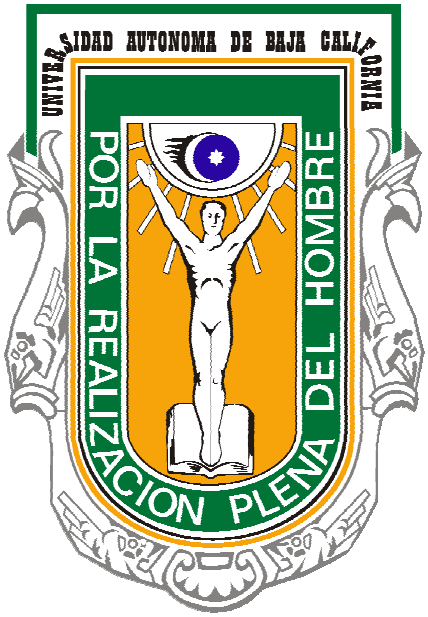
UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

INSTITUTO DE INGENIERÍA

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA



“Programación de la Producción para Casos Especiales en un Taller de Flujo Híbrido”

T E S I S

que para obtener el grado de
DOCTOR EN CIENCIAS
presenta

Víctor Hugo Yaurima Basaldúa

DIRECTOR
Dra. Larisa Burtseva

CO-DIRECTOR
Dr. Andrei Tchernykh

Mexicali, B. C. **Septiembre, 2008**

RESUMEN

Muchas empresas de manufactura enfrentan necesidades de mayor competitividad, mejores precios y aumento de rentabilidad. En un entorno de producción real comúnmente existen máquinas de diferentes marcas, capacidades y velocidades, y una creciente diversificación en los productos. Estas circunstancias propician cambios frecuentes en la configuración de las máquinas para la ejecución de diversos modelos del mismo producto, lo cual hace compleja la toma de decisiones. La elección del orden en el cual deben entrar los trabajos a la línea de producción y a qué máquina debe asignarse, representa un problema para las industrias, llamado “programación de la producción” que influye en la productividad de las máquinas, oportunidad del servicio y rentabilidad.

Abordar este problema a través del tiempo ha implicado el desarrollo de métodos y algoritmos para diferentes circunstancias en los modelos de producción. Sin embargo, a pesar que se lleva investigando muchos años, todavía existe una brecha considerable entre la teoría y la práctica, por lo cual, los problemas de la programación de la producción continúan en la mayoría de los casos sin una solución satisfactoria en entornos reales de producción y muchas industrias continúan realizando su planeación de manera manual.

Al modelo de un proceso productivo, con un conjunto de trabajos que se procesan sucesivamente en varias etapas, y donde cada etapa tiene un conjunto de máquinas capaces de ejecutar los trabajos, se le conoce como taller de flujo híbrido. Se ha demostrado en publicaciones relacionadas que el problema del taller de flujo híbrido es **NP**-Difícil. En la presente tesis se proponen algoritmos motivados por un modelo basado en un entorno real de producción de una industria electrónica de televisiones en la sección de auto inserción. El modelo representa un taller de flujo híbrido, con máquinas no relacionadas, tiempos de ajuste en las máquinas que dependen de la secuencia, elegibilidad de máquinas y búfer limitado.

Se proponen dos algoritmos heurísticos para resolver el taller de flujo híbrido con dos etapas. A partir del análisis de complejidad de los algoritmos propuestos se concluye

que poseen una alta eficiencia. Para el taller de flujo híbrido con múltiples etapas se propone un algoritmo genético, considerando las características del modelo en estudio basado en un caso real. Los resultados muestran que el algoritmo propuesto es entre 99% y 378% mejor que el algoritmo de referencia. Cabe mencionar que este modelo en particular no ha sido estudiado en su totalidad en la literatura y representa una nueva contribución.

Agradecimientos

Agradezco primeramente a Dios, nuestro Creador, que sin Él no existiríamos.

A mis maestros, de quienes aprendí no solamente en lo académico.

A mis compañeros, por el compartir esfuerzos y camaradería.

Al Dr. Rubén Ruiz a quien no tengo el gusto de conocer personalmente, pero me proporcionó sugerencias y comentarios valiosos en el transcurso de los trabajos.

A mis sinodales por aumentar la calidad de esta tesis con sus observaciones y recomendaciones.

Finalmente, a mis asesores la Dra. Larisa Burtseva y el Dr. Andrei Tchernkyh a quienes debo mucho de este trabajo.

Dedicatoria

A mi esposa María Jesús,
a mi hijo Victor Hugo
y al que “viene en camino”
a quienes amo
y son mi motivación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

Capítulo 1

Introducción

1.1 Problemática de la programación de la producción en la manufactura

Las industrias enfrentan necesidades de mayor competitividad, mejores precios y aumento de rentabilidad; una de las áreas que impacta en la satisfacción de estas necesidades, es la planeación de la producción. Un entorno de producción real, donde comúnmente existen máquinas de diferentes marcas, capacidades y velocidades y una creciente diversificación en los productos que propician cambios frecuentes en la configuración de las máquinas para la ejecución de diversos modelos del mismo producto, hacen compleja la toma de decisiones. Desde los años 50 se han desarrollado numerosos métodos y algoritmos basados en modelos teóricos. Muchas empresas de manufactura todavía realizan una programación de la producción por métodos manuales, lo que hace evidente la necesidad de enfocar la investigación en entornos reales de producción, hacia donde se enfoca el interés de esta tesis.

La elección del orden, en el cual deben entrar los trabajos a la línea de producción y a cuál máquina entre varias idénticas deben asignarse, representa un problema para muchas industrias; a esto se le llama “programación de la producción” que influye entre otras cosas en la productividad de las máquinas, oportunidad del servicio y la rentabilidad. Abordar este problema ha implicado a través del tiempo el desarrollo de métodos y algoritmos para diferentes circunstancias en los modelos de producción. Sin embargo, a pesar que se lleva investigando más de cincuenta años, todavía existe una brecha importante entre la teoría y

la práctica, por lo cual, los problemas de la programación de la producción permanecen en la mayoría de los casos sin una solución satisfactoria en entornos reales de producción y muchas industrias continúan realizando su planeación de manera manual.

Hay dos tipos de sistemas de producción (Johnson & Montgomery, 1974):

1. Sistemas de producción en continuo, en los cuales no hay mucha variación en los productos y se produce a gran escala. (Ejemplo: Industria del acero).
2. Sistemas de producción intermitentes o discretos, en los cuales los productos son diferentes pero a la vez similares, se requieren ajustes en las líneas de producción para cada tipo de producto. (Ejemplo: Industria electrónica).

Independientemente de los tipos de sistemas de producción, una empresa de manufactura debe hacer una planeación de la producción incluyendo en éste un "plan maestro de producción" (*master plan schedule*), con el cual define ciertos parámetros como el personal fijo a laborar, turnos, recursos financieros, productos a fabricar, cantidades, recursos disponibles, etc. En la programación de la producción se supone que la planeación ya se ha realizado, es decir, se conocen estos parámetros. A partir del plan maestro se realiza la programación de la producción, que establece asignación y secuenciación de los distintos pedidos del plan a las estaciones de trabajo, determinando los momentos de inicio y finalización para cada operación (Companys-Pascual & Corominas-Subias, 1996).

La programación de la producción (*scheduling*) es una etapa importante que debe tomarse en cuenta explícitamente para el logro de los objetivos en la producción (Thomas, 1993). El criterio de optimización más utilizado para este problema es la minimización del tiempo máximo de finalización (*makespan*). Este objetivo consigue una eficiente utilización de las máquinas y permite una finalización más rápida posible del producto.

Una de las primeras investigaciones dedicadas a problemas de la programación de la producción fue realizada por Johnson (1954), quien encuentra una solución óptima para ejecutar un conjunto determinado de trabajos en dos máquinas sucesivas. En las décadas de los 60 y 70 aparecen contribuciones en el campo de las técnicas exactas (programación dinámica, algoritmos de ramificación y acotación), los cuales son capaces de proporcionar soluciones a este tipo de problemas aunque muy concretos o de tamaño reducido. Después del desarrollo de la teoría de la complejidad, la comunidad científica se centra en técnicas heurísticas (Garey & Johnson, 1979). En los años 80 aparecen varios algoritmos de carácter

aproximado para resolver problemas de programación de la producción. Desde los años 90 hasta la actualidad se ha incrementado la variedad de técnicas avanzadas como algoritmos genéticos (*genetic algorithms*), recocido simulado (*simulated annealing*), búsqueda tabú (*tabu search*), búsqueda local iterativa (*iterated local search*), GRASP (*Greedy Randomized Adaptive Search*) (Allaoui & Artiba, 2006).

1.2 Producción de televisiones

Esta tesis enfoca su atención en la programación de la producción para la industria electrónica de televisiones, donde se presenta un problema complejo. En este apartado se describe el análisis del ambiente de producción, para lo cual se realizó un seguimiento a una planta productiva.

El proceso de manufactura de televisiones tiene tres secciones (Figura 1.1):

1) Auto Inserción: Se insertan componentes en tableros de circuito impreso (PCB - *Printed Circuit Board*) de manera automatizada.
2) Montaje Manual: Operadores ensamblan e insertan componentes grandes de manera manual.
3) Final: Se realizan pruebas y se hace el empaque final del producto terminado para su transportación a los distribuidores.

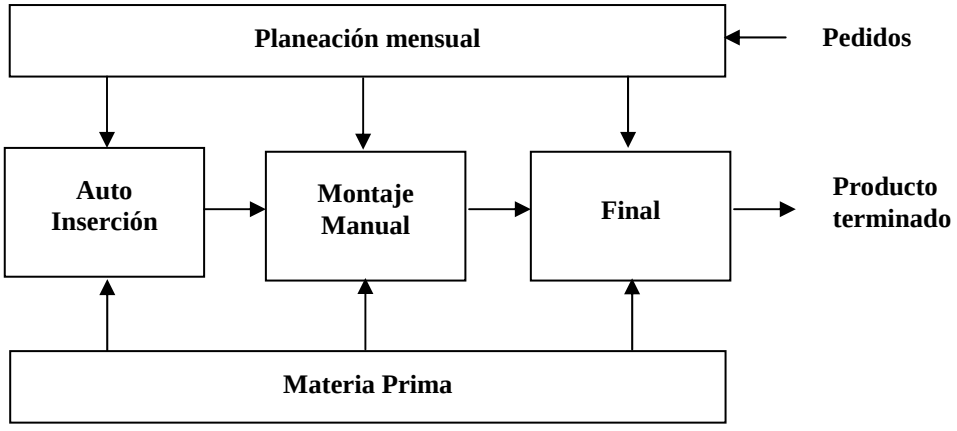


Figura 1.1 Proceso de la manufactura de televisiones

A partir del Plan de Producción mensual se calculan las necesidades en PCBs, usados en el ensamble de las televisiones y se determina el orden de su ejecución en las máquinas. En total se procesan mensualmente un promedio de 500,000 tableros para diferentes modelos de televisiones. La planeación mensual está sujeta a cambios durante el mes en curso por variaciones en los pedidos y circunstancias externas que se gestionan diariamente en la sección “Final”.

La presente investigación se enfoca en la sección de auto inserción. Siendo que esta sección es automatizada, los tiempos de procesamiento en cada máquina son estadísticamente establecidos. La sección cuenta con 45 máquinas distribuidas de acuerdo a los procesos que realizan (Figura 1.2):

1. Eyelet (Ey), se realizan orificios para insertar componentes.
2. Jumper (Jv), se insertan grapas (*jumpers*) para conectar circuitos.
3. Axial (Ax), se insertan componentes axiales de longitud fija.
4. Axial variable (Av), se insertan componentes axiales de longitud variable.
5. Radial (Rd), se insertan componentes radiales.
6. Surface Mount (Sm), se aplica soldadura.

