PLARETF está diseñado de tal manera que facilita la implementación de nuevos algoritmos sin afectar la entrada y la salida.

Actualmente la sección de ordenación contiene dos algoritmos que resuelven problemas:

$FF2,1^{(1)},RM^{(2)} \mid\mid C_{\max}$,

$FF2,RM^{(1)},RM^{(2)} \mid\mid C_{\max}$,

$FF2,RM^{(1)},1^{(2)} \mid\mid C_{\max}$.

Para solucionar un problema, un algoritmo debe realizar los pasos mostrados en la Figura 3.10.

Entrada (*Input*): El algoritmo realiza una copia de la entrada genérica de datos del sistema PLARETF y la ajusta a sus propias estructuras. El sistema utiliza la misma entrada de datos para distintos algoritmos, siempre y cuando los algoritmos sean compatibles con el problema.

Figura 3.10. Esquema general de pasos de un algoritmo en el sistema

Asignación (*Assignment*): En el caso general, los trabajos no especifican las máquinas en que deben ejecutarse. El algoritmo debe asignar los trabajos a las máquinas. En el caso de sistemas de máquinas con dos etapas, se obtiene como resultado la partición del conjunto J de trabajos:

1. m_1 disjuntos $J_{1,i}$ $i=\overline{1,m_1}$, $J=\bigcup_{i=1}^{m_1} J_{1,i}$, $J_{1,i}\neq\varnothing$ $J_{1,\mu}\cap J_{1,\nu}=\varnothing$, $\mu\neq\nu$,
2. m_2 disjuntos $J_{2,i}$ $i=\overline{1,m_2}$, $J=\bigcup_{i=1}^{m_2} J_{2,i}$, $J_{2,i}\neq\varnothing$ $J_{2,\mu}\cap J_{2,\nu}=\varnothing$, $\mu\neq\nu$.

Los algoritmos de asignación implementados en el sistema son:

SPT	S*hortest Processing Time first* (en primera etapa) & *minload,*
SPTB	*Shortest Processing Time first B* (en segunda etapa) & *minload,*
LPT	*Longest Processing Time first* (en primera etapa) & *minload,*
LPTB	*Longest Processing Time first B* (en segunda etapa) & *minload,*
JohnR	*Johnson Rule & minload,*
GEN	Asignación por métodos evolutivos (Genético).

El resultado de los algoritmos de asignación se deposita en un arreglo genérico de tres dimensiones, el cual representa un conjunto de colas locales de las máquinas (Figura 3.11).

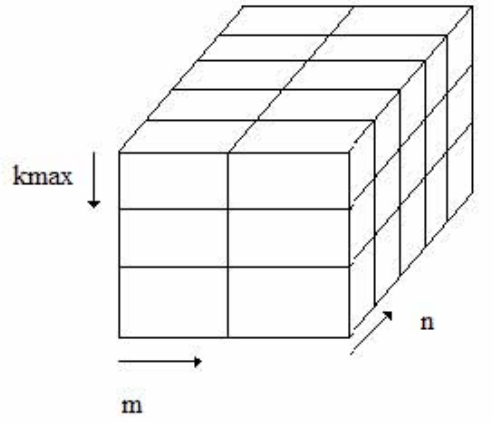

Figura 3.11. Representación del arreglo de colas locales en las máquinas [*kmax*][*m*][*n*]

Para crear el arreglo se toman en cuenta tres datos, uno por cada dimensión:

- el número máximo de máquinas para todas las etapas *k*max, donde $k\max = \max(m_1, m_2, ..., m_m)$,
- el número de etapas del modelo m,
- el número de trabajos a procesar en el modelo n.

La idea principal del arreglo es tener una representación total para cualquier modelo tomando en cuenta sus parámetros principales: número de trabajos, número de etapas y cantidad de máquinas, etc. (Figura 3.12).

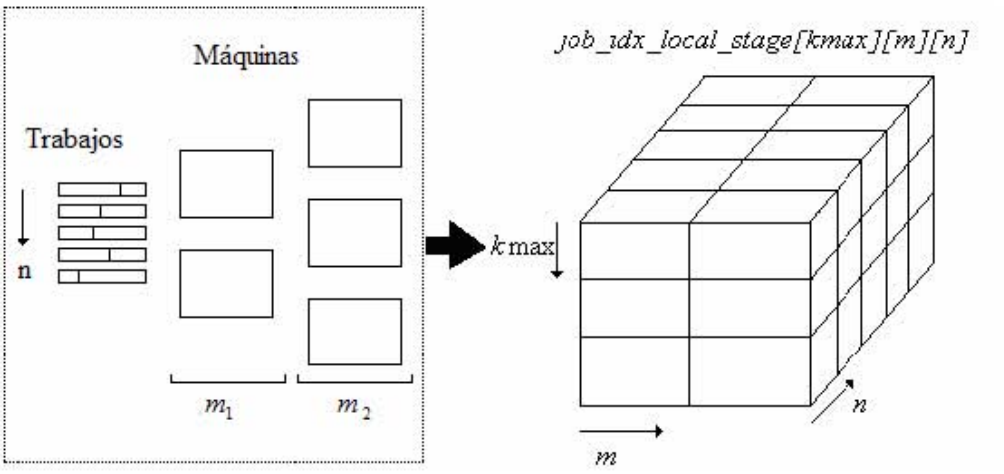

Figura 3.12. Representación de la asignación de los trabajos a las máquinas

Implementación de las colas locales en el sistema:

```
//asignación de memoria dinámica
//kmax  es el número máximo de máquinas entre todas las etapas
//n es el número de trabajos a ejecutar
//m es el número de etapas
job_idx_local_queues_stage = (long***)malloc(kmax*sizeof(long**));
        for (int j = 0; j < kmax; j++)
                job_idx_local_queues_stage[j] = (long**)malloc(m*sizeof(long*));
        for (int i = 0; i < kmax; i++)
                for (int j = 0; j < m; j++)
                        job_idx_local_queues_stage[i][j]= (long*)malloc(n*sizeof(long));
```

Secuenciación de trabajos (*Sequencing*): En esta sección los algoritmos toman los datos de entrada y la asignación para encontrar una secuencia (permutación), que minimice el valor de $C_{\max}$. Al concluir la ejecución del algoritmo, los datos generados se pasan a la sección de Programación.

Programación (*Timing*)**:** En esta sección el algoritmo toma la secuencia de trabajos generada y realiza una simulación de la ejecución de los trabajos en las máquinas para calcular la información de salida del experimento. De este procedimiento se obtienen los resultados para los valores mostrados en la Tabla 3.3.

Tabla 3.3. Información de salida del sistema

Resultado generado	Descripción
cmax	Tiempo de ejecución de los trabajos en las máquinas del modelo.
cmax_flow	El tiempo de ejecución de los trabajos por flujo
cmax_stage	Tiempo de ejecución de los trabajos por cada etapa del modelo.
estimated_time	Tiempo de estimación de ejecución
idle	Tiempo ocioso del sistema.
Idle_flow	Tiempo ocioso del flujo
idle_stage	Tiempo ocioso por etapa
Performance_ratio	Proporción de funcionamiento del sistema

ratio_stage	Proporción del funcionamiento por etapa
response_time	Tiempo de respuesta de las máquinas
response_time_flow	Tiempo de respuesta del flujo
response_time_stage	Tiempo de respuesta de la etapa
throughtput	Rendimiento
throughtput_flow	Rendimiento del flujo
throughtput_stage	Rendimiento de la etapa
utilization	Utilización de las máquinas
utilizations_flow	Utilización del flojo
utilization_stage	Utilización de la etapa
waiting_time	Tiempo de espera de las máquinas
waiting_time_flow	Tiempo de espera en el flujo
waiting_time_stage	Tiempo de espera en la etapa
work	Trabajo en las máquinas
work_flow	Trabajo en el flujo
work_stage	Trabajo en la etapa

Salida (*Output*)**:** Los resultados de la Programación se procesan y se formatean para entregarlos a la sección de salida genérica del sistema computacional. En el ANEXO D se muestra la implementación de estas estructuras.

3.5 Salida

La tercera y última sección del modelo del sistema computacional se encarga de procesar los resultados de cada algoritmo para generar archivos de salida.

El resultado obtenido por cada algoritmo se clasifica en un sistema de carpetas y archivos de texto que se deposita en la carpeta *Experiments*, dentro de la cual se crean carpetas, una por cada archivo de carga utilizado. Estas carpetas se nombran tomando como base el nombre del archivo de carga utilizado para el experimento (Figura 3.13).

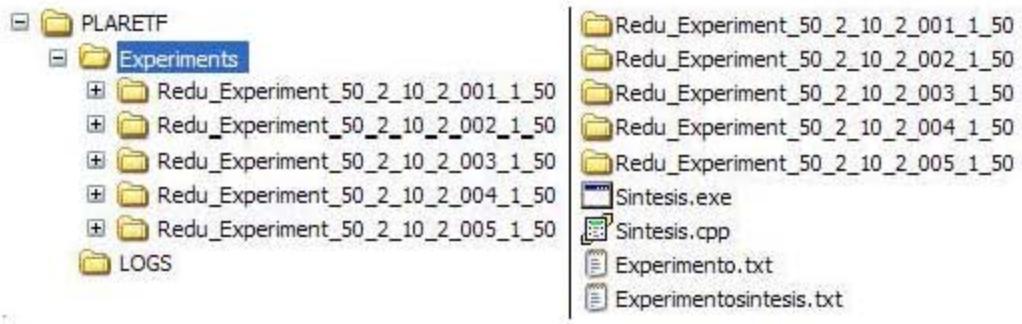

Figura 3.13. Carpetas y archivos generados en el experimento

Cada carpeta ubicada dentro de *Experiments* contiene a su vez una carpeta por cada algoritmo utilizado. Las carpetas de este nivel reciben el nombre del algoritmo al cual pertenecen (Figura 3.14).

Junto con las capetas de los algoritmos se colocan los archivos de configuración y un archivo de texto, el cual contiene el historial de los resultados obtenido correspondiente al archivo de carga. Un ejemplo del archivo que contiene el historial se muestra en el ANEXO E. El nombre del archivo se forma a partir del prefijo "historial" y concatenándolo con nombre del archivo de carga de datos utilizado.

El archivo historial contiene:

- el nombre del archivo;
- los parámetros del problema, como el número de trabajos, número de máquinas, número de máquinas por etapa y número de etapas;
- los resultados de *C*max y el Limite inferior (*Lower Bound, LB*) para cada algoritmo;
- las colas locales que representan la asignación y la secuencia de los trabajos en las máquinas;
- el tiempo de ejecución para cada algoritmo;
- los valores de Cmax;
- el resultado normalizado e incrementado del *C*max en unidades relativas para el experimento.