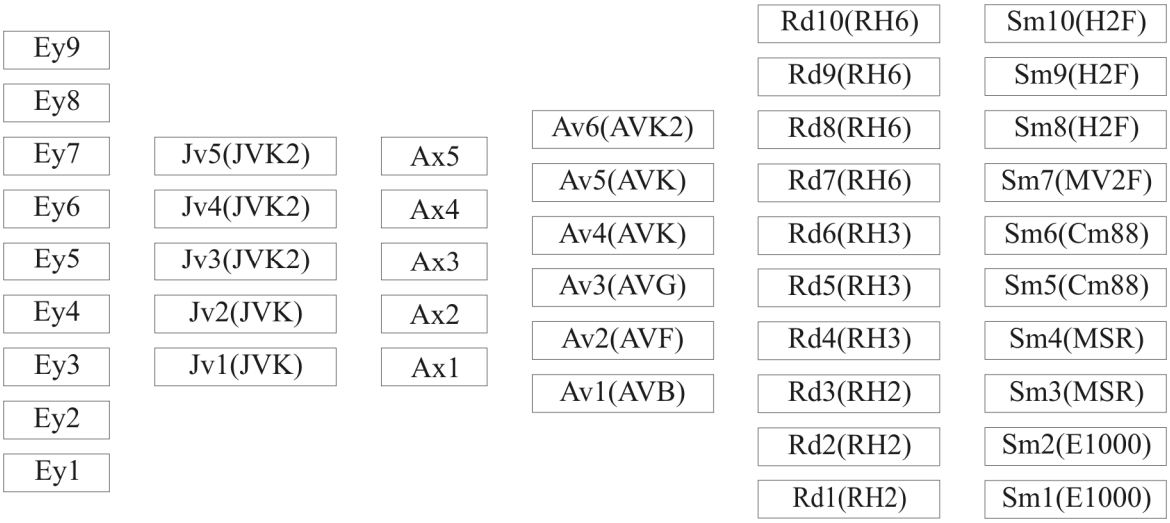


Figura 1.2 Planta productiva en la sección de auto inserción

La Figura 1.3 muestra el esquema de una máquina del tipo AVK2.

Cada tablero requiere un número de inserciones de componentes en cada máquina. El tiempo de procesamiento de un tablero varía de acuerdo a la marca de la máquina que lo ejecuta y se calcula a través de estadísticas de la capacidad de la máquina, específicamente de acuerdo al número de inserciones por segundo. En el Anexo A se presentan estadísticas acerca de las capacidades de las máquinas.

El plan actual de la empresa incluye televisiones de 11 familias. Los modelos de las televisiones se distinguen principalmente por los tipos de tableros, de tal manera que una familia de televisiones incluye varios modelos y cada modelo define la lista de tableros usados para su ensamble. En el Anexo B, se enumeran las familias de las televisiones, sus modelos y tableros.

Las características del ambiente productivo en esta sección, son las siguientes:

1) Los PCBs deben pasar sucesivamente por varios procesos, específicamente de 2 a 6, dependiendo del tipo de PCB.

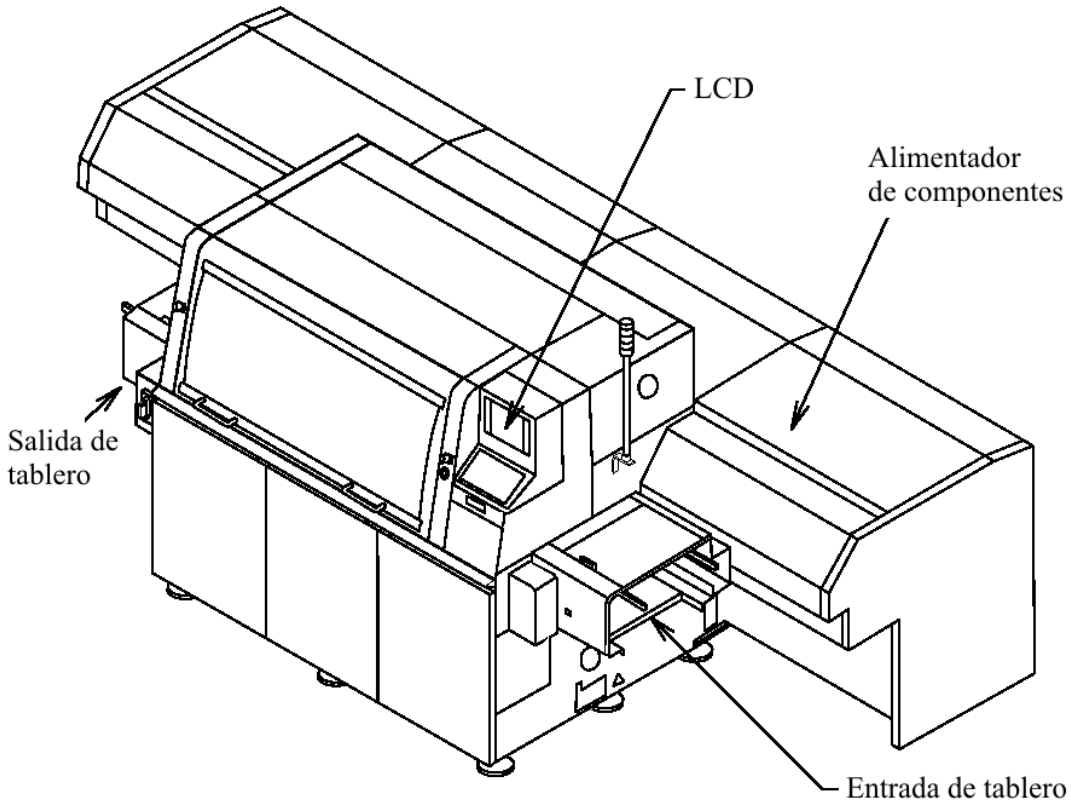


Figura 1.3 Esquema de una máquina del tipo AVK2

2) En cada proceso existen grupos de máquinas de distintas marcas, velocidades y capacidades. El tiempo de procesamiento de cada tarea depende de la máquina en particular.

3) Cada máquina cuenta con un alimentador de componentes (*feeder*), los cuales permiten la entrada de un número de componentes electrónicos para procesar un tablero. El conjunto de componentes depende del tipo de tablero; pues cada PCB requiere cierto número y tipos de componentes.

Los alimentadores tienen capacidades diferentes. Por ejemplo, una máquina tiene un alimentador con capacidad para 60 componentes, otra para 80. El alimentador de componentes de una máquina debe ser preparado antes de procesar los trabajos; el número de componentes dispuestos en el alimentador depende del tipo de tablero.

El tiempo de preparación de cada alimentador en una máquina y su ajuste correspondiente requerido por el cambio de tipo de tablero, depende esencialmente del tablero que pasó previamente por esa máquina, lo que significa que existe un tiempo de ajuste dependiente de la secuencia (*Sequence Dependent Setup Time* - SDST).

4) Una máquina está no habilitada (*unable*) cuando se encuentra en mantenimiento o porque no es capaz de procesar cierto tipo de trabajo; lo cual implica la consideración de disponibilidad / elegibilidad de máquinas (*Machine availability* / *elegibility*).

5) Cada máquina dispone de un espacio físico limitado para el almacenamiento temporal (búfer) de los trabajos procesados. Estos trabajos en el búfer son llamados inventario en proceso (*Work in Process* - WIP). Si se llega al tope de la capacidad en el espacio de una máquina, ésta no es capaz de seguir produciendo por lo que habría un "bloqueo" (*block*) en la producción. A esta característica se le conoce como búfer limitado (*limited buffer*).

6) Se requiere minimizar el tiempo total de procesar un conjunto de trabajos.

En resumen, este entorno productivo representa un taller de flujo híbrido (TFH) de seis etapas con máquinas paralelas no relacionadas, tiempos de ajuste dependientes de la secuencia, elegibilidad de máquinas y búfer limitado. Aumentar la productividad, implica establecer un orden en el flujo de los trabajos que permita un mayor uso de las máquinas de tal forma que se minimice el tiempo de completar los trabajos.

Las características y restricciones del ambiente productivo hacen compleja la toma de decisiones, la cual representa un problema para la empresa al querer aumentar la productividad. Otras condiciones como la diversificación en los productos, cambios frecuentes en los planes y dinámica en la disponibilidad de las máquinas implican cambios frecuentes en la programación de la producción.

Actualmente la programación de la producción se realiza por un grupo de personas (*planners*) basándose en la experiencia, por métodos manuales, y por lo tanto no garantiza buenas y rápidas soluciones. Por lo tanto, aumentar la productividad requiere mejorar la toma de decisiones en la programación de la producción.

1.3 Objetivo

El objetivo general de esta tesis es desarrollar algoritmos para la programación de la producción que ayude a optimizar un taller de flujo híbrido considerando un entorno de producción real, específicamente la sección de auto inserción en la industria electrónica de televisiones.

Objetivos particulares:

1. Formalizar el modelo del problema de programación de la producción en el contexto de una industria electrónica de televisiones en la sección de auto inserción.
2. Desarrollar algoritmos para casos con dos y múltiples etapas de un taller de flujo híbrido en la programación de la producción considerando un caso real de la industria electrónica de televisiones.

1.4 Metodología

Primeramente, se analiza el modelo del entorno de producción real, a través de una estancia de dos meses en la planta productiva dando seguimiento a las actividades de planeación y producción.

Se formaliza el modelo bajo estudio y se identifican las características de los procesos, recursos y el criterio de optimización.

Se realiza una revisión de la literatura acerca de la programación de la producción identificando las distintas contribuciones relacionadas con el caso real.

Se analizan casos especiales dentro del modelo teórico y su utilidad tomando en cuenta el número de etapas y restricciones adicionales.

Se aplican métodos heurísticos y metaheurísticos para desarrollar algoritmos que contribuyan a la solución del problema formulado en el apartado 1.2, tomando en cuenta investigaciones previas relacionadas.

Se evalúa la exactitud y complejidad de los algoritmos desarrollados a través de un análisis teórico y estadístico.

Finalmente, se analizan los resultados y se presentan las conclusiones.

1.5 Esquema general de la tesis

Después de la introducción, en el capítulo dos se presentan los conceptos preliminares donde se describen las notaciones. Se hace una clasificación de este tipo de problemas y se comenta sobre su complejidad computacional. En el capítulo tres se estudia el problema del taller de flujo híbrido (TFH), el cual representa un modelo básico del caso bajo estudio, presentando los métodos que se han utilizado, así como una revisión exhaustiva del estado del arte para este tipo de problemas. En el capítulo cuatro se presentan contribuciones al caso de dos etapas: con una máquina en la primera etapa y con múltiples máquinas en ambas etapas. Se analiza el concepto de dicotomía y las posibilidades de su aplicación en optimización combinatoria, específicamente en la programación de la producción. En el capítulo cinco se describe un algoritmo genético (AG) para el TFH con múltiples etapas, considerando las características: máquinas no relacionadas, tiempos de ajuste de máquinas dependientes de la secuencia y elegibilidad de máquinas. Este algoritmo se desarrolla para evaluar ciertas innovaciones implementadas. En el capítulo seis se propone un AG para el TFH con múltiples etapas, considerando características especiales del entorno de producción de una empresa de manufactura de televisiones en su sección de auto inserción. Se consideran máquinas no relacionadas, tiempos de ajuste de máquinas dependientes de la secuencia, elegibilidad de máquinas y búfer limitado. Se muestran los resultados de dos experimentos computacionales: calibración del algoritmo y evaluación de la calidad del

algoritmo en valores absolutos y en comparación con un algoritmo de referencia. Finalmente en el capítulo siete se presentan las conclusiones, aportaciones y se mencionan futuras líneas de trabajo.

Capítulo 2

Conceptos Preliminares

2.1 Notaciones

Los problemas de programación de la producción se describen a través de:

- el número de trabajos a procesar,
- la manera en que llegan las órdenes de fabricación a la planta productiva,
- el orden en que se utilizan las máquinas para realizar las operaciones,
- las restricciones tecnológicas sobre los trabajos,
- los objetivos de secuenciación.

Uno de los primeros sistemas de notación pertenece a Conway, Maxwell y Miller (1967). En este sistema se describe cada problema mediante cuatro campos A/B/C/D. El campo A, define los trabajos a realizar. Para problemas estáticos, especifica el número de trabajos cuya llegada se conoce exactamente (1,..., *n*). Para problemas dinámicos, identifica la distribución de probabilidad entre llegadas. El campo B, determina el número y tipo de máquinas que componen el taller (1,..., *m*). Cuando se consideran máquinas paralelas, el valor de este parámetro es el número de etapas y el número de máquinas de cada etapa es incluido en C. El campo C, describe el tipo de configuración del taller. El campo D, describe el criterio de eficiencia elegido, tal como: $F_{máx}$, $C_{máx}$, F_{med} , $T_{máx}$. Un ejemplo de esta notación sería: $n/m/J/C_{max}$ que indica un problema de secuenciación de un taller de trabajo (*Job shop*) con *n* trabajos y *m* máquinas en el que se pretende minimizar el tiempo máximo de ejecución de los trabajos.

Esta notación y sus complementos, no fue suficiente para representar toda la variedad de problemas, especialmente aquellos que se encuentran en la realidad. Más adelante hubo propuestas que conforman la notación que actualmente se usa. Estas propuestas fueron hechas por Graham, Lawler, Lenstra y Rinnooy Kan (1979), Blazewicz, Lenstra y Rinnooy Kan (1983) y adaptada para el caso de TFH por Vignier, Billaut y Proust (1999). Estos autores definen cualquier problema de programación de la producción con tres campos $\alpha|\beta|\gamma$, conocida como tripleta de Graham, donde:

- El primer campo α define el modelo y el número de máquinas.
- El segundo campo β representa características y restricciones de los trabajos.
- El tercer campo γ define el criterio de optimización.

De acuerdo con Baker (1974) y Pinedo (2002), en un problema de producción se tiene un conjunto *N* de trabajos, donde $N=\{1,2,...,n\}$, los cuales se procesan en un conjunto *m* de máquinas, $M=\{1,2,...,m\}$. Se utilizan los subíndices *j* y *k* para referirse a los trabajos y el subíndice *i* para las máquinas. De esta manera se definen los conceptos y parámetros descritos a continuación:

- Tarea u operación (O_{ij}). Denota la operación del trabajo *j* en la máquina *i*. Cada operación tiene un instante de inicio y finalización como resultado de la programación de la producción: $O_{ij}s$ y $O_{ij}f$.
- Tiempo de proceso (p_{ij}). Es el tiempo necesario para procesar el trabajo *j* en la máquina *i*. Después de la programación de la producción se cumple $p_{ij}=O_{ij}f-O_{ij}s$.
- Instante de entrada (r_j). Es conocido como "*release date*" o "*ready date*". Ninguna operación del trabajo comienza antes del instante de entrada.
- Instante de finalización (d_j). Es el instante o fecha de compromiso de finalización para el trabajo *j*. Se permite una finalización posterior, aunque con una penalización. En la literatura se conoce como "*due date*".
- Instante de finalización obligada ($\bar{d}$). En este caso es un compromiso firme de finalización de los trabajos. Conocido como "*deadline*".

- Tiempo de ajuste no dependiente de la secuencia (S_{ij}). Es el tiempo de preparación de la máquina *i* para procesar el trabajo *j*.
- Tiempo de ajuste dependiente de la secuencia (S_{ijk}). Para cada máquina *i* se define una matriz S_{jk} con número de filas y columnas igual a *n*. El elemento S_{jk} de la matriz *i*, *i* = 1,…, *m*, representa el tiempo de ajuste de la máquina *i* antes de procesar el trabajo *j* después del trabajo *k* procesado previamente.
- Prioridades (w_j). La prioridad o peso de un trabajo es una manera de establecer una jerarquía relativa de los trabajos en *N*. La prioridad se establece en función del costo del trabajo, el tipo de cliente, etc.

En ciertas circunstancias se permite variar las notaciones de acuerdo a las necesidades específicas del problema.

El campo α, define el entorno de producción o la configuración de las máquinas, y se divide en $\alpha_1\alpha_2$, donde:

α_1 = 1 : una sola máquina.

= *P* : máquinas paralelas idénticas.

= Q : máquinas paralelas proporcionales o uniformes. Se proporciona, para cada máquina *i* un coeficiente de velocidad v_i, donde $p_{ij} = p_j / v_i$.

= *R* : máquinas no relacionadas. Además de tener velocidades diferentes, el tiempo de ejecución depende de la tarea.

= *O* : taller abierto u "*Open Shop*".

= *F* : taller de flujo o "*Flow Shop*".

= *J* : taller de trabajo o "*Job Shop*".

α_2 = $M = \{1, 2, ..., m\}$: número de máquinas.

= $\varnothing$: el número de máquinas depende de la instancia a resolver.

El campo β indica las características atribuibles a los trabajos. Estas son numerosas y normalmente los autores dividen este campo en varios subcampos:

β_1 = *pmtn* : se permite la interrupción de operaciones o "*preemption*". Indica que una operación O_{ij} , una vez comenzada en una máquina no es necesario que se complete totalmente; tiene capacidad de ser interrumpida. De producirse la interrupción, se permite introducir otra tarea diferente en la máquina. El tiempo ya procesado para la operación no se pierde, el tiempo restante se procesa en la misma o en otra máquina. También se denota como *prmp*.

= $\varnothing$: no se permite la interrupción de operaciones.

β_2 = *prec* : relaciones de precedencia. Con este campo se especifica que existen relaciones de precedencia o "*precedence constraints*" entre los trabajos. Si un trabajo tiene un predecesor, no comienza hasta que éste haya terminado. Existen distintos tipos de relaciones de precedencia: *chains*, *intree*, *outtree*, *tree*.

= $\varnothing$: no existen relaciones de precedencia.

β_3 = r_j : existen instantes o fechas de entrada (*release date*).

= $\varnothing$: no existen instantes o fechas de entrada.

β_4 = S_{nsd} : existen tiempos de ajuste de las máquinas (*no sequence dependent setup times*) por cambios de tipo de trabajo que no dependen de la secuencia de los trabajos. Esto es, es necesario hacer ajustes en la máquina antes de procesar el trabajo y estos ajustes no dependen del trabajo procesado anteriormente en la misma máquina. Normalmente los ajustes son "anticipativos", es decir, los ajustes en la máquina se hacen antes de que el trabajo llegue a la máquina.

= S_{sd} : existen tiempos de ajuste de las máquinas por cambios de tipo de trabajo, los cuales dependen de la secuencia (*sequence dependent setup times*).

= $\varnothing$: los tiempos de ajuste de las máquinas por cambios de tipo de trabajo no existen o son muy pequeños y se incluyen en los tiempos de procesamiento.

β_5 = d_j: existen instantes o fechas de finalización (*due dates*). Son independientes para cada trabajo.

= $d_j = d$: los instantes o fechas de finalización son homogéneas para todos los trabajos (*common due dates*).

= $\bar{d}_j$: los instantes o fechas de finalización son de cumplimiento obligado (*deadlines*). En algunas notaciones esto se representa como Δ_j.

= $\varnothing$: no existen fechas de entrega.

β_6 = *prmu* : sólo aplicable cuando $\alpha_1 = F$. Indica que el orden de entrada de los trabajos es el mismo para todas las máquinas (la secuencia es idéntica para todas las máquinas del taller). En este caso existen *n*! posibles secuencias.

= $\varnothing$: indica que la secuencia de entrada en cada máquina varía.

β_7 = *brkdwn* : las máquinas están sujetas a averías o a períodos donde no procesan trabajos (*breakdowns*). Cuando después de una avería el trabajo se reanuda, se dice que es *resumable* (*r*), de lo contrario *non resumable* (*nr*).

= $\varnothing$: las máquinas están disponibles en todo momento.

β_8 = *nwt* : sin esperas o "*no-wait*". Esta condición impone la restricción de que un trabajo no permite esperar entre dos máquinas.

= $\varnothing$: se permite que los trabajos esperen por tiempo indefinido entre las máquinas.

β_9 = *block* : bloqueo (*blocking*). Indica que existe una capacidad finita entre las máquinas para almacenar el producto en curso (*buffer*). El caso extremo se da cuando no hay buffer entre las máquinas (*zero buffers*).

= $\varnothing$: existe un buffer ilimitado entre las máquinas y por tanto ningún trabajo o máquina queda bloqueado.

β_{10} = *recrc* : recirculación o "*recirculation*". Se da cuando α_1 = *O, F, J*. Indica que un trabajo (o todos en general) necesita ser procesado más de una vez en alguna o todas las máquinas.

= $\varnothing$: los trabajos sólo se procesan una vez en cada máquina.

β_{11} = M_j : restricción en el uso de máquinas (*machine eligibility constraints*). No todas las *m* máquinas son capaces de procesar el trabajo *j*. Se presenta para $\alpha_1 = P$ o en problemas mixtos más complejos.

= $\varnothing$: las máquinas siempre están disponibles para todos los trabajos.

β_{12} = $p_{ij} = 1$: todos los tiempos de procesamiento de los trabajos son iguales a 1.

= $p_{ij} = p$: todos los tiempos de procesamiento de los trabajos son iguales a una constante *p*.

= $\varnothing$: los tiempos de procesamiento son valores no negativos arbitrarios.

β_{13} = $size_{ij}$: cualquier tarea de un trabajo *j* requiere de un número $size_{ij}$ de máquinas para su procesamiento simultáneo en la etapa *i* (Oguz, et al., 2004).

= $\varnothing$: no existe esta característica.

β_{14} = $wait_{ij}$: el tiempo de espera es limitado para cada trabajo *j* en una etapa *i*. (Ling-Huey, 2003)

= $\varnothing$: no existe esta característica.

Antes de pasar al campo γ, es necesario listar algunas medidas de optimización:

a_j : período permitido para el procesamiento del trabajo *j* (*allowance*). Es el lapso de tiempo entre el instante de entrada y finalización (*deadline*); $a_j = d_j - r_j$.

W_{ij} : es el tiempo que el trabajo *j* espera antes de empezar la operación correspondiente en la máquina *i*. A partir de la definición de las operaciones se tiene que $W_{ij} = O_{ij}s - O_{i-1,j}f$. En un entorno *no-wait*, $W_{ij} = 0$.

W_j : tiempo total de espera del trabajo *j*. Se calcula a partir de la fórmula $W_j = \sum_{i=1}^{m} W_{ij}$.

C_j : instante o fecha en que finaliza el procesamiento del trabajo *j* en el taller (*completion time*). A partir de los parámetros anteriores se calcula como $C_j = r_j + \sum_{i=1}^{m} W_{ij} + p_{ij}$ (siempre que no existan tiempos de ajuste por cambio de tipo de trabajo).

$C_{\max}$: máximo instante o fecha de finalización. $C_{\max} = \max\{C_1, C_2, ..., C_n\}$.

I_i : tiempo ocioso (*idle time*) de la máquina *i*. Se refiere al tiempo que pasa la máquina *i* sin procesar trabajos. Se calcula como $I_i = C_{\max} - \sum_{j=1}^{n} p_{ij}$.

F_j : tiempo de flujo (*flow time*). Es la cantidad de tiempo que un trabajo *j* permanece en el taller. $F_j = C_j - r_j$.

L_j : holgura del trabajo frente a su instante o fecha de finalización (*deadline*). $L_j = C_j - d_j$, llamado también "*lateness*". Si $L_j = 0$, $\forall j = \overline{1,n}$, se dice que el trabajo termina a tiempo (*on time*); si $L_j < 0$, el trabajo ha terminado antes de lo previsto (*early*); si $L_j > 0$, entonces el trabajo termina después del instante o fecha máxima y se dice que se ha retrasado (*tardy*).

T_j : retraso (*tardiness*) de un trabajo *j*. Se calcula como $T_j = \max\{L_j, 0\}$.

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

El campo γ indica el criterio de optimización:

γ = $C_{\max}$: minimización del tiempo máximo de finalización (*makespan*). Este criterio de optimización es el más utilizado en la literatura.

= $\bar{C}$: minimización del tiempo medio de finalización. Se calcula como $\bar{C} = \frac{1}{n} \cdot \sum_{j=1}^{n} C_j$.

= $F_{\max}$: minimización del máximo tiempo de flujo, $F_{\max} = \max\{F_1, F_2, ..., F_n\}$. Si no existen instantes de entrada, esto es, si $r_j = 0, \forall j \in N$, entonces $F_{\max} = C_{\max}$.

= $\bar{F}$: minimización del tiempo medio de flujo, $\bar{F} = \frac{1}{n} \cdot \sum_{j=1}^{n} F_j$.