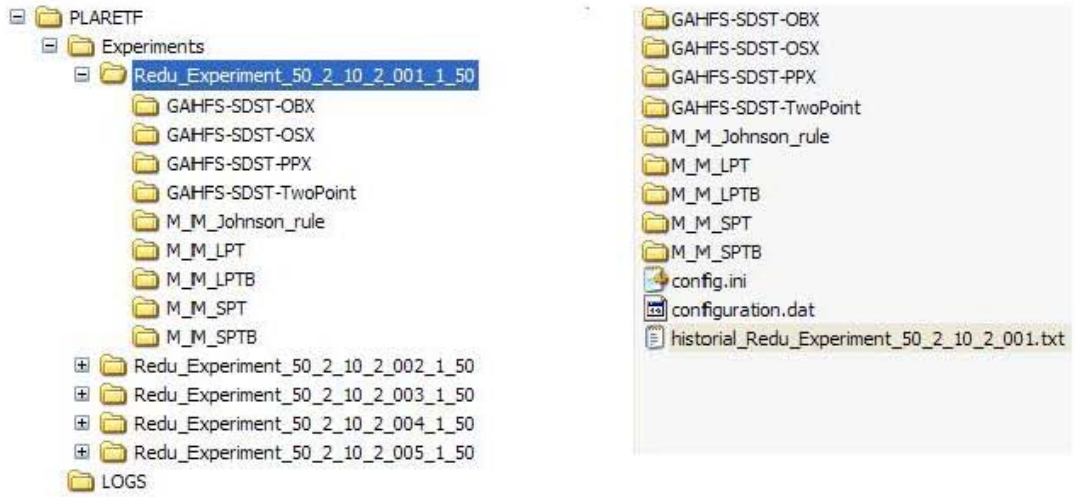

Figura 3.14. Contenido de una subcarpeta de *Experiments*

Para normalizar el *C*max se toma en cuenta el mejor resultado obtenido por el experimento, es decir, el mejor valor conocido de *C*max:

$$C_{max}\ \text{Normalizado} = \left(\frac{C_{max} - Best}{C_{max}}\right). \tag{3.1}$$

En los ANEXO F y ANEXO G se muestran respectivamente los archivos de C_{max} Normalizado para los algoritmos M+m y GABC-1.

El incremento de *C*max se calcula según la fórmula:

$$Inc_{Cmax} = \left(\frac{C_{max} - Best}{Best}\right) \times 100\% \tag{3.2}$$

En los ANEXO H y ANEXO I se muestran respectivamente los archivos de Inc_{Cmax} para los algoritmos M+m y GABC-1.

La carpeta de cada algoritmo incluye un archivo de texto llamado *Global_Data_Result.txt*, el cual contiene los datos de salida descritos en la Tabla 3.3. En el ANEXO J se muestra un ejemplo del archivo *Global_Data_Result.txt*.

En la carpeta *Experiments* se deposita un archivo, el cual contiene los valores normalizados de *Cmax* parea todos los archivos de carga procesados. Este archivo tiene el nombre de *Experimento.txt* (Figura 3.15).

```
Exp Redu_Experiment_50_2_10_2_001.txt
SPT     SPTB    LPT     LPTB    JohnR   OBX
9.934   0.662   0.000   9.934   12.583  8.609
Exp Redu_Experiment_50_2_10_2_002.txt
SPT     SPTB    LPT     LPTB    JohnR   OBX
4.698   4.698   0.000   4.027   4.698   9.396
Exp Redu_Experiment_50_2_10_2_003.txt
SPT     SPTB    LPT     LPTB    JohnR   OBX
2.837   12.766  9.220   3.546   2.837   0.000
Exp Redu_Experiment_50_2_10_2_004.txt
SPT     SPTB    LPT     LPTB    JohnR   TP
9.396   3.356   0.000   8.054   12.752  7.383
Exp Redu_Experiment_50_2_10_2_005.txt
SPT     SPTB    LPT     LPTB    JohnR   OBX
6.164   13.014  1.370   0.000   14.384  10.959
```

Figura 3.15. Contenido de un archivo *Experimento.txt*

El archivo *Expermiento.txt* muestra los resultados obtenidos por cada algoritmo para diferentes archivos de carga. El valor resultante es obtenido por medio de la ecuación de incremento (3.2). Un valor cero indica que el algoritmo obtuvo el mejor resultado para el archivo de carga específico.

3.6 Conclusión del capítulo

Los investigadores utilizan diferentes metodologías para evaluar algoritmos, puesto que estos poseen características propias, aun cuando se busca resolver un problema similar. Esto trae como consecuencia la dificultad de comparar la complejidad y exactitud de los algoritmos.

La práctica común es evaluar los algoritmos por medio de un experimento. Un sistema computacional diseñado para la ejecución de distintos algoritmos representa una herramienta útil para analizar su comportamiento.

El sistema computacional PLARETF, permite evaluar algoritmos para resolución de problemas de TFH en general. Esto es posible debido a la entrada genérica de datos que contempla distintos parámetros de un TFH. La estructura del sistema computacional permite implementar cualquier algoritmo independientemente de la heurística o metaheurística que este utiliza. Los algoritmos se agregan en forma autónoma, esto es, a

manera de módulos que deben respetar los estándares de entrada y salida del sistema. Además el sistema permite que un mismo problema sea resuelto con varios algoritmos de forma simultánea. Por ultimo, el sistema entrega como resultado la información para evaluar la exactitud y complejidad de los algoritmos aplicados a un problema.

4 EVALUACIÓN DE ALGORITMOS EN EL SISTEMA COMPUTACIONAL PLARETF

4.1 Algoritmos evaluados

En el experimento computacional se evalúan dos algoritmos para minimización de la fecha máxima en un TFH2 con múltiples máquinas en ambas etapas. El primero es un algoritmo heurístico M+m descrito en la Sección 2.5. Está diseñado para resolver un caso general de TFH2, con múltiples máquinas en cada etapa, sin tiempo de ajuste de la máquina entre trabajos. El algoritmo se basa en el método de dicotomía, el cual le permite encontrar el resultado en un tiempo corto. El segundo es un algoritmo evolutivo del tipo genético denominado GABC-1. Este algoritmo está diseñado para un caso general de TFH2, con máquinas de distintas velocidades, con tiempos de ajuste entre trabajo.

4.1.1 Implementación del algoritmo heurístico M+m

El algoritmo M+m requiere en su entrada una asignación predeterminada de los trabajos a las máquinas, lo que corresponde a una descomposición del conjunto J de n trabajos en m_1 disjuntos para la primera etapa y m_2 disjuntos para la segunda. Para realizar la asignación se implementan cinco algoritmos: dos variantes de SPT (*Shortest Process Time*), dos variantes de LPT (*Largest Process Time*) y una adaptación de la regla de Johnson combinado con MinLoad (Mínima Carga).

El algoritmo M+m es iterativo. En cada iteración construye una solución que corresponde a un límite de la búsqueda dicotómica, si esta existe. El algoritmo actúa en dos direcciones: Primero, intenta construir la solución en la dirección directa, la cual toma

como referencia la primera etapa del modelo para mejorar las colas locales de las máquinas en la primera etapa. En caso de obtener una solución valida, el algoritmo regresa m_1 permutaciones de trabajos, es decir, una permutación por máquina de la primera etapa. En caso contrario el algoritmo actúa en la dirección opuesta buscando la solución valida para la segunda etapa en forma de m_2 permutaciones.

Como la solución se busca entre las permutaciones, los trabajos conservan el orden mutuo de precedencia. Entonces, las m_1 permutaciones de la primera etapa de manera unívoca determinan m_2 permutaciones de la segunda etapa, y viceversa: una permutación de n trabajos de manera unívoca determina el orden de procesamiento en todas las máquinas.

Después de obtener las permutaciones de trabajos en cada máquina se aplica el algoritmo de fusión para entregar como resultado una permutación única, lo que corresponde a las estructuras de datos de salida del sistema computacional.

A continuación se presenta el algoritmo FUSIÓN, desarrollado e implementado para unir las m_1 o m_2 permutaciones entregadas por el algoritmo M+m, en una única permutación.

Se crea una única permutación, la cual se pone en equivalencia a las m_1 (ó m_2) permutaciones obtenidas por el algoritmo M+m. Sin perder la generalidad, el algoritmo se diseña para realizar la fusión de m_1 permutaciones de la primera etapa.

El algoritmo recibe como entrada las siguientes constantes:

- $m, m_1, m_2, n, J_j,\ j = \overline{1,n}$.
- El número de elementos por permutación $|n_i|,\ i = \overline{1,m_1}$, tal que $\sum_{i=1}^{m_1} n_i = n$.
- El vector dinámico $P[n]$ compuesto por m_1 permutaciones de trabajos concatenadas en el orden natural de las máquinas de la primera etapa:

  $P = \{j_{1,1},\ldots,j_{1,n_1}, j_{2,1},\ldots j_{2,n_2}, j_{m_1,1},\ldots,j_{m_1,n_{m_1}}\}$,

donde $\{j_{1,1},\ldots,j_{1,n_1}\}$ es la permutación de trabajos en la máquina 1 de la primer etapa,…, $\{j_{m_1,1},\ldots,j_{m_1,n_{m_1}}\}$ es la permutación de la máquina m_1 de la primer etapa.

Es necesario poner los n trabajos del vector P en orden, el cual unívocamente define la secuencia de los trabajos en cada una de las m_1 máquinas.

En el primer paso del algoritmo se crea un vector dinámico complementario ST (*Start Time*) con la misma estructura del vector $P[n]$. El vector $ST[n]$ representa la lista de los instantes de inicio de los trabajos a ejecutar (*job ready instant*). Sus elementos se calculan de acuerdo a la posición en el vector $P[n]$ y tiempos de procesamiento en la primera etapa.

En el segundo paso se crea el vector dinámico resultante $Pf[n]$, con el mismo tamaño del vector $P[n]$, pero con distinta estructura lógica. Los elementos del vector $P[n]$ son depositados en el vector $Pf[n]$ según el orden de no crecimiento de los elementos en el vector $ST[n]$. El vector $Pf[n]$ es la permutación única, la cual equivale a m_1 permutaciones de trabajos en las máquinas.

Inicialmente el *ready instant* de todas las máquinas es 0.

El algoritmo FUSIÓN de las m_1 permutaciones contiene los pasos siguientes:

Paso 1: Calculo de *start times*

$P = \{j_{1,1},\ldots, j_{1,n_1}, j_{2,1},\ldots j_{2,n_2}, j_{m_1,1},\ldots, j_{m_1,n_{m_1}}\}$

FOR $iter = 1$ TO n

$ST[iter] = 0;$

FOR $mach = 1$ TO m_1

IF $mach \neq 1$

$$machstep = \sum_{i=1}^{mach-1} n_i$$

ELSE

$machstep = 0;$

FOR $iter = 1$ TO m_1

$ST[iter + machstep] = ST[iter - 1 + machstep] + j[P[iter + machstep]].p^1$

Paso 2: Creación de la permutación resultante

WHILE ST[k]!=-1, $\forall k = \overline{1,n}$

FOR $mach = 1$ TO m_1 {

minimo = $\max_{1 \le iter \le n} (ST[iter])$;)

IF $mach = 1$

$machstep = 0;$

ELSE

$machstep = \sum_{i=1}^{mach} n_i;$

FOR $iter = 1$ TO n_i

IF ST[iter+machstep]<minimo && ST[iter+machstep]!=-1

$job = P[iter + machstep]$

minmach=mach;

match=machstep+iter;

% Colocación del número de trabajo en la posición l, $l = \overline{1,n}$. %

$Pf[l] = job;$

$ST[match] = -1;$

$l = l + 1;$

Si ST[k]=-1, $\forall k = \overline{1,n}$, la permutación final está formada.

4.1.2 Algoritmo genético GABC-1

El algoritmo genético GABC-1 se desarrolló para resolver el problema de planificación de trabajos en la sección de Auto-Inserción de una empresa de manufactura de televisiones. El algoritmo genético resuelve problemas de 2, 3 y 6 etapas (Yaurima, et al., 2007).

El trabajo se asigna a la máquina que lo finaliza más temprano en la etapa dada, tomando en cuenta distintas velocidades de procesamiento, tiempos de cambio de partida dependientes del trabajo, búferes limitados y la disponibilidad de las máquinas. Para la selección se usa el esquema conocido como el torneo (*tournament*) estocástico binario (Mijalevich, 1996).

Se usan cuatro tipos de cruzamiento: OBX (*Order-Based Crossover*), PPX (*Precedence Preservation Crossover*), OSX (*One Segment Crossover*), TX (*Two Point Crossover*) con 6 niveles de probabilidad de cruzamiento: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5; 5

niveles de probabilidad de mutación: 0.0, 0.005, 0.01, 0.015, 0.02 y 2 niveles de criterios generacionales: Fijo y Repeticiones del mejor individuo.