= L_{max} : minimización de la máxima holgura, $L_{\max} = \max\{L_1, L_2, ..., L_n\}$. Con esto se busca minimizar el máximo incumplimiento del instante de finalización.

= $\bar{L}$: minimización de la holgura media. $\bar{L} = \frac{1}{n} \cdot \sum_{j=1}^{n} L_j$

= T_{max}: minimización del retraso máximo. $T_{\max} = \max\{T_1, T_2, ..., T_n\}$. Se busca minimizar el máximo retraso con respecto al instante de finalización.

= $\bar{T}$: minimización del retraso medio. $\bar{T} = \frac{1}{n} \cdot \sum_{j=1}^{n} T_j$.

= E_{max} : minimización del adelanto máximo. $E_{\max} = \max\{E_1, E_2, ..., E_n\}$.

= $\bar{E}$: minimización del adelanto medio. $\bar{E} = \frac{1}{n} \cdot \sum_{j=1}^{n} E_j$.

= N_T : minimización del número de trabajos retrasados. Se calcula a partir de la expresión: $N_T = \sum_{j=1}^{n} U_j$, donde

$$U_j = \begin{cases} 1, \text{ si } C_j > d_j \\ 0, \text{ en caso contrario} \end{cases}.$$

2.2 Clasificación de problemas en la programación de la producción

Los problemas en la programación de la producción se clasifican de la siguiente manera (Portman, 1997):

1. (1) Una máquina (*single machine*).
   Cada trabajo tiene una sola operación. No existe el problema de asignación pero si el de secuenciación.
2. Máquinas paralelas
   Más de una máquina. Cada trabajo tiene una sola operación. Si el número de máquinas es superior al número de trabajos no existe problema de secuenciación

pero si el de asignación. En el caso general se tienen que asignar y secuenciar las máquinas.

Las máquinas paralelas se dividen en:

a) (*Pm*) Máquinas idénticas (*identical parallel machine*).- Procesan los trabajos a la misma velocidad.

b) (*Qm*) Máquinas Proporcionales o Uniformes (*Uniform parallel machine*).- Tienen velocidades diferentes y se miden con un factor de proporción.

c) (*Rm*) Máquinas no relacionadas (*unrelated parallel machine*).- La velocidad de procesamiento depende de la máquina y la tarea que se está procesando. Es un caso general que engloba a los dos anteriores.

3. Problemas de tipo taller (*Shop*).

Se dispone de varias máquinas; cada trabajo tiene un número de operaciones, hay un número de máquinas. Existe una relación de precedencia entre las tareas para cada trabajo, a esta relación se le denomina “ruta” y de acuerdo a las características de las rutas, son:

a) (*Jm*) Taller de Trabajo (*Job Shop*).- Cada trabajo tiene su propia ruta a seguir en las *m* máquinas.

b) (*Om*) Taller Abierto (*Open Shop*).- No existe ninguna restricción en la ruta. Las operaciones se realizan en cualquier orden.

c) (*Fm*) Taller de flujo (simple) (*Flow Shop*).- Todos los trabajos tienen la misma ruta en las *m* máquinas. Existe una ordenación predeterminada de las tareas que es la misma en todos los trabajos.

El taller de flujo se extiende en:

d) (*FFm*) Taller de flujo flexible (*Flexible flowshop*).- Es una generalización del taller de flujo simple. Consiste en *m* etapas en serie. Las etapas contienen máquinas paralelas idénticas. Un trabajo requiere pasar por todas las etapas sucesivamente y por cualquier máquina (una sola) de cada etapa.

e) (*HFm*) Taller de flujo híbrido (*Hybrid flowshop*).- Es una generalización del taller de flujo flexible (TFF). Las máquinas paralelas en al menos una etapa no son idénticas.

Ejemplo: La expresión $HF3,((PM^{(i)})_{i=1}^{(3)}) \mid prec \mid C_{\max}$ indica un TFH de tres etapas con máquinas paralelas en cada etapa, restricciones de precedencia y considera como criterio de optimización la minimización del tiempo máximo de finalización.

La Figura 2.1 muestra una clasificación de problemas en la programación de la producción.

2.3 Talleres de Flujo

En un taller de flujo se procesa un conjunto *N* de trabajos, $N = \{1, 2, ..., n\}$ en un conjunto *M* de *m* máquinas en serie, $M = \{1, 2, ..., m\}$. Cada trabajo pasa por todas las máquinas; la secuencia de proceso es la misma para todos los trabajos. Es decir, existe un "flujo" común de los trabajos en las máquinas. (Figura 2.2)

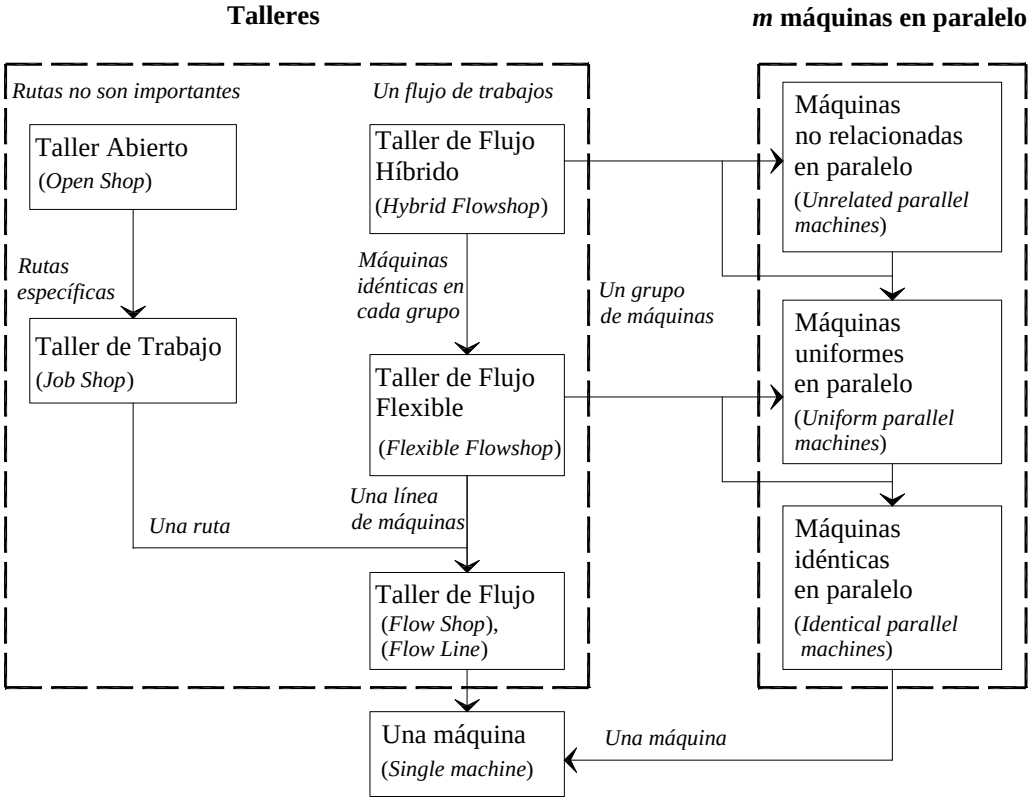


Figura 2.1 Clasificación de problemas en la programación de la producción

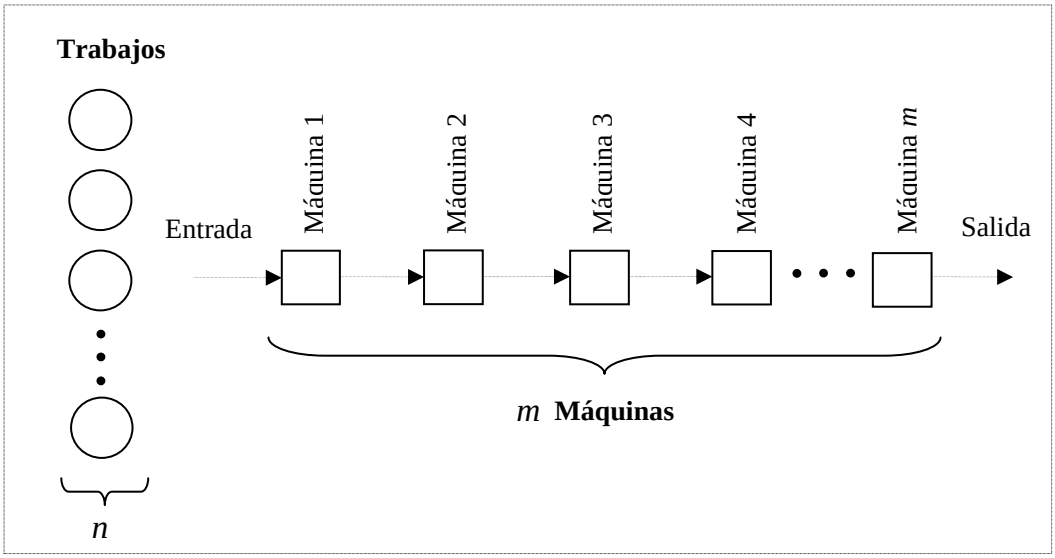


Figura 2.2 Taller de flujo

El flujo de los trabajos en las máquinas implica una relación de precedencia en las operaciones, es decir, para cualquier trabajo *j*, el orden de procesamiento de sus *m* operaciones es el siguiente: $O_{1j} \prec O_{2j} \prec ... \prec O_{mj}$. Respecto a los tiempos de inicio y finalización, la última expresión significa que

$$\forall j \in N,\ O_{i-1,j} f \leq O_{ij} s,\ i = (2,...,m),$$

donde $O_{i-1,j}\ f$ es el instante en que finaliza el trabajo *j* en la máquina *i*-1 y $O_{ij} s$ es el instante de inicio del trabajo *j* en la máquina *i*.

El criterio de optimización más utilizado en la literatura para este problema es la minimización del tiempo máximo de finalización de un conjunto determinado de trabajos ($C_{\max}$). Este objetivo consigue la más eficiente utilización de las máquinas, al tiempo que permite una entrega más rápida del producto finalizado. Este criterio es el que usaremos para el objetivo de esta tesis.

El taller de flujo con la minimización de $C_{\max}$ como objetivo se denota por $F \| C_{\max}$ de acuerdo a la notación de Graham et al. (1979). Otros autores, como Pinedo (2002), utilizan una notación ligeramente distinta y denotan el problema como $Fm \| C_{\max}$.

Uno de los primeros trabajos para este problema fue realizado por S.M. Johnson (1954) quien encontró una solución óptima para el taller de flujo cuando se trata de dos máquinas.

Una generalización del taller de flujo es el TFF, también conocido como taller de flujo con múltiples procesadores (*Flowshop with multiple processors*, FSMP). En el TFF se procesa un conjunto N de trabajos, $N = \{1, 2, ..., n\}$, en un conjunto M de etapas, $M = \{1, 2, ..., m\}$ y en cada etapa $i \in M$, un conjunto $M_i = \{1, ..., m_i\}$ de máquinas paralelas idénticas procesan los trabajos. Un trabajo *j* se procesa en todas las *m* etapas y dentro de cada etapa *i* en cualquiera de las m_i máquinas idénticas.

Una generalización del TFF, es el TFH. El modelo del TFH es similar al TFF, siendo la única diferencia que las máquinas paralelas en al menos una de *m* etapas no son idénticas (Figura 2.3).

Las investigaciones del TFH se enfocan generalmente en:

1) TFH con dos etapas.
2) TFH con tres etapas.
3) TFH con *k* etapas.

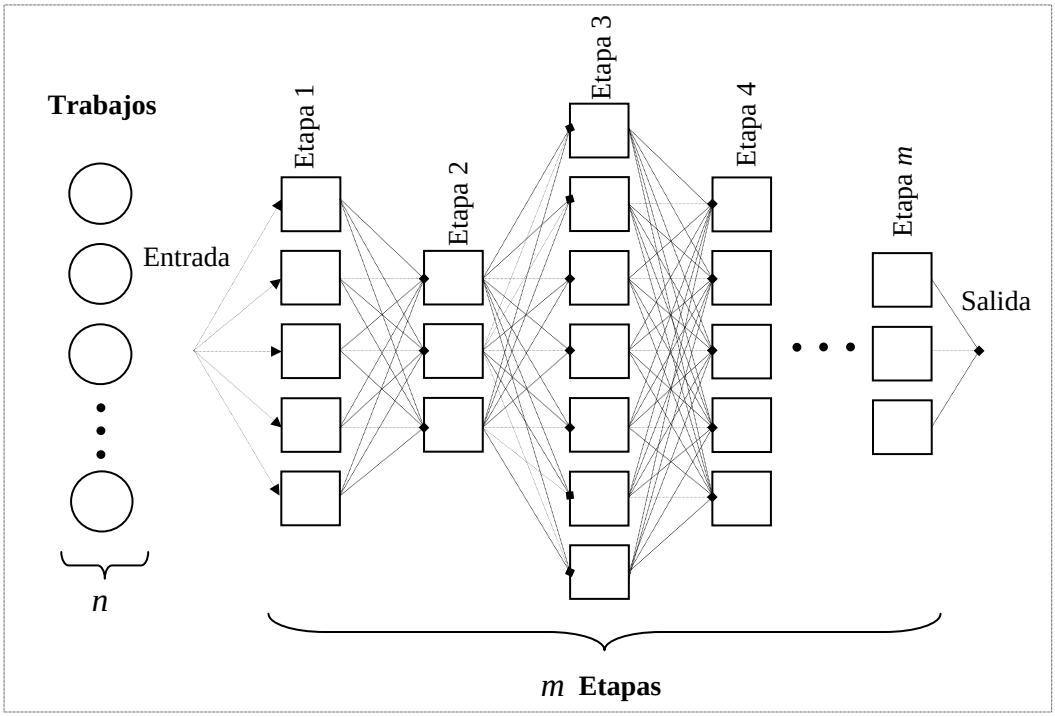


Figura 2.3 Taller de flujo híbrido

2.4 Complejidad Computacional

En la teoría de complejidad se estudia la dificultad computacional de los problemas. Un algoritmo es más eficiente cuantas menos operaciones de cálculo requiere. La eficiencia suele medirse en términos de consumo de recursos: 1) cantidad de memoria que un algoritmo consume o utiliza durante su ejecución (complejidad espacial), 2) tiempo que necesita el algoritmo para ejecutarse (complejidad temporal).

Un problema que se resuelve utilizando un algoritmo de complejidad polinomial en el peor caso es tratable. Sin embargo, si el polinomio tiene un grado alto (como, por ejemplo, 100) o si los coeficientes son extremadamente grandes, el algoritmo necesita un tiempo muy grande para solucionar el problema. Un problema el cual no es posible resolver utilizando un algoritmo de complejidad polinomial en el peor caso es intratable.

Los problemas para los cuales se demuestra que no existen algoritmos que los resuelvan son irresolubles. Por ejemplo, el problema de la parada demostrada por Alan Turing. Este problema toma como entrada un programa junto con la entrada de este programa. El problema pregunta si el programa se interrumpirá cuando se ejecute con la entrada del programa (Kelley, 1995).

Los problemas tratables pertenecen a la complejidad de clase **P** (*complexity class*). No todos los problemas se resuelven dentro de un tiempo polinomial. Para muchos existe un algoritmo de resolución cuya duración tiene una función exponencial. Tales algoritmos son inaceptables. Muchos problemas resolubles tienen la propiedad de que no hay ningún algoritmo con complejidad polinomial en el peor caso que los resuelva, pero en caso de conocerse alguna solución se comprueba que ésta, resuelve el problema en tiempo polinomial (procedimiento inverso).

La clase **NP** (*Non-deterministic polynomial time*) está conformada por todos aquellos problemas para los que sólo se han encontrado métodos o algoritmos de complejidad exponencial. Es la clase formada por aquellos problemas, para los que no se ha encontrado un algoritmo de complejidad polinomial, pero existe un método para verificar una solución obtenida en tiempo polinomial. Esta clase contiene todos los problemas que se verifican en tiempo polinomial. Según la definición, $\mathbf{P} \subseteq \mathbf{NP}$.

En la teoría de la complejidad se investigan los llamados problemas de decisión (*decision problems*) cuya respuesta es “si” o “no” para cualquier entrada. Si el problema a resolver es de optimización (*optimization problems*), donde se requiere minimizar o maximizar un valor de alguna magnitud, primero es necesario transformarlo durante un tiempo polinomial en algún conocido problema de decisión.

Reducir un problema de decisión *p* en un problema *q* significa exhibir una función *f* polinomial tal que para todo *w*: $p(w) = q(f(w))$. El procedimiento de reducción polinomial de *p* en *q*, se denota por $p \propto q$. Si un algoritmo resuelve *q*, entonces resuelve *p* (pero la reciproca no es necesariamente verdadera). Entonces los problemas reducidos son más “difíciles”. Un problema de decisión *p* es **NP**−Difícil (**NP**-*Hard*) si todo problema *q* de NP se reduce polinomialmente en *p*. Además, si el problema *p* en que se reducen los problemas de NP es el mismo dentro de NP, se dice que *p* es **NP**−Completo (**NP**-*Complete*)

Básicamente, un problema es **NP**-Completo (NPC) si:

1) Pertenece a **NP**.

2) Es tan difícil como cualquier otro problema en **NP** (NP-Difícil).

Para esta clase de problemas no se han encontrado algoritmos polinomiales, sin embargo no está comprobado que estos algoritmos no existen.

La propiedad principal de los problemas **NP**-Completos es que si al menos un problema de la clase **NP**/**P** tiene solución polinomial, entonces todo problema de clase **NP** también se resolvería en tiempo polinomial. La investigación de los problemas **NP**-Completos está relacionada con la pregunta **P** ≠ **NP** (Garey & Johnson, 1979). Esta pregunta fue formulada en 1971 y sigue abierta en la actualidad. Es generalmente aceptado, aunque no se ha demostrado, que ningún problema **NP**-Completo se resuelve en un tiempo polinomial.

De manera intuitiva se representa la clase **P** como una clase de problemas que se resuelve rápidamente, y la clase **NP** como una clase de problemas que se revisa rápidamente. La mayoría de los problemas de clase **NP** son tratables o su complejidad **NP** está comprobada. Sin embargo existen problemas cuyo estatus sigue indefinido. La determinación del estatus de un problema discreto calculatorio es importante tanto para la teoría, como para la práctica. La comprobación de la complejidad **NP** de un problema permite dejar de buscar algoritmos eficientes para encontrar su solución exacta, y

concentrar la atención en el desarrollo de procedimientos aproximados con una exactitud aceptable. Los problemas de optimización combinatoria son en general **NP**-Difícil. Por tanto se trabaja en buscar la mejor solución en el menor tiempo posible. Algunos problemas NP-Difícil comunes en la programación de la producción para su reducción, son: Problema del agente viajero (*travelling salesman problem*), satisfactibilidad booleana (*boolean satisfiability*), partición (*partition*), circuito de Hamilton (*hamiltonian path*) (Morton & Pentico, 1993; Gupta, 1988; Leung, 2004).

2.5 Métodos de la programación de la producción

Los métodos que resuelven los problemas de la programación de la producción se dividen en: 1) métodos exactos que aseguran una solución óptima, y 2) métodos heurísticos que no garantizan el óptimo pero proporcionan una buena solución. La Figura 2.4 muestra una clasificación de estos métodos.

2.5.1 Métodos exactos

Los métodos que buscan una solución óptima, se clasifican en eficientes y de enumeración.

Los primeros son aplicables a un grupo de problemas relativamente reducido; principalmente problemas de una máquina, problemas de taller de flujo con un número pequeño de máquinas, o problemas con restricciones en las duraciones de las operaciones. Un ejemplo de un algoritmo eficiente es la Regla de Johnson que encuentra una solución óptima para un taller de flujo con dos máquinas (Johnson, 1954):

La regla de Johnson determina que un trabajo *j* precede en la secuencia a un trabajo *k* si: $\min\{p_{1j}, p_{2k}\} < \min\{p_{1k}, p_{2j}\}$. De esta manera, los trabajos con un tiempo de proceso corto en la primera máquina van primero en la secuencia y los trabajos con un tiempo corto en la segunda máquina se procesan al final para así evitar tiempos muertos en la programación. El algoritmo de la regla de Johnson es el siguiente:

1. Hacer *Ini* = 1, *Fin* = *n*. Crear una lista *l* de trabajos ya secuenciados y una lista de trabajos no secuenciados *l'*. Inicializar $l = 0$ y $l' = \{1,...,n\}$

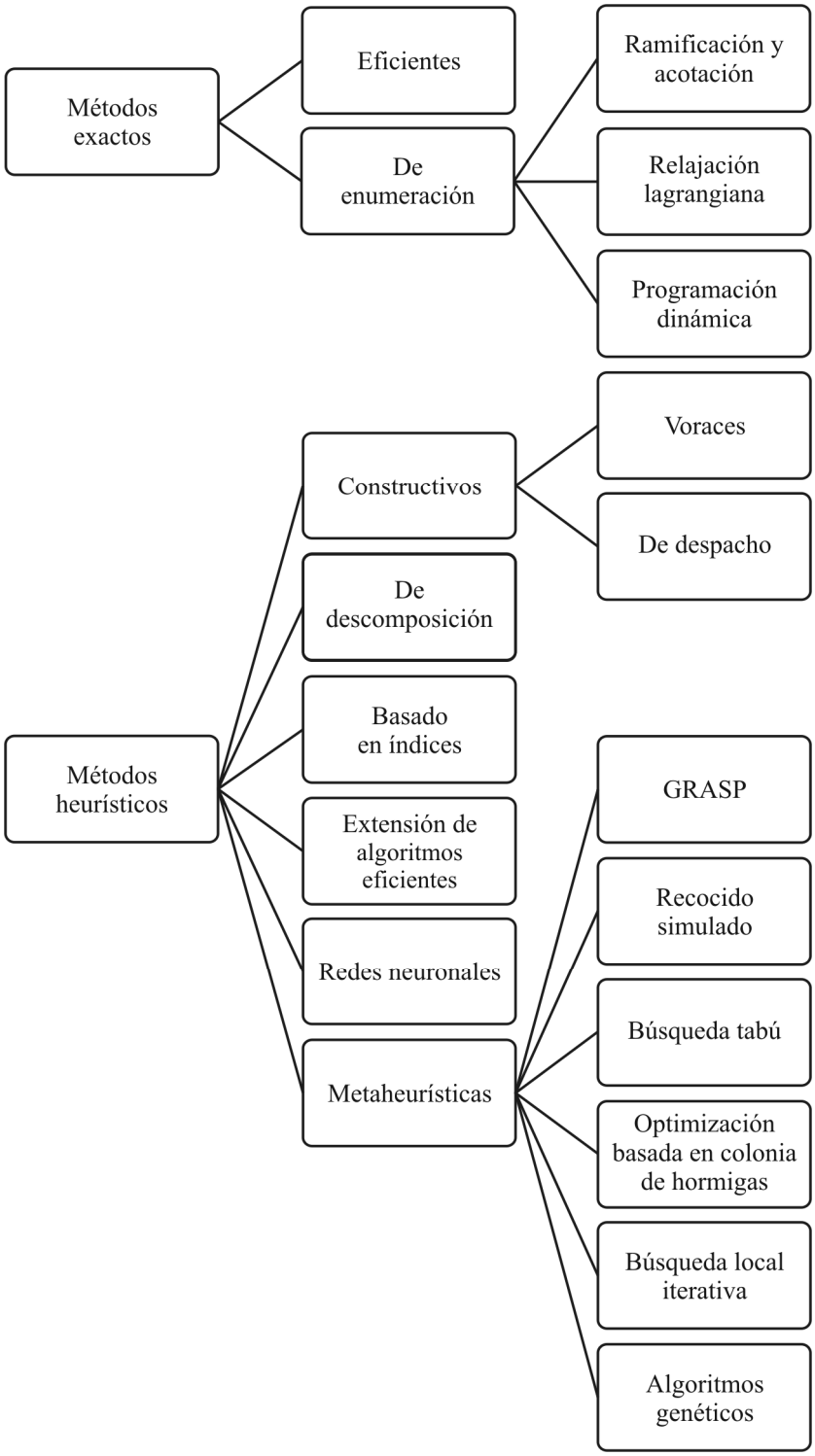


Figura 2.4 Clasificación de métodos de la programación de la producción

2. Para todos los trabajos de *l'*, buscar el trabajo con el mínimo tiempo de proceso en la máquina 1 y el trabajo con el mínimo tiempo de proceso en la máquina 2. Hacer $\min_1 = \min_{j \in l}(p_{1j})$, $h_1 = l'(j)$ y $\min_2 = \min_{k \in l}\{p_{2k}\}$, $h_2 = l'(k)$.
3. Si $\min_1 < \min_2$ entonces ir al paso 4. $\min_1 > \min_2$ ir al paso 5. Si $\min_1 = \min_2$ ir aleatoriamente al paso 4 o al 5.
4. Insertar el trabajo h_1 en la posición *Ini* de *l*, $l(Ini) = h_1$. Hacer $Ini = Ini + 1$. Ir al paso 6.
5. Insertar el trabajo h_2 en la posición *Fin* de *l*, $l(Fin) = h_2$. Hacer $Fin = Fin - 1$. Ir al paso 7.
6. Eliminar h_1 de l'. Si $l' = 0$ entonces fin, de lo contrario ir al paso 2.
7. Eliminar h_2 de l'. Si $l' = 0$ entonces fin, de lo contrario ir al paso 2.