La configuración del algoritmo genético se adapta para resolver problemas con dos etapas, sin restricciones adicionales, específicamente sin tiempos de cambio de partida, lo cual hace comparables sus resultados con los del algoritmo M+m.

4.2 Datos de entrada

Uno de los problemas principales al realizar un experimento computacional es elegir los tiempos de procesamiento para los trabajos, los cuales permitan la extracción de resultados válidos. Para resolución de problemas en Talleres de Flujo se utiliza el estándar de datos desarrollado por Éric Taillard (1993), cuyos valores se obtienen a través de un generador lineal congruente de números pseudoaleatorios. El estándar de datos de Taillard se basa en la aritmética modular, donde el valor $(n+1)$ se calcula a partir de n.

A continuación se presenta el Generador de la Semilla para los tiempos de procesamiento de los trabajos (Taillard, 1993):

1. Semilla inicial y constantes $X_0 (0 < X_0 < 2^{31} - 1)$

   $a = 16807$, $b = 127773$,

   $c = 2836$, $m = 2^{31} - 1$
2. Modificación de la semilla $k := \lfloor X_i / b \rfloor$

   $X_{i+1} := a(X_i \bmod b) - kc$

   If $X_{i+1} < 0$ then $X_{i+1} := X_{i+1} + m$
3. Nuevo valor de la semilla X_{i+1}
4. Valor actual del generador. X_{i+1} / m

Aquí a, b, c son los parámetros, m es el valor máximo de los números enteros de 32 bits (*Long Integer*).

Sea $U(0,1)$ el número pseudoaleatorio producido por el generador, $0 < U(0,1) < 1$. Para un intervalo $[a,b]$, $a < b$, se obtiene un número entero $U[a,b]$ en el rango $[a,b]$ de acuerdo a la formula:

$U[a,b] = \lfloor a + U(0,1) \cdot (b - a + 1) \rfloor$.

En el generador se utiliza la distribución uniforme discreta, por lo tanto cada entero positivo en el rango $[a,b]$ tiene la misma probabilidad de ser seleccionado.

Para automatizar el experimento, se crea un generador de Archivos de Carga para el sistema computacional, tomando en cuenta el estándar de datos de Taillard y los parámetros del problema, en total 960 archivos, agrupados de acuerdo a los modelos de recursos y número de trabajos. Un ejemplo del Archivo de Carga se muestra en el ANEXO A.

4.3 Calibración del algoritmo M+m

En el análisis previo se descubrieron tres parámetros capaces de afectar al $C_{\max}$: número de trabajos, modelo de recursos y algoritmo de asignación utilizado. Los dos primeros, en realidad, son parámetros internos del experimento, mientras que el tercero es por elegir. Además, para un caso real el número de trabajos y modelo son datos de entrada.

En el experimento se estudia, como afecta el algoritmo de asignación a $C_{\max}$, con el propósito de seleccionar la mejor asignación entre cinco heurísticas SPT, SPTB, LPT, LPTB, JohnR y de tal manera calibrar el algoritmo M+m con respecto al algoritmo de asignación. Por lo tanto, el experimento es unifactorial. El algoritmo de asignación es considerado como un factor, el cual representa la variable independiente que interviene en el experimento y que ejerce determinado efecto sobre la variable dependiente, la cual es $C_{\max}$. En el diseño del experimento el factor es considerado como un tratamiento.

El procedimiento de calibración se realiza en el siguiente orden:

1. Diseño del experimento: elección de parámetros para el factor seleccionado, definición de la hipótesis nula e intervalos de confianza, etc.
2. Revisión de la adecuación del modelo mediante el análisis residual.
3. Aceptación o rechazo de la hipótesis nula a través del análisis de varianza (ANOVA) y con esta base calibrar el algoritmo.

4.3.1 Diseño de experimento

Se desea probar la hipótesis sobre las medias de los tratamientos y estimar los efectos del tratamiento con el propósito de escoger la mejor asignación. Las condiciones no

extienden a tratamientos similares que no fueron considerados, es decir a otros algoritmos de asignación. Esto se conoce como modelo de efectos fijos.

Para evaluar el comportamiento del algoritmo M+m con $a = 5$ niveles del tratamiento, se utilizan doce modelos de recursos y cuatro valores de n (número de trabajo). Para cada combinación de los parámetros se realizan 20 observaciones. El número total de observaciones se calcula a partir de la cardinalidad de los conjuntos *MODELO* y números de trabajos N.

El conjunto *MODELO* describe el sistema de recursos para dos grupos de máquinas. Los elementos del conjunto están formados por pares ordenados, cuyo primer elemento es el número de máquinas en el primer grupo y el segundo elemento es el número de máquinas en el segundo grupo.

$$MODELO = \{(1,2), (1,3), (1,5), (2,1), (2,3), (2,5), (2,10), (3,1), (3,2), (5,1), (5,2), (10,2)\}.$$

Para especificar el modelo se usa la notación M(I, i), donde I es el número de máquina de la primera etapa e i es el número de máquina de la segunda etapa.

El conjunto N contiene 4 valores de números de trabajos a ejecutar

$$N = \{20, 50, 100, 200\}.$$

De tal manera, se utilizan 48 distintos diseños de experimentos, los cuales se forman por las combinaciones de 4 valores de N y 12 valores de *MODELO*.

Para cada diseño se toman $k = 20$ observaciones, en total $K = 48 \times 20 = 960$ resultados (ANEXO K y ANEXO L).

El análisis de varianza se realiza para un diseño, puesto que lo requiere la condición de homogeneidad del ambiente del experimento. En particular, el diseño utilizado corresponde a 20 trabajos y modelo de recursos M(2, 3) con dos máquinas en la primera etapa y tres máquinas en la segunda. A continuación se describe el análisis estadístico para el diseño seleccionado.

4.3.2 Modelo estadístico

Los datos para el análisis se presentan según el formato de la Tabla 4.1 (Montgomery, 1991).

Tabla 4.1. Representación de datos para el análisis estadístico, $k = 20$, $a = 5$

Tratamiento	**Observaciones**						**Totales**	**Medias**
	1	**2**	**...**	***j***	**...**	***k***		
1	y_{11}	y_{12}	···	y_{1j}	···	y_{1n}	$y_{1\cdot}$	$\bar{y}_{1\cdot}$
2	y_{21}	y_{22}	···	y_{2j}	···	y_{2n}	$y_{2\cdot}$	$\bar{y}_{2\cdot}$
...	...	...	...	...	...	...	...	...
i	y_{i1}	y_{i2}	···	y_{ij}	···	y_{in}	$y_{i\cdot}$	$\bar{y}_{i\cdot}$
...	...	...	...	...	...	...	...	...
a	y_{a1}	y_{a2}	···	y_{aj}	···	y_{an}	$y_{a\cdot}$	$\bar{y}_{a\cdot}$
							$y_{\cdot\cdot}$	$\bar{y}_{\cdot\cdot}$

En la tabla:

$$y_{i\cdot} = \sum_{j=1}^{k} y_{ij},$$

$$\bar{y}_{i\cdot} = y_{i\cdot} / k,$$

$$y_{\cdot\cdot} = \sum_{i=1}^{a} \sum_{j=1}^{k} y_{ij},$$

$$\bar{y}_{\cdot\cdot} = y_{\cdot\cdot} / K,$$

$i = \overline{1, a}$, $j = \overline{1, k}$, $K = ak$ es el número total de observaciones.

El modelo estadístico lineal es:

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij} \begin{cases} i = 1, 2, ..., a \\ j = 1, 2, ..., k \end{cases}, \tag{4.1}$$

o:

$$Y_{ij} = \mu_i + \varepsilon_{ij} \begin{cases} i = 1, 2, ..., a \\ j = 1, 2, ..., k \end{cases}, \quad \text{donde } \mu_i = \mu + \tau_i$$

Aquí Y_{ij} es una variable aleatoria que denota la observación (i, j); μ es una media global, un parámetro común para todos los tratamientos; τ_i es efecto del tratamiento *i*; ε_{ij} es un componente del error aleatorio; μ_i es la media del tratamiento *i*.

Entonces cada tratamiento define una población con una media μ_i, que consiste de la media global μ más un efecto del tratamiento τ_i. Se supone que los errores ε_{ij} están distribuidos de manera normal e independiente, $N(0,\sigma^2)$ (con la media 0 y varianza σ^2).

El interés es probar como hipótesis nula la igualdad de las medias

$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5$.

La hipótesis alternativa es

H_1 : al menos una μ_i, $i = \overline{1,5}$, es diferente de las demás.

Si la H_0 es verdadera, entonces $Y_{ij} = \mu + \varepsilon_{ij}$, todas las K observaciones se toman de una distribución normal, $N(\mu,\sigma^2)$ y el cambio en los niveles del factor no tiene efecto directo sobre la respuesta promedio.

La hipótesis nula H_0 se prueba con el nivel de confianza $100(1-\alpha) = 95\%$, donde $\alpha = 0.05$.

En la Tabla 4.2 se presentan los valores de $C_{\max}$ obtenidos en el experimento Redu_Experiment_20_2_2_3, es decir para $n = 20$ trabajos, y modelo de recursos M(2, 3).

Tabla 4.2. Resultados del experimento Redu_Experiment_20_2_2_3

Asignación	Observaciones									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
SPT	256	219	225	216	198	171	221	192	199	221
SPTB	307	185	180	221	242	172	250	209	199	228
LPT	272	200	203	225	219	187	189	186	192	295
LPTB	244	202	179	220	195	209	234	197	199	283
JohnR	242	196	194	211	194	174	190	184	192	193

Continuación de Tabla 4.2

Asignación	Observaciones										Totales	Medias
	11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**		
SPT	228	227	208	175	181	168	233	179	173	227	4117	205.85
SPTB	207	215	196	172	183	186	259	201	169	242	4223	211.15
LPT	202	235	268	184	164	167	212	197	168	211	4176	208.8
LPTB	229	234	224	174	207	190	237	190	171	249	4267	213.35
JohnR	210	239	200	166	169	178	217	183	167	216	3915	195.75

4.3.3 Análisis de residual

Se supone que los valores recibidos de $C_{\max}$ tienen la distribución normal, con la misma varianza para cada nivel de factor. Estas suposiciones se verifican mediante el examen de los residuos. Un residuo es la diferencia entre una observación y_{ij} y su valor estimado obtenido a partir del modelo estadístico bajo estudio.

Los residuos se calculan según la formula $e_{ij} = y_{ij} - \bar{y}_i$, donde $\bar{y}_i$ es el estimador de la media del tratamiento. El uso de $\bar{y}_i$ (valor estimado) para calcular cada residuo elimina en esencia el efecto del factor (efecto del algoritmo de asignación a $C_{\max}$). Los residuos contienen la información sobre la variabilidad no explicada. En la Tabla 4.3 se presentan los residuos del experimento Redu_Experiment_20_2_2_3.

Tabla 4.3. Residuos del experimento Redu_Experiment_20_2_2_3

Asignación	Observaciones									
	1	2	3	4	5	6	7	8	9	10
SPT	50.2	13.2	19.2	10.2	-7.9	-34.9	15.2	-13.9	-6.9	15.2
SPTB	95.9	-26.2	-31.2	9.9	30.9	-39.2	38.9	-2.2	-12.2	16.9
LPT	63.2	-8.8	-5.8	16.2	10.2	-21.8	-19.8	-22.8	-16.8	86.2
LPTB	30.7	-11.4	-34.4	6.7	-18.4	-4.4	20.7	-16.4	-14.4	69.7
JohnR	46.3	0.3	-1.8	15.3	-1.8	-21.8	-5.8	-11.8	-3.8	-2.8

Continuación de Tabla 4.3

Asignación	Observaciones									
	11	12	13	14	15	16	17	18	19	20
SPT	22.2	21.2	2.2	-30.9	-24.9	-37.9	27.2	-26.9	-32.9	21.2
SPTB	-4.2	3.9	-15.2	-39.2	-28.2	-25.2	47.9	-10.2	-42.2	30.9
LPT	-6.8	26.2	59.2	-24.8	-44.8	-41.8	3.2	-11.8	-40.8	2.2
LPTB	15.7	20.7	10.7	-39.4	-6.4	-23.4	23.7	-23.4	-42.4	35.7
JohnR	14.3	43.3	4.3	-29.8	-26.8	-17.8	21.3	-12.8	-28.8	20.3

Continuación de Tabla 4.3

Asignación	Totales	Medias
SPT	1.14E-13	5.684E-15
SPTB	-1.14E-13	-5.68E-15
LPT	-2.27E-13	-1.14E-14
LPTB	1.14E-13	5.684E-15
JohnR	0	0

La idoneidad del modelo se verifica a través de la revisión de la normalidad de los residuos, las varianzas iguales e independencia de los residuos.

La normalidad de los residuos se verifica mediante una gráfica de probabilidad. La gráfica de probabilidad normal de los residuos es un método común para determinar si los datos muestrales se ajustan a la distribución normal con base en un examen subjetivo de los datos:

1) la muestra $r_1, r_2, ..., r_l, ..., r_L$ se acomoda en el orden de no decrecimiento;

2) las observaciones r_l se grafican contra su frecuencia acumulada observada $(l - 0.5)/L$.

Si la distribución recibida describe de manera adecuada los datos, los puntos de la gráfica se ubicarán aproximadamente a lo largo de la línea recta; si los puntos se desvían significativamente de una línea recta, entonces el modelo propuesto no es idóneo. Al visualizar dicha línea se debe poner énfasis en los valores centrales de la gráfica.

La gráfica de la probabilidad normal para los residuos del experimento se presenta en la Figura 4.1. La gráfica muestra que en la parte central los residuos se aproximan a la línea recta, lo que confirma la suposición de la normalidad del modelo.

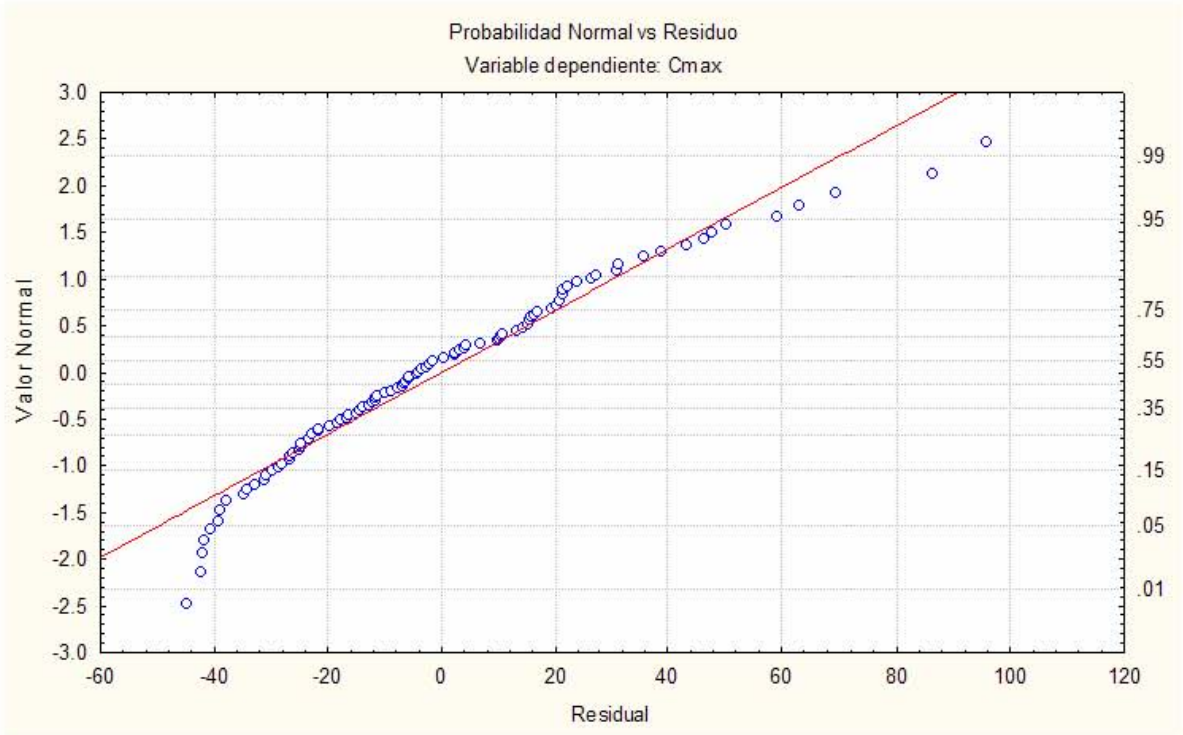

Figura 4.1. Gráfica de la probabilidad normal de los residuos

La Figura 4.2 muestra que los residuos se desvían en ambos lados con respecto a $C_{\max}$, y no existen desviaciones extremadamente grandes.

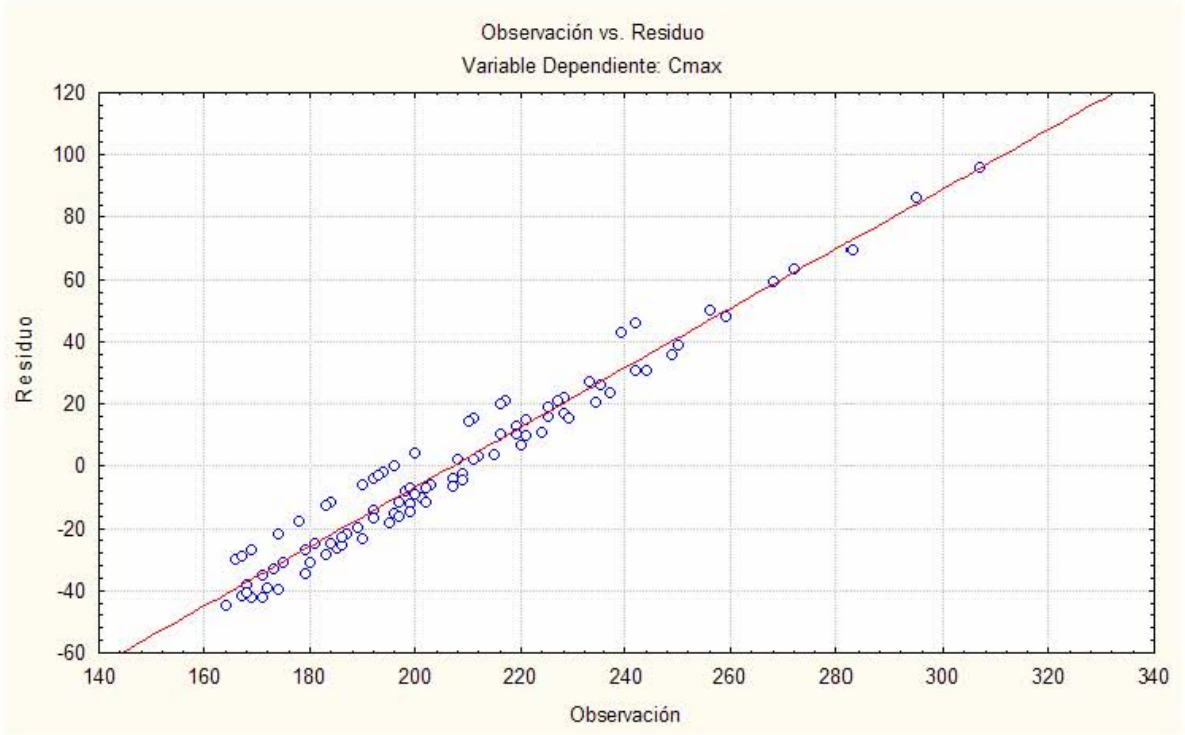

Figura 4.2. Gráfica de desviación de residuos con respecto a las observaciones

Las varianzas iguales para cada nivel del tratamiento se prueban con la gráfica de los residuos contra los niveles del factor y la gráfica de los residuos contra la media de tratamiento $\overline{y}_i$. Se compara la dispersión de los residuos. La variabilidad de los residuos no debe depender del tratamiento o de la media. Por ejemplo, la variabilidad de los residuos no debe aumentarse con $\overline{y}_i$. La aparición de un patrón en estas graficas sugiere la necesidad de una transformación.

En la Figura 4.3 se muestra la variabilidad de residuos con respecto al nivel de tratamiento y en la Figura 4.4 con respecto a las medias. Estas gráficas confirman la ausencia de un patrón entre residuos niveles de tratamiento o medias.