Los métodos exactos de enumeración realizan búsquedas exhaustivas para encontrar una solución óptima. A cada problema se asocia un árbol que represente el espacio de estados. Cada nodo de este árbol es un estado del problema. El camino desde la raíz hasta este nodo define una tupla que pertenece al conjunto de soluciones. Estos nodos son de dos tipos: nodos vivos, cuando todavía no se han generado todos sus descendientes, y nodos muertos, en caso contrario.

Los nodos de este árbol de estados se generan en dos formas distintas, lo que da lugar a dos técnicas: Búsqueda con retroceso (*backtracking*) y ramificación y acotación (*branch and bound*).

En los algoritmos de búsqueda con retroceso, los nodos del árbol del espacio de estados se generan de la siguiente forma:

Si *R* es un nodo actual (el que se está extendiendo en un determinado momento), cada vez que se genere un nuevo hijo suyo *C*. El último se convierte en el nodo actual y únicamente vuelve a ser nodo actual *R* cuando el subárbol generado por *C* haya sido completamente explorado.

Como la búsqueda con retroceso, el método de ramificación y acotación es una técnica de exploración que en la mayor parte de las ocasiones busca una solución óptima a

un problema. El sistema para generar nodos en el árbol de estados posteriores es el siguiente:

Se generan los hijos de un nodo *R* de manera que este nodo sigue en estado actual hasta que muere. A continuación, en cada nodo se calcula una cota de los valores de las soluciones situadas por debajo. Si esta cota demuestra que tales soluciones son eventualmente peores que una solución ya encontrada, se abandona la exploración de esa parte del árbol.

El cálculo de la cota se combina con un recorrido a lo ancho (*breadth-first search*) o en profundidad (*depth-first search*) que sirve para eliminar ciertas ramas del árbol (Cormen, et al., 1990). Además, esta cota se utiliza para escoger la rama más prometedora con el fin de realizar una exploración con prioridad.

Este método realiza cálculos adicionales para decidir en cada momento, cuál es el siguiente nodo a explorar, utilizando una lista para conservar los nodos generados pero aún no examinados.

En la programación de la producción su aplicación está limitada a sistemas de poca dimensión; Carlier y Rebaï (1996) publicaron dos algoritmos tipo ramificación y acotación para el taller de flujo permutacional. Aunque estos algoritmos contienen numerosas aportaciones, no resuelven problemas de más de nueve trabajos en un tiempo razonable.

La eficiencia de ambos métodos depende del tiempo necesario para generar nuevos nodos, del número de soluciones que existan y del tiempo empleado para realizar las acotaciones. En algunas ocasiones, cuando el árbol de estados es grande, se hace imprescindible manejar una buena función de acotación para obtener la solución en un tiempo aceptable.

Otro método de búsqueda enumerativa es la programación dinámica. Sus fundamentos son elaborados por Richard Bellman (Bellman, 1957). Representa una optimización recursiva, con etapas múltiples. Se interpreta como un proceso multietapa de toma de decisiones.

La técnica llamada “Divide y vencerás” consiste en descomponer el problema a resolver en un cierto número de subproblemas más pequeños, resolverlos independientemente (utilizando en numerosas ocasiones técnicas recursivas) y finalmente

recoger los resultados obtenidos en dirección “de abajo hacia arriba” para construir la solución del problema inicial (Wirth, 1976).

La programación dinámica es aplicable en casos, cuando los subproblemas no son independientes, es decir, cuando los subroblemas tienen subsubproblemas en común. En este caso, el algoritmo del tipo “Divide y vencerás” realiza un trabajo repetitivo, resolviendo los mismos subproblemas varias veces, mientras que un algoritmo basado en la programación dinámica, resuelve cada subproblema una sola vez e inscribe las respuestas en una tabla especial.

Relajación lagrangiana (l*agrangian relaxation*) (Shapiro, 1979; Tang, Luh, Liu & Fang, 2002) se basa en modelos de Programación Lineal Mixta para problemas de programación de la producción. El objetivo es transformar una o varias restricciones en términos de la función objetivo utilizando coeficientes de penalización proporcionales al grado de violación de cada restricción. Una vez realizada la simplificación se intenta resolver el modelo entero mediante procedimientos de exploración dirigida.

El tiempo de ejecución de un algoritmo, el cual aplica un método de los descritos anteriormente, es exponencial, pero, si cada subproblema se resuelve una única vez almacenando el resultado en una tabla de forma que esté accesible siempre que sea necesario, el tiempo de ejecución resulta polinomial. Tales algoritmos resuelven el nombre de pseudopolinomiales (Rosen, 1999). 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. En otras ocasiones, no es posible tomar una secuencia de decisiones que nos lleven al resultado óptimo.

Consiste en la obtención de límites inferiores de subconjuntos de soluciones admisibles, que reflejen valores por encima de los cuales se encuentran todas las soluciones de dicho subconjunto (en un problema de minimización). La aplicación completa de este método proporciona una solución óptima aunque algunas veces se trunca la búsqueda antes de finalizar la exploración (en este caso el procedimiento se vuelve heurístico).

A partir de la teoría de la complejidad, se llega a la conclusión que para la mayoría de los problemas de Programación de la Producción, es muy improbable encontrar procedimientos exactos de resolución. Sin embargo la necesidad de ofrecer resultados aunque sea aproximados, hacen aparecer los procedimientos llamados heurísticos; esto es,

algoritmos que proporcionan soluciones factibles aunque no necesariamente óptimos, a problemas difíciles, en un tiempo de cálculo razonable (Zanakis & Evans, 1981).

2.5.2 Métodos heurísticos

a) Constructivos

Consisten en ir añadiendo componentes individuales a la solución, hasta que se obtiene una solución factible (Silver, et al., 1980). Los algoritmos basados en métodos constructivos, se dividen en:

1) voraces (*greedy*). Son procedimientos constructivos que utilizan una función de "vista corta" que añade un componente a la solución, de tal manera que se obtenga el máximo beneficio en cada paso (El-Zahar & Rival, 1985). El algoritmo más famoso de tipo voraz que se aplica a la resolución del problema del agente viajero es el algoritmo de Dijkstra (Dréo, et al., 2007).

2) de despacho (*dispatching*). Simulan el comportamiento del sistema en tiempo real. Las decisiones se toman cada vez que una máquina finaliza un trabajo. Para ello se usan reglas de prioridad que proporcionan pautas para establecer la secuencia de los trabajos (Giffer & Thompson, 1960). Las reglas de prioridad son estáticas, cuando el valor de un parámetro sobre el que se basa la decisión de secuenciación (tiempo de procesamiento, instante de entrada, etc.) es fijo a lo largo del problema, o dinámica cuando su valor varía. Además se clasifican en locales y globales. Algunas de estas reglas de prioridad son: FIFO (*First In First Out*), donde el primer trabajo en llegar será procesado en primer lugar; SPT (*Short Processing Time first*), los trabajos más cortos se realizan primero; EDD (*Earliest Due Date first*), el trabajo con tiempo de entrega más temprano se elige en primer lugar; LPT (*Largest Processing Time first*), los trabajos más largos se eligen primero. (Coffman & Sethi, 1976).

Tradicionalmente la toma de decisiones se hacía por personas que sistemáticamente iban aplicando reglas según su experiencia para resolver el problema. Con aparición de las computadoras, es posible proporcionar una solución a un costo relativamente bajo. Las reglas de despacho se realizan a través de programas de simulación o software de secuenciación. Estos procedimientos por su propia dinámica trabajan hacia delante

(*forward scheduling*) a partir del instante inicial. En contraposición, también existen procedimientos de programación hacia atrás (*backward scheduling*) donde se secuencia tomando como base los tiempos de entrega de los trabajos. En este sentido, a esta familia pertenecen los procedimientos basados en cuello de botella (*bottleneck*), donde la distribución del tiempo extra afecta fuertemente la calidad de la solución (Leung, 2004).

b) De descomposición

Se trata de dividir el problema en subproblemas más pequeños, siendo la solución de uno, la entrada del siguiente, de forma que al resolverlos todos, se obtenga una solución para el problema global. Un ejemplo de este tipo de métodos, es el de cuello de botella mencionado anteriormente. Por cuello de botella se entiende un factor crítico del sistema, a partir del cual se inicia la secuenciación y se utiliza esta información para secuenciar en el resto de los recursos (Adams, et al., 1988).

c) Basado en índices

Se aplica un procedimiento que determina la secuencia de los trabajos en cada máquina del sistema, mediante un procedimiento sencillo que no exige una simulación de la secuencia (Palmer, 1965).

d) Aplicación limitada de algoritmos eficientes

Un ejemplo de este enfoque es la heurística CDS de Campbell et al. (1970), que extiende el algoritmo eficiente de Johnson (1954) a casos más generales aunque la solución óptima no esté asegurada.

e) Redes neuronales

Son estructuras de procesamiento distribuido, paralelo y adaptativo, los cuales se inspiran en la estructura cerebral humana (Martin & Sanz, 1997). La idea básica es que a partir del entrenamiento de la red con diferentes problemas de ejemplo basados en casos conocidos del área, la red sea capaz de extrapolar los resultados a otros problemas diferentes que se presentarían en el futuro. En Zhou et al. (1991) se encuentra una aplicación de redes neuronales a un problema simple de taller de trabajo en la programación de la producción.

e) Metaheurísticas

Incluyen todos aquellos procedimientos que en un proceso iterativo, guían a una heurística subordinada, combinando inteligentemente diferentes conceptos tomados de la analogía de la naturaleza y explorando el espacio de soluciones utilizando estrategias de aprendizaje para estructurar la información, con el objetivo de encontrar soluciones cercanas al óptimo (Osman, 1995). Estos métodos parten de una solución factible inicial y mediante iteraciones de esa solución, van generándose iterativamente otras soluciones, almacenando como óptima la mejor de las soluciones generadas hasta que se cumple un criterio de paro. Dentro de las metaheurísticas se encuentran:

1) Algoritmos GRASP (*Greedy Randomized Adaptive Search Procedure*). Son procedimientos voraces aleatorizados y adaptativos en la búsqueda de soluciones, desarrollados por Feo y Resende (1989). Generan una solución al problema en dos fases. Inicialmente, de una forma iterativa se construye una solución y con cierto grado de aleatoriedad, mediante una heurística de vista corta (se tienen en cuenta aspectos muy limitados del estado actual de la solución), para después mejorarla mediante un postprocesado de optimización local. Usualmente se intenta obtener un equilibrio entre la fase construcción y la de mejora. Esto es, algoritmos con una fase inicial que ofrezcan soluciones de alta calidad, no necesitan segunda fase de mejora muy exhaustiva. (Laguna y Gonzalez, 1991; Kontoravdis y Bard, 1995; Bautista, et al., 1999; Feo, et al., 1995).