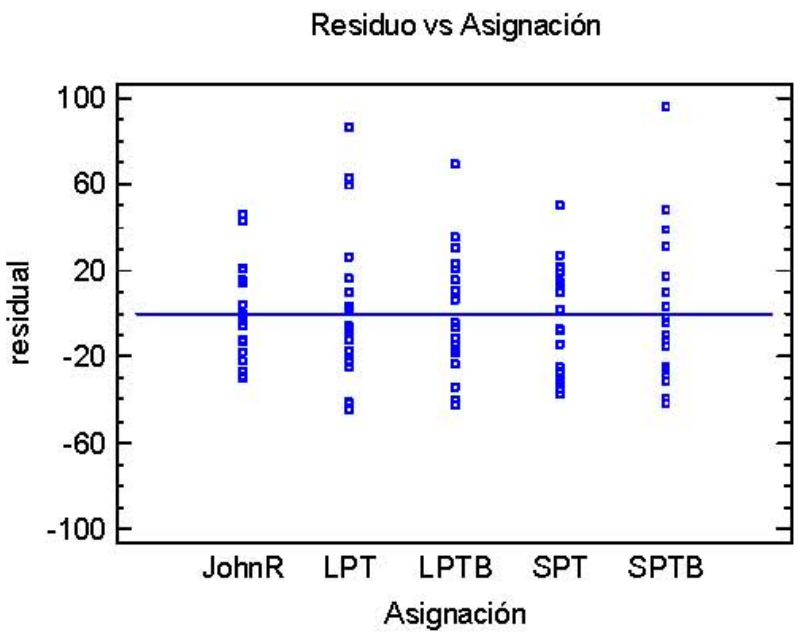

Figura 4.3.Gráfica de residuos contra niveles de factor

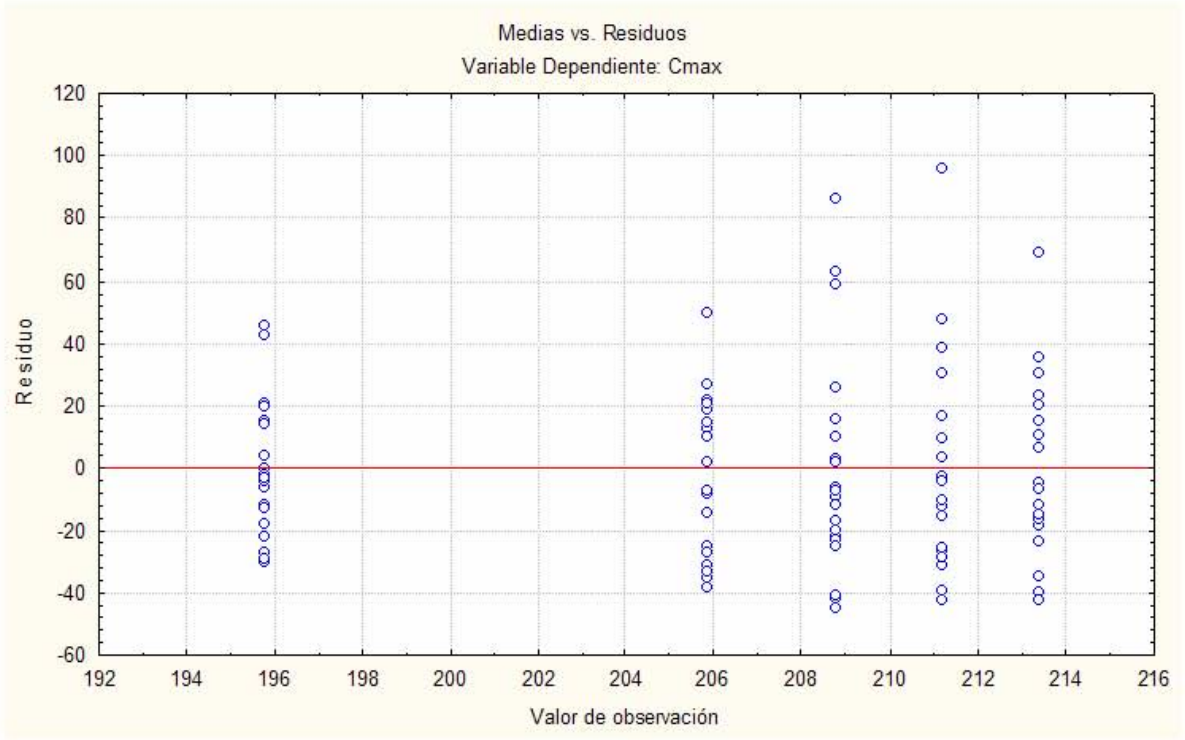

Figura 4.4. Gráfica de residuos contra medias de tratamiento

La independencia de residuos se confirma a través de la gráfica de los residuos contra el tiempo y orden de la corrida en que se realizó el experimento. Si el modelo es adecuado, los residuos no deben tener estructura. Un patrón en esta gráfica, tal como secuencias de residuos positivos y negativos, indica que las observaciones no son independientes.

En la Figura 4.5 se muestra la ausencia de una estructura o tendencia en los residuos.

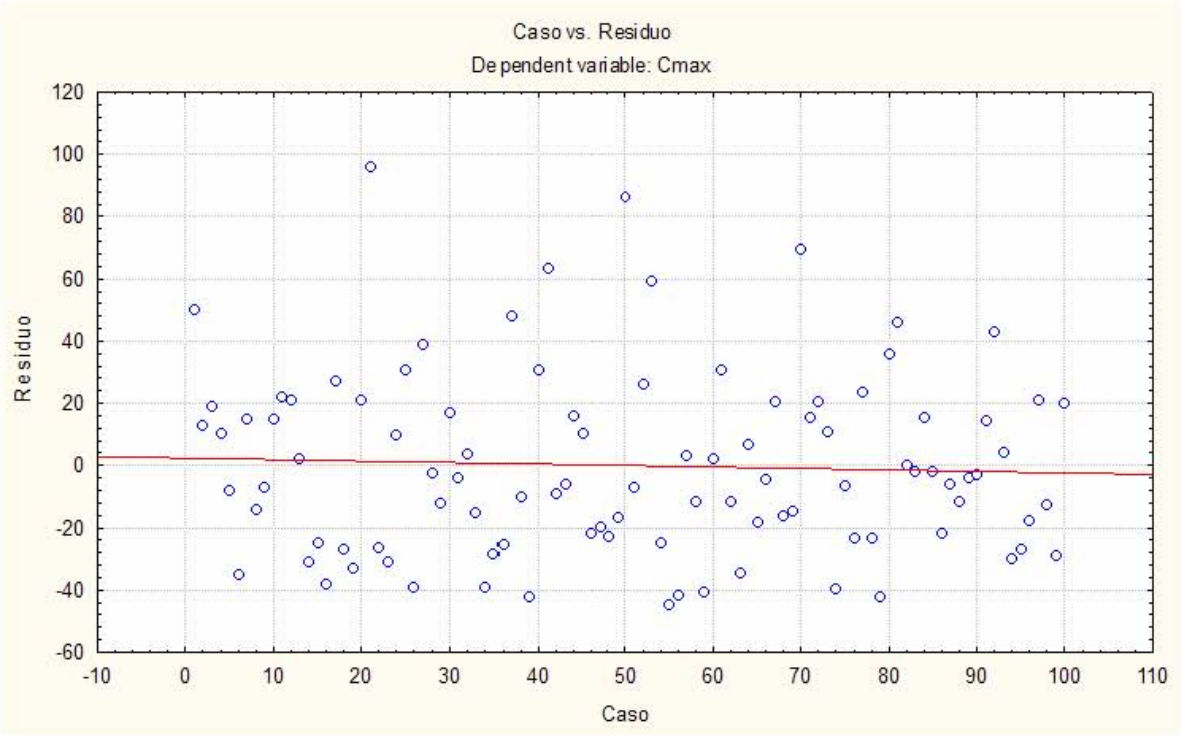

Figura 4.5. Gráfica de independencia de los residuos

De tal manera, las tres condiciones de idoneidad del modelo son confirmadas, lo que permite aplicar el análisis de varianza al modelo.

4.3.4 Análisis de varianza

El análisis de varianza mide la variabilidad total de los datos en dos componentes (Tabla 4.4). La variabilidad total de los datos esta descrita por la suma total de cuadrados (Montgomery & Runger, 1996).

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n}(y_{ij} - \overline{y}..)^2 \text{ ó } SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n} y_{ij}^2 - \frac{y^2}{N}.$$

Tabla 4.4. Tabla de análisis de varianza

Fuente de variación	Suma de cuadrados	Grados de libertad	Media de cuadrados	f_0
Tratamientos	$SS_{Tratamientos}$	$a-1$	$MS_{Tratamientos}$	$\frac{MS_{Tratamiento}}{MS_E}$
Error	SS_E	$a(n-1)$	MS_E	
Total	SS_T	$an-1$		

La suma total de cuadrados SS_T se divide por la suma de cuadrados de tratamientos $SS_{Tratamientos}$ y la suma de cuadrados del error SS_E:

$$SS_T = SS_{Tratamientos} + SS_E,$$

donde

$$SS_{Tratamientos} = \sum_{i=1}^{a} \frac{y_{i\cdot}^2}{n} - \frac{y_{\cdot\cdot}^2}{N}$$

$$SS_E = SS_T - SS_{Tratamientos}.$$

SS_T tiene $an-1$ grados de libertad, para $N = an$ observaciones.

$SS_{Tratamientos}$ tiene $an-1$ grados de libertad, puesto que existen a niveles de factor.

SS_E tiene $n-1$ grados de libertad, puesto que cualquier tratamiento contiene n réplicas.

El cociente $MS_{Tratamientos} = SS_{Tratamientos}/(a-1)$ es la media de cuadrados para tratamientos.

El cociente $MS_E = SS_E/(a(n-1))$ es el error cuadrático medio. Si la hipótesis nula H_0 es verdadera, el cociente

$$f_0 = \frac{SS_{Tratamientos}/(a-1)}{SS_E/[a(n-1)]} = \frac{MS_{Tratamientos}}{MS_E} \tag{4.2}$$

tiene una distribución F de Fisher con (a - 1) y $a(n$ - 1) grados de libertad. Debe rechazarse H_0 si la estadística es grande: H_0 debe rechazarse si $f_0 > f_{\alpha,a-1,a(n-1)}$, donde f_0 se calcula según la fórmula (4.2).

El análisis de varianza (ANOVA) para el experimento Redu_Experiment_20_2_2_3 se muestra en la Tabla 4.5. La tabla se generó a través del paquete STATISTICA 7.

Tabla 4.5. Análisis de varianza para el experimento Redu_Experiment_20_2_2_3

Fuente de Variación	Grados de libertad	Cmax SS	Cmax MS	Cmax f_0	Cmax p
Intersección	1	4284072	4284072	4836.181	0
Asignación	4	3773	943	1.065	0.378281
Error	95	84155	886		
Total	99	87928			

Para 4 grados de libertad en asignación, 95 grados de libertad en el error y $\alpha = 0.05$ (nivel de confianza del 95%), $f_{0.05,4,95} = 2.49$. El valor recibido por ANOVA es $f_0 = 1.065$.

Lo último implica que $f_0 < f_{0.05,4,95}$, por tanto se acepta la hipótesis nula H_0: las medias son iguales, es decir no existe efecto del tratamiento.

4.3.5 Análisis adicional

Resultados del análisis de varianza no permiten calibrar el algoritmo con el mejor algoritmo de asignación. Por tanto es necesario realizar un análisis adicional.

El experimento proporciona 960 casos resueltos para distintos parámetros del problema. Para que los parámetros sean comparables, es conveniente operar con los valores del Incremento del $C_{\max}$ (ANEXO H), los cuales son los valores relativos, en lugar de valores absolutos (crudos) de $C_{\max}$.

La Tabla 4.6 contiene algunos resultados específicos del experimento. El primer renglón contiene las medias de los algoritmos. Su comparación muestra que el mejor comportamiento en media lo tiene JohnR con 2.54032 mientras SPT tiene la media 3.975167. Los demás algoritmos muestran un comportamiento parecido a SPT.

Tabla 4.6. Análisis del Incremento de Cmax

No	Medición	Asignación				
		SPT	**SPTB**	**LPT**	**LPTB**	**JohnR**
1	Media	3.97517	3.876985	3.767272	3.332018	2.54032
2	No. de casos de obtención el mejor resultado	276	329	471	381	447
3	% de casos de obtención el mejor resultado	28.75	34.27083	49.0625	39.6875	46.5625
4	Valor máximo del Inc_{Cmax}	87	72.07207	54.41176	69.58525	74.87437
5	Redu_Experiment_20_2_1_5_003, Inc_{Cmax}	4.14747	8.294931	10.13825	**69.5853**	0
6	Redu_Experiment_20_2_1_5_014, Inc_{Cmax}	2.01005	1.507538	0	2.01005	**74.8744**
7	Redu_Experiment_20_2_1_5_015, Inc_{Cmax}	**87**	0	3.5	0	0
8	Redu_Experiment_20_2_2_5_003, Inc_{Cmax}	12.6126	**72.0721**	7.207207	0	11.71171
9	Redu_Experiment_20_2_2_5_004, Inc_{Cmax}	8.08824	22.79412	**54.4118**	0	6.617647

La comparación gráfica de las medias se presenta en la Figura 4.6.

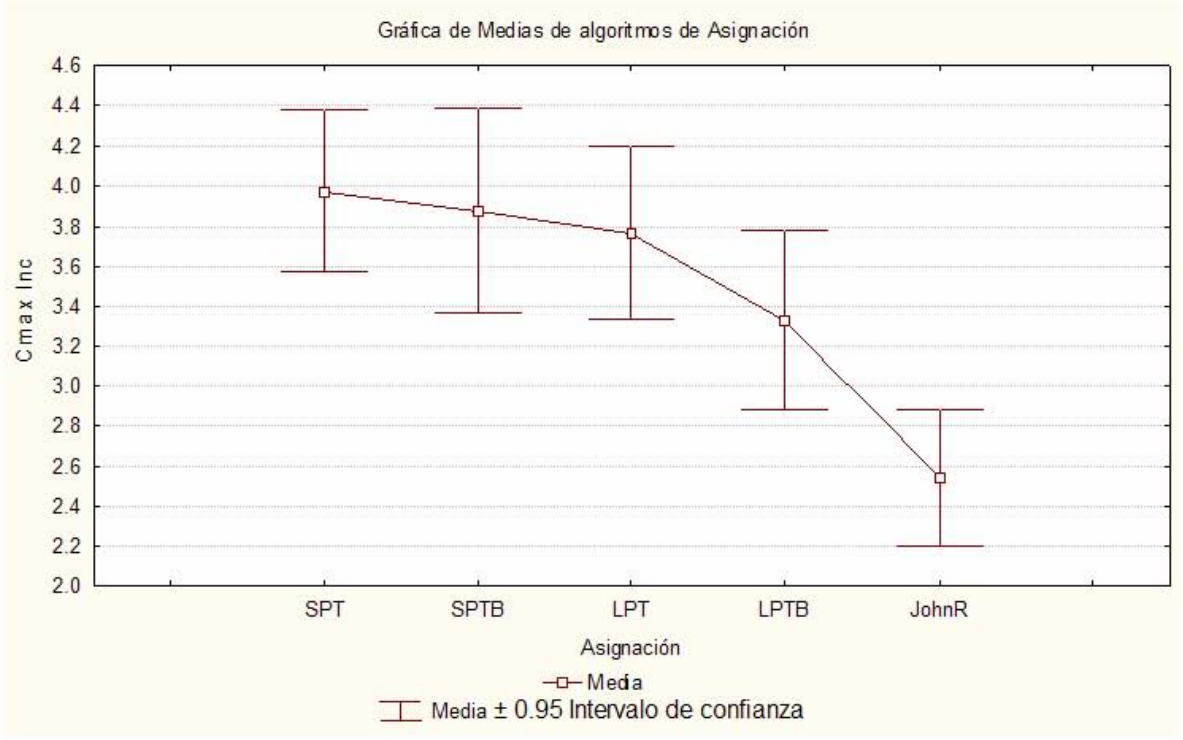

Figura 4.6. Gráfica de Medias de Algoritmos de Asignación en Incremento de Cmax

El renglón 2 (Tabla 4.6) contiene el número absoluto de casos, cuando el algoritmo especificado entrega el mejor resultado entre los 5 algoritmos (el valor "nulo" del incremento), y el renglón 3 contiene la misma información en porcentaje (con respecto a 960 observaciones). Estos renglones muestran el mejor comportamiento del algoritmo LPT, el cual en este caso supera en 2% el algoritmo JohnR, el mejor en media.

El renglón 4 de la Tabla 4.6 contiene el valor máximo del Incremento del $C_{\max}$ obtenido por cada algoritmo entre 960 observaciones. Se concluye que todos algoritmos muestran las desviaciones bastante grandes (54-87% en incremento) de la mejor solución.

Los renglones 5-9 para cada una de asignaciones muestran resultados de experimentos, en los cuales aparecen valores máximos del Incremento de $C_{\max}$ en comparación con el mejor obtenido entre las 5 asignaciones en el experimento indicado. Los datos muestran que todos algoritmos obtienen tanto mejores, como significativamente malos resultados.

En el ANEXO M, se muestran las medias de los tiempos de utilización de CPU para la obtención de resultados en cada una de las 48 combinaciones del modelo, aplicando las 5 asignaciones. La media máxima es 1874.05 milisegundos (1.87 segundos), la cual corresponde a los experimentos con el modelo de recursos M(5, 1) y 200 trabajos, mientras que para la media mínima el tiempo obtenido en los experimentos con el modelo de recursos M(2,5) y 20 trabajos es 70.15 milisegundos (0.07 segundos).

La alta eficiencia de los algoritmos de asignación, así como del algoritmo M+m, implica que los tiempos de utilización del CPU son prácticamente insignificantes. Por tanto, para aprovechar los mejores resultados obtenidos por todos los algoritmos de asignación, para cada archivo de entrada se efectúan las cinco asignaciones y se elige la mejor para procesarla en el algoritmo M+m.

4.4 Comparación del algoritmo M+m con el GABC-1

Para terminar la evaluación del algoritmo M+m, este debe compararse con todos algoritmos implementados en el sistema computacional, que resuelven el mismo problema. En este caso, con el algoritmo GABC-1.

La comparación de algoritmos M+m y GABC-1 incluye la comparación en la exactitud y en tiempo de utilización del CPU.

Para el análisis de la exactitud se usan los valores del Incremento de $C_{\max}$, los cuales se calculan a partir de la fórmula (3.2), donde *Best* se busca entre dos algoritmos comparados (ANEXO N).

En la Tabla 4.7 se presentan datos recibidos a partir del $Inc_{C\max}$ para la comparación en la exactitud entre algoritmos.

Tabla 4.7. Análisis del Incremento de Cmax de algoritmos M+m y GABC-1

No	Medición	M+m	GABC-1
1	Media	0.56	2.17
2	No. de casos de obtención el mejor resultado	801	195
3	% de casos de obtención el mejor resultado	83.4	20.3
4	Valor máximo del $Inc_{C\max}$	17.8	20.3

El primer renglón de la tabla contiene las medias de los algoritmos M+m y GABC-1. Su comparación muestra que el algoritmo M+m tiene el mejor comportamiento en media: 0.56 contra 2.17 para la media de GABC-1. La comparación gráfica de las medias se presenta en la Figura 4.7.

El renglón 2 contiene el número absoluto de casos, cuando el algoritmo especificado entrega el mejor resultado (el valor "nulo" del incremento), y el renglón 3 contiene la misma información en porcentaje (con respecto a 960 observaciones). Estos renglones muestran que el mejor comportamiento lo tiene el algoritmo M+m, el cual supera en 63.1% el algoritmo GABC-1. El valor máximo del $Inc_{C\max}$ muestra los peores casos para ambos algoritmos: 17.8 unidades de incremento para M+m mientras que para GABC-1 es más alto, de 20.3, lo que también confirma la prioridad del algoritmo M+m.

En la Tabla 4.8 se muestran los tiempos de CPU promediados para 20, 50, 100 y 200 trabajos de los algoritmos M+m y GABC-1 y la razón entre los resultados obtenidos.

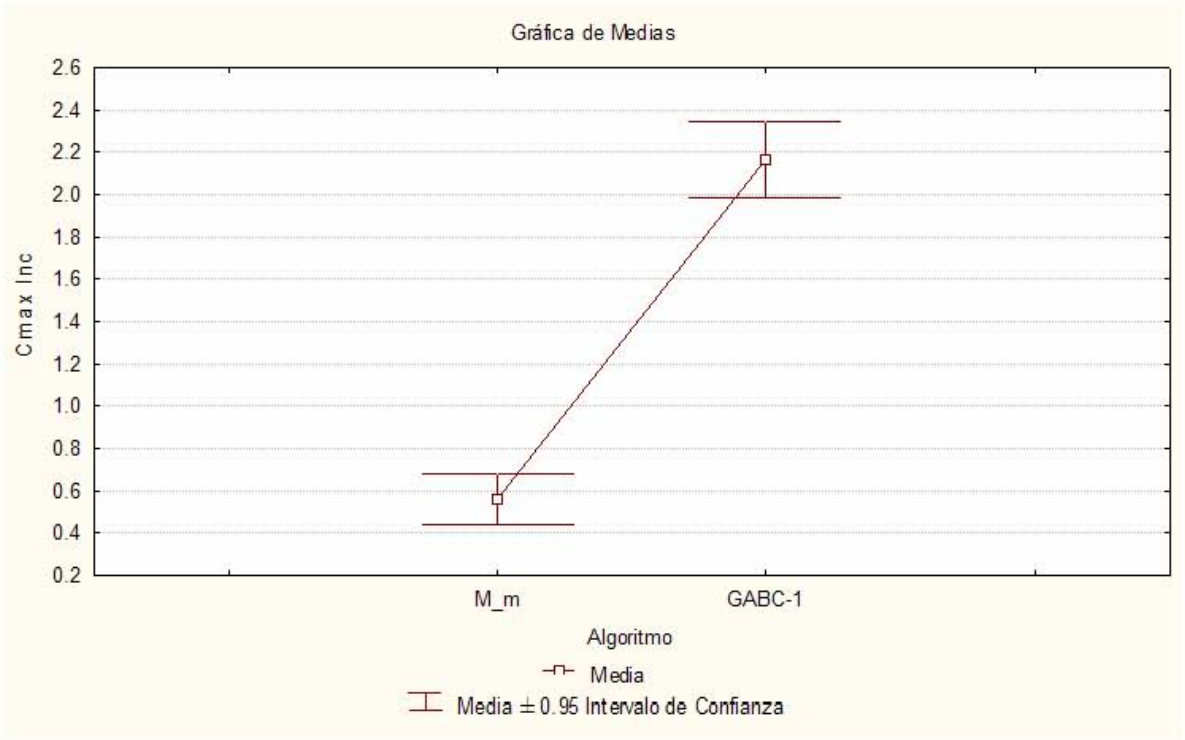

Figura 4.7. Gráfica de medias del Incremento de Cmax para M+m y GABC-1

Tabla 4.8. Análisis de tiempos CPU (promediados)

No	Medición	M+m	GABC1	GABC1/(M+m)
1	20	79.37	272.03	3.4
2	50	133.65	1062.93	8
3	100	327.24	5388.60	16.5
4	200	1530.93	35616.76	23.3
5	Total	2071.19	42340.31	20.4
6	Media	828.48	10585.08	12.8

La Tabla 4.8 muestra la prioridad completa del algoritmo M+m contra GABC-1, tanto para cada grupo de trabajos como para valores totales, medias y extremos. La comparación gráfica de las medias de la utilización de CPU se presenta en la Figura 4.8.