2) Recocido simulado (s*imulated annealing*). Se basa en una analogía con el proceso de enfriamiento lento de los metales, formalizado por Metrópolis et al. (1953). A partir de este procedimiento Kirkpatrick et al. (1983) resuelven un problema de optimización combinatoria que aparece en el diseño de circuitos impresos. La característica de esta heurística es que no depende de la solución inicial. En el recocido simulado esto se consigue mediante el uso de una temperatura inicial alta que permite pasar por muchas soluciones muy alejadas entre sí en las primeras iteraciones del algoritmo. Así, la primera solución a partir de la cual el algoritmo empieza a evolucionar, se genera de manera aleatoria. La definición de la temperatura inicial está relacionada con una solución inicial. Una temperatura elevada aumenta la probabilidad de descartar una solución buena, por eso no tiene sentido invertir tiempo en generar soluciones iniciales buenas mediante alguna

heurística. Algunos enfoques reinicializan repetidamente el algoritmo desde varios puntos del espacio de soluciones, para lograr una exploración más exhaustiva, mientras que otros autores sugieren tomar una muestra de soluciones y usar la mejor de ellas como solución inicial.

Muchos autores han señalado la gran influencia que tiene la definición del entorno sobre los procedimientos de búsqueda local, y sobre los algoritmos de recocido simulado en particular (Cerny, 1985). Así, (Osman & Potts, 1989) definen dos tipos de movimientos, intercambio de trabajos e inserción de un trabajo entre otros dos y, para generar el entorno, usan dos estrategias: una ordenada y otra aleatoria. Los resultados de los experimentos son mejores en el caso de movimientos de inserción generados de forma aleatoria. Jeffcoat & Bulfin (1993) plantean que es más conveniente utilizar el primer tipo de movimientos en problemas con pocas restricciones, mientras que el segundo es mejor utilizarlo en problemas altamente restringidos (mediante tiempos de inicio o entrega, restricciones tecnológicas, etc.). Los autores usan este último tipo de movimiento en un problema de programación en recursos paralelos a capacidad finita, demostrando mejores resultados que los procedimientos basados en heurísticas específicas.

En cuanto al enfriamiento, es necesario determinar el grado de enfriamiento, el cual es una función que regula la evolución de la temperatura en cada iteración del algoritmo. Para evitar una convergencia prematura, esta función no debe decrecer muy rápidamente, pero tampoco debe evolucionar muy lentamente, para evitar un tiempo excesivo de cálculo.

El número de iteraciones definido sin modificar la temperatura debe ser lo suficientemente grande para no converger a un óptimo local. Algunos autores indican que, a medida que disminuye la temperatura, el número de iteraciones debe aumentar para realizar una exploración cada vez más exhaustiva. Las iteraciones finalmente paran cuando se llega a la temperatura final. Otro criterio de paro es, por ejemplo, parar cuando no hay mejora en un número de iteraciones consecutivas.

3) Búsqueda tabú (*tabu search*). Consiste en la búsqueda por entornos donde las posiciones visitadas se almacenan en una lista. La búsqueda tabú selecciona el mejor de los movimientos posibles en un entorno $N(x)$ de una solución x. Dentro del proceso de exploración, al alcanzar un óptimo local x_0 se repite la elección del mejor movimiento del

vecindario $N(x_0)$ y se vuelve a la posición previa al óptimo local. A partir de esta posición, se vuelve a visitar x_0, con lo que, el algoritmo entra en un ciclo sin fin. Para evitar esta situación, se dota al algoritmo de una estructura en forma de lista circular (lista tabú) de tamaño *L*, donde se almacenan las posiciones visitadas más recientemente, prohibiendo que se visiten mientras se encuentran en ella, a menos que se cumpla un criterio de aspiración (usualmente, que se mejore el valor previo de la función objetivo). Con esta estrategia, se intenta desarrollar una exploración lo más exhaustiva posible del espacio de soluciones (Glover, 1990). La lista tabú donde se almacenan estos atributos, se actualiza conforme se explora el espacio de soluciones, de tal manera que los movimientos que se han realizado hace más de *L* iteraciones, se sustituyen por los más recientes (estrategia FIFO).

4) Optimización basada en colonia de hormigas (a*nt colony optimization*). Esta metaheurística fue desarrollada por Colorni, et al. (1992). Se inspira en el comportamiento de las colonias de hormigas para solucionar problemas de optimización combinatoria. Las colonias de hormigas tienen la capacidad de encontrar la ruta más corta entre el hormiguero y las fuentes de alimento. Mientras hacen sus recorridos las hormigas depositan en la tierra una sustancia química llamada feromona, formando en el camino un rastro de dicha sustancia. Las hormigas perciben la feromona y cuando tienen que escoger una ruta se deciden por aquella que tenga el rastro más fuerte. El rastro de feromonas también les permite encontrar el camino de regreso al hormiguero (ó a la fuente de alimento) y transmitir esta información a otras hormigas. Si no se encuentra ningún rastro de feromona, las hormigas se mueven de manera básicamente aleatoria, pero cuando existe feromona depositada, tienen mayor tendencia a seguir el rastro. La elección entre distintos caminos toma lugar cuando varios caminos se cruzan. Entonces, las hormigas eligen el camino a seguir con una decisión probabilística sesgada por la cantidad de feromona: cuanto más fuerte es el rastro de feromona, mayor es la probabilidad de elegirlo. Puesto que las hormigas depositan feromona en el camino que siguen, este comportamiento lleva a un proceso de auto refuerzo que concluye con la formación de rastros señalados por una concentración de feromona elevada.

Los algoritmos de ACO (A*nt Colony Optimization*) se basan en una colonia de hormigas artificiales, esto es, unos agentes computacionales simples que trabajan de

manera cooperativa y se comunican mediante rastros de feromona artificiales. En Colorni, et al. (1992) se aplica un algoritmo de hormigas a la resolución del problema del comerciante viajero. Para ello, se definen los siguientes elementos del algoritmo:

En cada ciudad se sitúan inicialmente *bi*(0) hormigas o agentes. Cada una de ellas construirá una trayectoria mediante los siguientes elementos:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t,t+1)$$

Donde $\tau_{ij}(t)$ es la intensidad de la trayectoria entre las ciudades *i* y *j* en el instante *t*, *p* es un coeficiente denominado de evaporación, y $\Delta\tau_{ij}(t,t+1) = \Sigma_{k=1}^{m}\Delta^{k}\tau_{ij}(t,t+1)$ (el aumento de la intensidad entre dos ciudades es la suma de los aumentos de intensidad para lo *m* agentes u hormigas definido. En el trabajo se define $\tau_{ij}(0)$ un valor pequeño. Los autores proponen tres variantes para calcular la intensidad que denominan *ant-quantity*, *ant-density* y *ant-cycle*.

En el primer caso:

$$\Delta^{k}\tau_{ij}(t,t+1) = \begin{cases} \dfrac{Q_1}{d_{ij}} & \text{si la hormiga } k \text{ viaja desde } i \text{ a } j \text{ entre } t \text{ y } t+1; \\ 0, & \text{en cualquier otro caso.} \end{cases}$$

En el segundo caso:

$$\Delta^{k}\tau_{ij}(t,t+1) = \begin{cases} Q_2 & \text{si la hormiga } k \text{ viaja desde } i \text{ a } j \text{ entre } t \text{ y } t+1; \\ 0, & \text{en cualquier otro caso.} \end{cases}$$

Por último:

$$\Delta^{k}\tau_{ij}(t,t+1) = \begin{cases} \dfrac{Q_3}{L^{k}} & \text{si la hormiga } k \text{ utiliza el tramo } ij \text{ en su recorrido;} \\ 0, & \text{en cualquier otro caso.} \end{cases}$$

En este último caso L^{k} es la longitud del recorrido de la hormiga *k*.

$Q_i(i = 1,2,3)$ es una constante denominada intensidad de feromona.

Se define visibilidad al valor $\eta_{ij} = \frac{1}{d_{ij}}$, siendo d_{ij} la distancia euclídea entre dos ciudades.

En cada iteración cada una de las *k* hormigas tiene una probabilidad de pasar de la ciudad *i* a la *j* definida por:

$p_{ij} = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}(t)\right]^{\beta}}{\sum_{j=1}^{n}\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}(t)\right]^{\beta}}$, siendo α y β parámetros de control de la importancia de los dos factores que afectan.

Mediante una sencilla lista denominada tabú se prohíbe a lo largo de la ejecución del algoritmo que una hormiga visite una ciudad ya introducida en la lista.

Cada hormiga en la iteración *t* completa un recorrido, partiendo de la ciudad donde inicialmente fue asignada. Cada movimiento realizado se usa para actualizar la intensidad y la probabilidad de transición. Cuando la lista tabú se llena indica que se ha completado, se memoriza el recorrido más corto obtenido y se reinicia el proceso.

5) Búsqueda local iterativa (*iterative local search*). Está basada en la búsqueda local que es la base de muchos de los métodos usados en problemas de optimización.

La búsqueda local empieza con una solución inicial y busca en su vecindad una mejor solución. Si la encuentra, reemplaza su solución actual por la nueva y continua con el proceso, hasta que no se pueda mejorar la solución actual. Su procedimiento es el siguiente:

Procedimiento de búsqueda local:
s = genera una solución inicial
mientras *s* no es óptimo local **hacer**
$s' \in N(s)$ con $f(s) < f(s')$
(solución mejor dentro de la vecindad de *s*)
$s \leftarrow s'$
fin
regresar *s*

La búsqueda local tiene la ventaja de encontrar soluciones muy rápidamente. Su principal desventaja es que queda atrapada fácilmente en mínimos locales y su solución final depende fuertemente de la solución inicial. Un conjunto de entradas nos generan la misma salida (mínimo local). Esto se ve como un pozo de atracción.

La búsqueda local iterativa trata de generar vecinos que nos generen pozos de atracción vecinos (o al menos diferentes). Una posibilidad es generar un camino de vecinos seleccionados aleatoriamente, $s_1, s_2, ..., s_i$, donde s_{j+1} es vecino de s_j. Determinar entonces, el primer vecino que pertenece a otro pozo de atracción. Con esto, se va generando pozos de atracción vecinos. Sin embargo, es un proceso computacionalmente muy costoso, por la cantidad de llamadas a búsqueda local que se requieren. En lugar de la noción de vecinos en pozos locales, podemos tomar una noción de vecinos más débil. Dado un mínimo local, se crea una perturbación que pertenezca al espacio de estados legales y que nos genere un nuevo estado. En ese nuevo estado se aplica búsqueda local. Si el óptimo local encontrado pasa una prueba de aceptación (es mejor), entonces, se acepta, si no se regresa al punto anterior. El procedimiento es el siguiente:

Procedimiento de búsqueda local iterativa:

s_0 = genera una solución inicial

$s*$ = búsqueda local (s_0)

repetir

s' = perturbación($s*$, historia)

$s*'$ = búsqueda local (s_0)

$s*$ = criterio de aceptación($s*, s*'$, historia)

hasta criterio de terminación

La búsqueda local iterativa produce un muestreo con un sesgo adecuado, siempre y cuando las perturbaciones no sean ni muy grandes ni muy pequeñas. Las perturbaciones grandes nos regresan a un reinicio aleatorio, mientras que las perturbaciones pequeñas no nos sacan del pozo de atracción. Normalmente la búsqueda local iterativa no es reversible (no regresamos a los mismos sub-espacios explorados antes). A las perturbaciones se les deben incluir aspectos aleatorios o ser adaptivas para evitar ciclarse.

Kochhar y Morris (1987) resuelven un problema concreto de líneas de flujo flexibles utilizando búsqueda local, donde se consideran tiempos de cambio, bloqueo y descompostura aleatoria de las máquinas. Separan el problema en secuenciación y asignación. El criterio considerado es la minimización del tiempo medio de flujo.

6) Algoritmos genéticos (*genetic algorithms*). Un AG es un procedimiento basado en una analogía con la evolución de los seres vivos. Se buscan soluciones aproximadas a problemas de gran complejidad computacional mediante un procedimiento de evolución simulada matemáticamente en una computadora (Holland, 1975).

Un AG básico consta de cuatro fases bien definidas. En primer lugar se genera una "población de individuos" formada por un conjunto de soluciones al problema que se crean de manera aleatoria o mediante alguna heurística conocida. La información relevante de cada individuo es almacenada en los cromosomas o conjunto de genes (valores posibles). A continuación se realiza un proceso de selección de cromosomas o individuos padres, donde una o más soluciones se escogen mediante algún procedimiento aleatorio, teniendo en cuenta el valor de la función de aptitud (*fitness*), la cual mide el grado de adaptación del individuo a su entorno y que usualmente depende de la función objetivo. Así, la probabilidad de elegir individuos "buenos" es mayor que la de elegir individuos poco relevantes en términos de la función objetivo a optimizar. Después, a los individuos elegidos se les aplican operadores genéticos: cruce y mutación. Por último, se evalúa la aptitud de las nuevas soluciones generadas (individuos hijos) y se modifica la población para repetir el proceso hasta que se cumpla el criterio de paro.