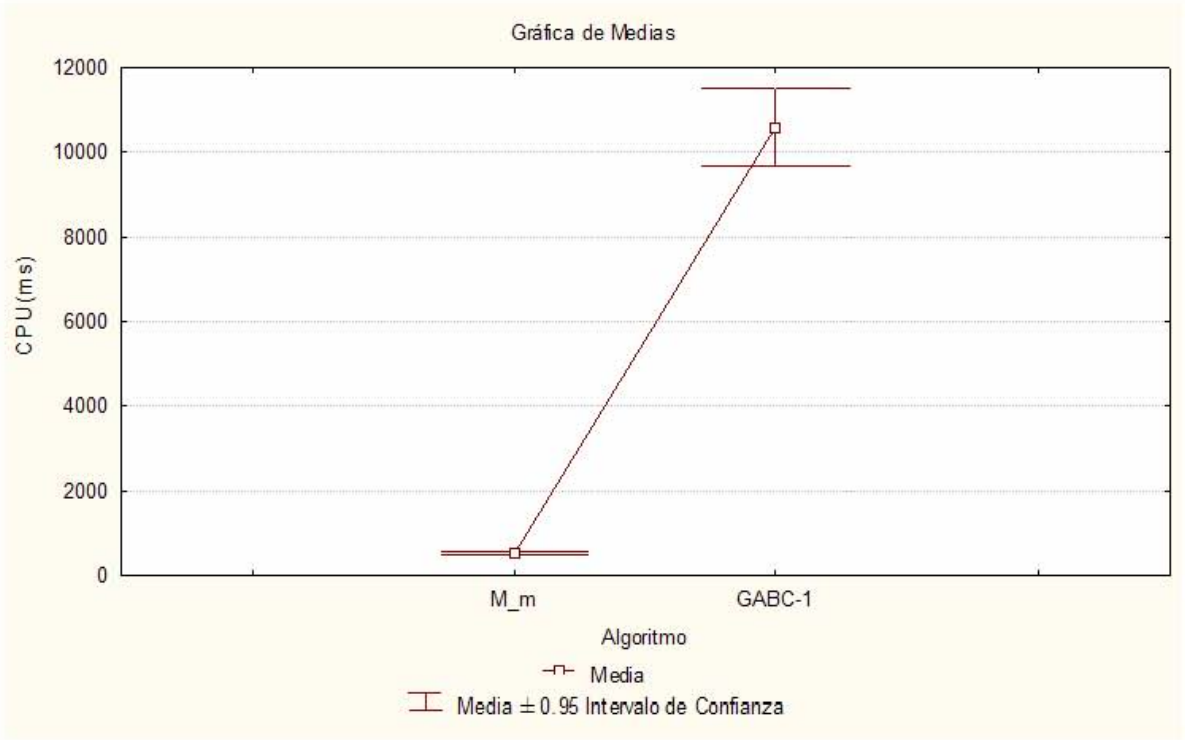

Figura 4.8. Gráfica de medias de tiempos de utilización de CPU

4.5 Conclusiones del capítulo

Para la verificación del funcionamiento del sistema PLARETF se realiza el experimento computacional de la evaluación de la exactitud y eficiencia del algoritmo M+m. A partir del análisis de los posibles factores que influyen en $C_{\max}$ se elige el diseño unifactorial con el algoritmo de asignación de trabajos a las máquinas como el factor. Se aplican 5 niveles del factor: SLPT, SPTB, LPT, LPTB y JohnR para análisis de su influencia sobre el valor de $C_{\max}$. Se utilizan 4 valores de número de trabajo y 12 valores de modelos, en total 48 combinaciones. Para calibrar el algoritmo M+m es necesario escoger el algoritmo de asignación, el cual proporciona los mejores resultados en $C_{\max}$. A partir de una combinación de parámetros se calculan estadísticas y se realiza un análisis comprobando la normalidad, homogeneidad e independencia de residuos, lo que permite aplicar el análisis de varianza ANOVA para probar la hipótesis nula: todas las medias del tratamiento son iguales. Como resultado del análisis de varianza se acepto la hipótesis nula y se recibe la conclusión de que el algoritmo de asignación no afecta directamente a $C_{\max}$.

El aceptar la hipótesis nula no permite calibrar el algoritmo M+m con un solo algoritmo de asignación. Todos los algoritmos de asignación resultaron estadísticamente iguales. La revisión del índice del incremento de $C_{\max}$ muestra, que los algoritmos de asignación complementan uno al otro. Tomando es cuenta la insignificancia del tiempo de utilización de CPU, se concluye que se apliquen todas las cinco asignaciones y se elija aquella que proporciona el mejor resultado en $C_{\max}$.

La evaluación del algoritmo M+m finaliza con su comparación con el algoritmo de referencia GABC-1. La comparación de estadísticas de ambos algoritmos muestra la prioridad absoluta del algoritmo heurístico contra el genético: más que en 4 veces en la obtención del mejor resultado (Tabla 4.7) y 12.8 veces menos en la utilización de CPU en un caso medio.

5 CONCLUSIONES

Para la elaboración de esta tesis se planteó como objetivo general crear un sistema computacional que permita evaluar experimentalmente la exactitud y complejidad de algoritmos para un TFH. Esto derivó en dos objetivos particulares, los cuales se cumplieron durante el desarrollo de la investigación. El procedimiento utilizado para cumplir con estos objetivos incluyó conocer a detalle los conceptos y notaciones involucrados en los problemas de planificación de trabajos para TFH. Dentro de estos conceptos se encuentran los modelos para solucionar problemas de planificación como estáticos y estocásticos, complejidad computacional, definición del problema de TFH. Además, se realizó una investigación de técnicas heurísticas y metaheurísticas utilizadas para resolver problemas de planificación de trabajos. De igual forma, se describió un algoritmo para minimización de la fecha máxima en un TFH2 con múltiples máquinas en ambas etapas (M+m), el cual se utilizó como parte del experimento computacional para demostrar el funcionamiento del sistema.

El desarrollo de nuevos algoritmos para la resolución de problemas de TFH agrega la problemática del desconocimiento de los valores reales de su exactitud y eficiencia. Esto motiva el desarrollo de metodologías para la comparación de distintos algoritmos. Una de las técnicas más comunes es el análisis estadístico de resultados. La implementación de técnicas avanzadas que permitan evaluar la calidad de algoritmos requiere el desarrollo de un sistema computacional flexible que permita incluir distintos algoritmos para resolución de distintos problemas. Los sistemas computacionales desarrollados por los investigadores del área han sido diseñados para evaluar la exactitud y complejidad de sus propios algoritmos, lo que impide utilizar estos sistemas para evaluar otros algoritmos.

Por esta razón se presenta como alternativa el sistema computacional PLARETF, cuyo propósito es ejecutar y estudiar algoritmos que resuelven problemas de TFH. Para el desarrollo del sistema se ha realizado una revisión de bibliografía sobre los parámetros utilizados por autores del área en sus experimentos computacionales.

El sistema desarrollado cuenta con una estructura dividida en tres secciones: ENTRADA, ORDENACIÓN Y SALIDA.

La sección ENTRADA contiene estructuras genéricas de datos, donde se guarda la información de modelo de recursos, tiempos de procesamiento, número de trabajos, y otros parámetros específicos del problema. Además estas estructuras tienen la funcionalidad de adaptarse a nuevas necesidades.

La sección ORDENACIÓN está diseñada como una colección de algoritmos para resolución de problemas de TFH. El método utilizado en un algoritmo no afecta el funcionamiento del sistema. Esto debido a que los algoritmos utilizan una copia de las estructuras genéricas de la sección de entrada para crear sus propias estructuras y así manipular la información con cualquier heurística o metaherurisca. Los algoritmos depositan los resultados en las estructuras de salida del sistema, localizados en la sección de salida.

La sección SALIDA recibe los resultados de la sección de ordenación y genera archivos con los resultados obtenidos por los algoritmos para su procesamiento y análisis posterior.

Tal organización del sistema permite realizar las siguientes funciones:

1. Mantener una colección de algoritmos para resolución de problemas de TFH.
2. Agregar nuevos algoritmos.
3. Seleccionar los algoritmos para ejecución.
4. Manipular los parámetros de ejecución de los algoritmos.
5. Desarrollar nuevos algoritmos.
6. Entregar los resultados en una forma estándar.
7. Proporcionar los datos para el procesamiento y análisis posterior.

El estudio de un algoritmo incluye:

- Aprobación de la codificación del algoritmo.

- Ejecución en el sistema.
- Análisis de resultados.
- Calibración.
- Comparación con un algoritmo de referencia.

La aprobación de la codificación del algoritmo supone su ejecución con un ejemplo de control. Una vez aprobado, el algoritmo se incluye en el sistema.

Para revisar el funcionamiento del sistema, se han implementado dos algoritmos que utilizan distintas metodologías en su diseño: M+m y GABC-1. El primero se toma como algoritmo de estudio y el segundo como referencia. El algoritmo M+m representa un algoritmo determinístico que recurre al método de dicotomía como esquema externo de optimización, mientras que GABC-1 es un algoritmo genético, de naturaleza estocástica.

Las variables independientes que intervienen en el algoritmo M+m son: el número de trabajos, modelo de recursos y algoritmo de asignación. Las dos primeras variables son parámetros del problema, mientras que la variación del algoritmo de asignación es propia del algoritmo. La variable dependiente es $C_{\max}$.

El análisis de resultados del algoritmo se realiza en un experimento computacional. El experimento ha sido diseñado como unifactorial, con el algoritmo de asignación como único factor de 5 niveles (algoritmos de asignación de trabajos a las máquinas: SPT, SPTB, LPT, LPTB y JohnR). Análisis residual confirmó la idoneidad del modelo del experimento, lo que permite aplicar el análisis de varianza para probar la hipótesis nula: los factores no influyen directamente en el resultado.

La calibración del algoritmo implica la elección de aquel nivel de factor, el cual proporciona los mejores resultados en $C_{\max}$, es decir, el mejor algoritmo de asignación.

La comparación de medias de los algoritmos de asignación muestra que el algoritmo JohnR proporciona los mejores resultados. Sin embargo, como resultado del análisis de varianza se acepta la hipótesis nula: ningún algoritmo de asignación tiene influencia directa sobre $C_{\max}$, debido a que todos los algoritmos de asignación tiene la misma probabilidad alcanzar un valor de $C_{\max}$ mínimo (*Best*).

Como conclusión se obtiene que no es posible calibrar el algoritmo M+m con un algoritmo de asignación utilizado.

El experimento mostró que el tiempo de ejecución del algoritmo M+m con las 5 variantes de asignación es insignificante. Por lo tanto, como una alternativa de calibración se eligió la ejecución de todos los algoritmos de asignación para cada carga de datos, seleccionando el mejor resultado, para procesarlo posteriormente.

El aceptar la hipótesis nula no implica que no existe un mejor algoritmo de asignación que influyera directamente sobre el valor de $C_{\max}$. Tal algoritmo permitiría calibrar el algoritmo M+m.

Para los algoritmos M+m y GABC-1 se ha realizado la comparación directa, contando los mejores resultados obtenidos para el mismo archivo de carga (en total 960 observaciones) y tomando en cuenta el tiempo de ejecución.

La comparación mostró la prioridad completa del algoritmo M+m contra GABC-1: en media aproximadamente 4 veces por la exactitud y 20 veces en relación al tiempo total de ejecución.

La creación del sistema computacional PLARETF y su aprobación con los procedimientos descritos anteriormente cumple con los objetivos definidos en la introducción de esta investigación y permite su futuro desarrollo.

Entre los problemas a resolver utilizando PLARETF se proponen los siguientes:

- Desarrollar un banco de pruebas propio de problemas resueltos de TFH.
- Desarrollar algoritmos de asignación de tareas utilizando métodos avanzados.
- Es interesante investigar el comportamiento del algoritmo M+m utilizando la mejor asignación del algoritmo evolutivo.
- Analizar algunos resultados secundarios como: tiempos muertos, cuellos de botella, la exclusión de máquinas excesivas en procesos de producción.
- Agregar arquitectura de cómputo de alto rendimiento para la experimentación eficiente.

PRODUCTOS GENERADOS EN EL DESARROLLO DE LA TESIS

- **Romero R.**, Yaurima V., Burtseva L.. *"Metodología De Comparación De Algoritmos Para La Programación De La Producción En Un Taller De Flujo Híbrido Con Dos Etapas"*. XXVIII CONGRESO INTERNACIONAL DE INGENIERIA ELECTRÓNICA (ELECTRO 2006). IT Chihuahua, Creel, Chihuahua. México. ISSN 1405-2172. Páginas: 329-332. 2006

- Yaurima V., **Romero R.**, Burtseva L., Valle Y. 2006. *"Aplicación Del Método De Dicotomía Para Optimización Combinatoria"*. XXVIII CONGRESO INTERNACIONAL DE INGENIERIA ELECTRÓNICA (ELECTRO 2006), IT Chihuahua, Creel, Chihuahua. México. ISSN 1405-2172. Páginas: 283-288. 2006

- Ruiz N. R, **Romero R**, Burtseva L. *Biblioteca Virtual Especializada.* IV Foro Nacional "La problemática en el aprendizaje de las ciencias básicas". SEP. IT Mexicali. Pagina: 9. 2007.

- Burtseva L., Yaurima V., **Romero R.**, Y. Valle. *Scheduling: Two Stage Hybrid Flowshop.*, Exposición del cartel realizada como parte de la celebración del XXV aniversario del Instituto de Ingeniería de la UABC en la semana de Computación e Informática. 11 del Mayo de 2006.

- Conferencia impartida dentro de los festejos de la 13ª. Semana Nacional de Ciencia y Tecnología, CBTyS No.33, San Luís R.C., Sonora, 24 de Octubre del 2006.

REFERENCIAS

Aho A., Hopcroft J. & Ullman J. *Estructuras de datos y algoritmos.* Wilmington, Delaware. Addison-Wesley Iberoamericana, Paginas: 438., 1988

Allaoui H. & Artiba A. *Scheduling two-stage hybrid flow shop with availability constraints.* Computers & Operations Research, Vol: 33: pages: 1399–1419. 2006.

Brassard G. & Bratley P. *Fundamentals of Algorithmics.* Prentice Hall, Englewood Cliffs, NJ, 1996.

Selman Bart, Henry Kautz, and Bram Cohen. *Local Search Strategies for Satisfiability Testing*. Final version appears in Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993. David S. Johnson and Michael A. Trick, ed. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, AMS, 1996.

Burden R.L. & Faires J. D. *Análisis Numérico.* Grupo Editorial Iberoamérica, 1985.

Burtseva L., Yaurima V. Espinoza Guzmán D. y Arce Martínez O. *Calendarización de trabajos en un Flexible Flow Shop con dos etapas.* Memoria del XXVII Congreso Internacional de Ingeniería Electrónica ELECTRO 2005, Vol. XXVI. Creel, Chih., México, Páginas: 65-70. 2005.

Cambell H.D., Dudek R.A. & Smith M.L. *A Heuristic Algorithm for the n Job, m machina Sequencing Problem.* Management Science, Vol: 16 Sec: 10: Pages: 630-637. 1970.

Cerny V. *Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm.* J. of Optimization Theory and Applications, Vol: 45, Sec: 1, Pages: 41–51. 1985.

Ceyda Oguz, M. Fikret Ercan & Yu-Fai Fung. *Heuristic Algorithms for Scheduling Multi Layer Computer Systems.* IEEE Internacional Conferencie on Intelligent Processing Systems. Beijing, China. Pages: 1347-1350. 1997.

Colorni A., Dorigo M., & Maniezzo V. *Distributed Optimization by Ant Colonies*. In Varela, F. and Bourgine, P., editors, Proceedings of ECAL'91 - First European Conference on Artificial Life. Pages 134–142, Paris, France. Elsevier Publishing. 1992.

Croes G. A. *A method for solving traveling salesman problems.* Operations Research. Vol: 6. Pages: 791-812. 1958.

Chelouah R. & Siarry P. *A Continuous Genetic Algorithm Designed for the Global Optimization.* Journal of Heuristics, 6:191–213. 2000a.

Chelouah R. & Siarry P. *Tabu Search Applied to Global Optimization.* European Journal of Operational Research. Vol: 123. Pages: 256–270. 2000b.

Chen Bo. *Analysis of Classes of Heuristics for Scheduling a Two-Stage Flow Shop with Parallel Machines at One Stage.* The Journal of the Operational Research Society. Vol: 46. Issue: 2. Pages: 234-244. February, 1995.

Dorigo M. & Gambardella L. M. *Ant Colony System: A cooperative learning approach to the traveling salesman problem.* IEEE Transactions on Evolutionary Computation, Vol: 1. Issue: 1.Pages: 53–66. 1997.

Fogel L. J., Owens A. J. & Walsh M. J.. *Artifical Intelligence through Simulated Evolution.* Wiley. 1966.

Garey M.R. & Johnson D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA. 1979.

Fraser A. S. *Simulation of genetic systems by automatic digital computers.* Australian Journal of Biological Sciences, Vol: 10. Pages: 484–491. 1957.

Garey M. & Johnson D. *Computers and Intractability. A Guide to the Theory of NP-completeness.* Freeman, San Francisco, CA, 1979.

Glover F. *Future paths for integer programming and links to artificial intelligence.* Computers and Operations Research. Vol: 13. Pages: 533–549. 1986.

Glover F. & Laguna M. *Tabu Search.* Kluwer Academic Publishers. 1997.

Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine learning.* Addison-Wesley. Pages: 432. ISBN-10: 0201157675. January 1, 1989.

Graham R. L., Lawler E. L., Lenstra J. K., & Rinnooy Kan A. H. G. *Optimization and Approximation in Deterministic Sequencing and Scheduling: A survey . Annals of Discrete Mathematics.* Vol: 5. Pages: 287-326. 1979.

Guinet A. *A computational study of heuristics for two-stage flexible flowshops.* International Journal of Production Research. Vol: 34. Issue: 5. Pages: 1399-1415, May 1996.

Guirchoun S., Martineau P. & Billaut J.C. *Total completion time minimization in a computer system with a server and two parallel processors,* Computers & Operations Research, Vol: 32. Pages: 599–611. 2005.

Gupta J.N.D. *Two-Stage, Hybrid Flowshop Scheduling Problem.* Journal of the Operational Research Society. Vol: 39. Issue: 4. Pages: 359–364. 1988.

Gupta J. *Schedules for a two-stage hybrid Flowshop with parallel machines at the second stage.* Journal of the Operations Research Society, Vol: 29. Issue: 7. Pages: 1489-1502. 1991.

Gupta J. & Tunc E.A. *Schedules for a two-stage hybrid flowshop with parallel machines at the second stage.* International Journal of Production Research. Vol: 29. Issue: 7. Pages: 1489–1502. 1991.

Handbook of Scheduling, algorithms, Models, and Performance Analysis. Edited by Joseph Y-T. Leung. CHAPMAN & HALL/CRC. 2004.

Holland John H. *Outline for a logical theory of adaptive systems.* J. Assoc. Comput. Mach (JACM). Vol: 9. Issue: 3. Pages: 297–314. ISSN: 0004-5411. 1962.

Johnson S.M. *Optimal two and tree-stage production schedules with setup time included.* Naval. Research Logistics Quarterly 1, Pages: 61-68. 1954.

Kelley Dean. *Teoría de autómatas y lenguajes formales.* PRENTICE HAL. Pages: 302. 1995.

Kirkpatrick S., Gelatt C. & Vecchi M. *Optimization by simulated annealing.* Science, Vol: 220. Issue: 4598. Pages: 671–680. 1983.

Land A. H. & A. G. Doig. *An automatic method of solving discrete programming problems.* In: Econometrica 28, S. Pages: 497-520. 1960

Lee C-Y. & Vairaktarakis G.L.. *Minimizing makespan in hybrid flowshops.* Operational Research Letters Vol: 1 Issue: 6. Pages: 149–58. 1994.

Ling-Huey S. *A hybrid two-stage Flowshop with limited waiting time constraints.* Computers & Industrial Enginering. Vol: 44. Pages: 409-424. Pages: 2003.

Metropolis N., Rosenbluth A. N. Rosenbluth M. N., Teller A. H., & Teller E. *Equation of state calculation by fast computing machines.* Journal of Chemical Physics, Vol: 21. Issue: 6. Pages: 1087–1092. 1953.

Mijalevich Z. *Genetic Algorithm + Data Structures = Evolutions Programs.* Third (ed)., Springer'Verlag, Berlin. 1996.

Montgomery D.C. *Diseño y Análisis de Experimentos,* Grupo Editorial Iberoamérica, México, Páginas: 590. 1991.

Montgomery D.C. & Runger G. C. *Probabilidad y estadística aplicadas a la ingeniería.* McGRAW-HILL, México, Páginas: 910. 1996.

Pinedo M. *Scheduling: Theory, Algorithms, and Systems.* New Jersey: Prentice Hall, Pages: 586. 2002.

Rechenberg I. *Cybernetic Solution Path of an Experimental Problem.* Royal Aircraft Establishment Library Translation. 1965.

Renders J. & Flasse S. *Hybrid methods using genetic algorithms for global optimization.* IEEE Trans. on Systems, Man, and Cybernetics — Part B: Cybernetics. Vol: 26. Issue: 2. 1996.

Riane F., Artiba A., Elmaghraby S.E. *Sequencing a hybrid two-stage flowshop with dedicated machines*, International Journal of Production Research, Vol: 40. Issue: 17. Pages: 4353-4380. 2002.

Romero R., Yaurima V., Burtseva, L. *Metodología de comparación de algoritmos para la programación de la producción en un taller de flujo híbrido con dos etapas.* XXVIII Congreso Internacional de Ingeniería Electrónica ELECTRO 2006, IT de Chihuahua, Creel, Chih., México, Vol: XXVIII. Paginas: 329-332. 2006.

Rosen K.H. *Mathematica Discreta y sus Aplicaciones.* McGraw-Hill-Interamericana de España, Pages: 860. 2004

Ruiz R. y Maroto C. Evaluación de Heurísticas para el problema del Taller de Flujo. 27 Congreso de Estadística e Ingeniería Operativa. 2003.

Siarry P. & Berthiau G. *Fitting of tabu search to optimize functions of continuous variables.* International Journal for Numerical Methods in Engineering, Vol: 40. Pages: 2449–2457. 1997.

Sriskandarajah C. & Sethi S.P. *Scheduling algorithms for flexible flowshops: Worst and average case performance.* European Journal of Operational Research, Elsevier, Vol: 43 Pages: 143–160. 1989.

Taillard É. D. *Benchmarks for basic scheduling problems.* European Journal of Operational Research. Vol: 64. Pages: 278-285. 1993.

Xie J. & Wang X. *Complexity and Algorithms for Two-Stage Flexible Flowshop Scheduling with Availability Constraints*, Computers and Mathematics with Applications, Vol: 50. Pages: 1629-1638. 2005.

Yaurima V., Romero R., Burtseva L. y Valle Y. *Aplicación del método de dicotomía para optimización combinatoria.* XXVIII Congreso Internacional de Ingeniería Electrónica ELECTRO 2006, IT de Chihuahua, Creel, Chih., México, Vol: XXVIII. Páginas: 283-288. 2006.

Yaurima V., Burtseva L., & Tchernykh A. *Hybrid Flowshop with Unrelated Machines, Sequence Dependent Setup Time and Availability Constraints: An Enhanced Crossover Operator for a Genetic Algorithm.* The Seventh International Conference on Parallel Processing and Applied Mathematics PPAM. Gdansk, Poland, 2007.

ANEXOS

ANEXO A. Archivos de experimento "Redu_Experiment_20_2_1_2_001.txt"

Vea CD Tesis: Experimento_20_2_1_2_001.pdf

ANEXO B. Archivo de configuración del sistema computacional "config.ini"

Vea CD Tesis: Configuración.pdf

ANEXO C. Archivo de la lista de los archivos de carga "listacarga.ini"

Vea CD Tesis: Carga.pdf

ANEXO D. Función de la inicialización de estructuras internas del sistema

Vea CD Tesis: Inicializacion_sistema.pdf

ANEXO E. Archivo historial

Vea CD Tesis: Historial.pdf

ANEXO F. Ejemplo del archivo Cmax_normalizado aplicado al algoritmo M+m

Vea CD Tesis: Cmax_normalizado_M_mas_m.pdf

ANEXO G. Ejemplo del archivo Cmax_normalizado aplicado al algoritmo GABC-1

Vea CD Tesis: Cmax_normalizado_GABC_1.pdf

ANEXO H. Ejemplo del archivo Cmax_incremento aplicado al algoritmo M+m

Vea CD Tesis: Cmax_incremento_M_mas_m.pdf

ANEXO I. Ejemplo del archivo Cmax_incremento aplicado al algoritmo GABC-1
Vea CD Tesis: Cmax_incremento_GABC_1.pdf

ANEXO J. Archivo de resultados Global_Data_Result.txt

Vea CD Tesis: Resultados_global.pdf

ANEXO K. Resultados crudos para el algoritmo M+m

Vea CD Tesis: Cmax_crudos_M_mas_m.pdf

ANEXO L. Resultados crudos para el algoritmo GABC-1

Vea CD Tesis: Cmax_crudos_GABC_1.pdf

ANEXO M. Medias de los tiempos de utilización de CPU

Vea CD Tesis: media_cpu_time.pdf

ANEXO N. Análisis de exactitud M+m vs GABC-1

Vea CD Tesis: analisis_exactitud_M_mas_m_vs_GABC_1.pdf