Uno de los primeros trabajos donde se muestra la utilidad de AGs en la programación de la producción, se debe a Vancza y Markus (1991), quienes aplican AGs para la resolución de un problema simple de taller de flujo con relaciones de precedencia. El problema de programación de la producción en un taller de flujo es analizado de manera general por (Chen, et al., 1995). Sin embargo, uno de los primeros trabajos relacionados con AGs y con el problema del taller de flujo se debe a Cleveland y Smith (1989) que resuelven un problema muy parecido al taller de flujo.

Reeves (1995) publicó un AG para el taller de permutación con características novedosas. Las características fundamentales del algoritmo son una mutación adaptativa y

el esquema generacional utilizado. Mediante la mutación adaptativa, Reeves controla que la población no converja prematuramente. Inicialmente se permite una probabilidad de mutación dada que se aumenta conforme la diversidad de la población baja (tomando como medida la razón entre el valor de adecuación del mejor individuo y el valor de adecuación medio de la población). Si la diversidad de la población sobrepasa un cierto umbral máximo, entonces se restituye la probabilidad de mutación inicial. El esquema generacional difiere fundamentalmente de los AGs estándar: Una vez que se han aplicado los operadores de selección, cruce y mutación, los hijos generados no reemplazan directamente a los padres, sino que reemplazan a aquellos individuos de la población que tienen un valor de adecuación inferior a la media de la población. A los AGs que utilizan este tipo de esquemas generacionales (donde los hijos no reemplazan a los padres sino a otros individuos) se les conoce como AG tipo "*steady state*". El autor compara su AG con el recocido simulado de Osman y Potts (1989) y con una sencilla búsqueda local. El AG propuesto por Reeves resultó más eficaz. Para la evaluación el autor utiliza dos bancos de pruebas: uno con datos aleatorios propuesto por él mismo y el conjunto de ejemplos de Taillard (Taillard, 1993); este último se sigue utilizando como un estándar en la actualidad con adaptaciones y extensiones.

Algunos autores estudian posteriormente la hibridación de AGs con otras técnicas (Reeves & Höhn, 1996; Murata, et al., 1996; Reeves & Yamada, 1998; Yamada & Reeves, 1997). Cabe destacar, que los tiempos de procesamiento necesarios para obtener buenas soluciones superan en algunos casos las tres horas de cómputo en una supercomputadora. Onwubolu y Mutingi (1999) presentan un AG, considerando los criterios de la tardanza media, el número de trabajos retrasados y una combinación de ambos. El AG propuesto es sencillo y la aportación de este algoritmo se centra en la consideración de los objetivos mencionados anteriormente, los cuales no son frecuentes en la literatura. Jain, Bagchi y Wagneur (2000) realizan un extenso estudio sobre el resultado de la hibridación de un AG (Jain & Bagchi, 2000). A partir del estudio se aclara que la inicialización dirigida de la población tiene un efecto positivo sobre la eficacia del AG. También se observa cómo la hibridación mediante la aplicación de mejoras a los individuos con diversas heurísticas resulta en AGs más eficaces. Ponnambalam, Aravindan y Chandrasekaran (2001) desarrollan un sencillo AG con un operador de cruce llamado GPX (*Generalized Position*

Crossover), mutación por intercambio y una inicialización de la población aleatoria. Los autores comparan el rendimiento de este algoritmo con algunas de las heurísticas más conocidas (Palmer, Gupta, CDS, RA y NEH), resultando ser, según sus pruebas, superior en todos los casos. Tang y Liu (2002) publican un AG para un problema con criterio de tardanza media. El AG aplica dos operadores no estándares, los cuales son el operador de filtrado (*filtering*) y el de cultivo (*cultivation*). El primer operador se aplica después de cada generación y reemplaza los dos peores individuos de la población con los dos mejores. El segundo operador es una modificación de la búsqueda local. Simplemente, cuando el mejor individuo de la población ha permanecido sin cambios durante un número determinado de generaciones, se aplica una búsqueda local en el vecindario restringido (sólo se intercambian trabajos adyacentes). Los autores comparan el AG con la heurística de Ho y Chang y con la de Rajendran, resultando el AG mejor.

Recientemente Ruiz y Maroto, (2006) y Ruiz et al. (2008) aplican AGs a un problema complejo en la programación de la producción considerando el TFH de múltiples etapas con restricciones de máquinas no relacionadas, tiempos de ajuste dependientes de la secuencia y elegibilidad de máquinas. Se demuestra que sus resultados son mejores para resolver este tipo de problemas; entre 53% y 135% por encima del segundo mejor método, comparando con 9 variantes de métodos metaheurísticos de alto desempeño, entre los cuales se encuentran los algoritmos de Reeves (1995), Nawaz et el. (1983), Osman y Potts (1989), Widmer y Hertz (1989), Chen, et al. (1995), Murata, et al. (1996), Aldowaisan y Allahvedi (2003), Rajendran y Ziegler (2004).

Capítulo 3

Problema del taller de flujo híbrido

En un TFH se procesa un conjunto *N* de trabajos, $N = \{1, 2, ..., n\}$, en un conjunto *M* de etapas, $M = \{1, 2, ..., m\}$ y en cada etapa $i \in M$, un conjunto $M_i = \{1, ..., m_i)$ de máquinas paralelas (no relacionadas en al menos una etapa) procesan los trabajos. Un trabajo *j* se procesa en todas las *m* etapas y dentro de cada etapa *i* en una de las m_i máquinas. El TFH es una generalización del taller de flujo simple (ver apartado 2.3).

El problema de TFH se estudió por primera vez en Arthanary y Ramaswamy (1971). Salvador (1973) y Brah & Hunsucker (1991) demostraron que el número de posibles rutas para los trabajos es muy elevado:

$$\prod_{i=1}^{m} \left[\binom{n-1}{m_i} \cdot \frac{n!}{m_i!} \right],$$

donde *n* es el número de trabajos, *m* el número de etapas y m_i es el número de máquinas en la etapa *i*, $i = (1, ..., m)$.

Gourgand, Grangeon y Norre (1999) proponen otra evaluación para el número de posibles soluciones en un problema de tipo HFS:

$$n! \left(\prod_{i=1}^{m} m_i \right)^n$$

El taller de flujo simple se investiga desde los años 50 con la aportación de Johnson (1954) que proporciona un resultado óptimo para dos máquinas ($F2 \,||\, C_{\max}$). Cuando el número de máquinas es 3 ó más, el problema es de tipo **NP**-Difícil. Gupta (1988) demostró que este problema con solo dos etapas es **NP**-Difícil aun cuando una de las dos etapas

contenga una máquina. De acuerdo a Garey y Johnson (1979) el problema con tres etapas ($F3||C_{\max}$), es **NP**-Difícil en sentido estricto. Siendo el TFH una generalización del taller de flujo simple, podemos concluir que el TFH es también **NP**-Difícil.

El TFH por el número de etapas, se clasifica en:

1) TFH con dos etapas (TFH2).
2) TFH con tres etapas (TFH3).
3) TFH con *k* etapas (TFH*k*).

3.1 Taller de flujo híbrido con dos etapas

Dentro de esta clasificación se distinguen tres diferentes configuraciones:

1) TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda etapa. (m_1=1, m_2>1).
2) TFH2 con múltiples máquinas en la primera etapa y una máquina en la segunda etapa. (m_1>1, m_2=1).
3) TFH2 con múltiples máquinas en la primera etapa y múltiples máquinas en la segunda etapa. (m_1>1, m_2>1).

3.1.1 TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda etapa

La Tabla 3.1 muestra los algoritmos para el problema del TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda. Una de las primeras publicaciones que tratan los TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda etapa se debe a Sriskandarajah y Sethi (1989). También 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 simple.

Más adelante Gupta y Tunc (1991) extienden el trabajo de Gupta (1988) para el caso de una máquina en la primera etapa y máquinas múltiples en la segunda etapa; probándose que ofrece resultados óptimos en el caso en el que el número de máquinas en la segunda etapa sea mayor que el de trabajos. En caso contrario, se plantean dos heurísticas y un

algoritmo de ramificación y acotación para abordar problemas de tamaño pequeño. El objetivo es minimizar el tiempo de completar todos los trabajos (*makespan*).

Gupta y Tunc (1998), vuelven a considerar el mismo problema pero ahora el objetivo es minimizar el número de trabajos retrasados (*tardy jobs*). En este artículo se proponen seis métodos heurísticos y se hace una extensa comparativa entre ellos.

Tabla 3.1 Algoritmos para el TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda

Año	Autor	Notación	Método
1989	Sriskandarajah y Sethi	$FF2,((1^{(1)}),(PM^{(2)}))//C_{\max}$	Heurísticas de lista
1991	Gupta y Tunc	$FF2,((P1^{(1)}),(P2^{(2)}))//C_{\max}$	Heurísticas basadas en reglas de prioridad
1998	Gupta y Tunc	$FF2,((P1^{(1)}),(PM^{(2)}))//N_T$	Algoritmos heurísticos
1998	Huang y Li	$FF2,((1^{(1)}),(QM^{(2)}))//C_{\max}$	Heurísticas para caso real
2002	Riane, Artiba y Elmaghraby	$FF2,((1^{(1)}),(R2^{(2)}))//C_{\max}$	Programación dinámica, heurística voraz
2003	Lin y Liao	$FF2,((1^{(i)}),(R2^{(2)}))/S_{sd}/T_{\max}$	Heurística
2004	Chang, Yan y Shao	$FF2,((P1^{(1)}),(PM^{(2)}))//C_{\max}$	Algoritmos heurísticos
2004	Lee y Kim	$FF2,((P1^{(1)}),(PM^{(2)}))//T_{\max}$	Ramificación y acotación
2005	Guirchoun, Martineau y Billaut	$FH2,((1^{(1)}),(PM^{(2)}))/$ $/nwt, p_{i,1}=1/\sum C_i$	Algoritmo polinomial para un caso particular
2005	Xie Y Wang	$FF2,((1^{(1)}),(PM^{(2)}))/M_j/C_{\max}$	Heurísticas de lista
2006	Allaoui y Artiba	$FF2,((1^{(1)}),(PM^{(2)}))/brkdwn,nr/C_{\max}$	Ramificación y acotación. Heurísticas

Huang (1998), estudia un caso real con una máquina en la primera etapa y múltiples máquinas uniformes en la segunda. El objetivo es minimizar el C_{max}. Para resolver el problema desarrolla dos heurísticas y ocho reglas de secuenciación, sobre los cuales se realizan las comparaciones.

Riane, Artiba y Elmaghraby (2002) tratan este problema con una máquina en la primera etapa y dos diferentes máquinas en la segunda etapa para minimizar el C_{max}. Formulan una programación dinámica, adoptan enfoques de Johnson e incluyen una heurística del tipo voraz (*greedy*). El objetivo es minimizar el máximo tiempo de completar los trabajos.

Lin y Liao (2003), aplican su investigación a una empresa de manufactura de etiquetas con la configuración de un TFH2 con la característica de tiempos de cambio de partida dependientes de la secuencia en la primera etapa, múltiples máquinas en la segunda etapa y dos tiempos de finalización (*due dates*). El objetivo es minimizar la máxima tardanza ponderada (*weighted maximal tardiness*).

Chang, Yan y Shao (2004), investigan el TFH2 con una máquina en la primera etapa y múltiples máquinas en la segunda. El objetivo es minimizar el C_{max}. En el modelo no hay tiempos de espera (*no-wait*). Los tiempos de ajuste (*setup time*) y liberación del trabajo (*removal time*) se consideran independientes al tiempo de procesamiento. Se proponen dos algoritmos heurísticos que ayudan a resolver este problema.

Lee y Kim (2004), consideran un TFH2 cuando hay una máquina en la primera etapa y múltiples máquinas idénticas en la segunda etapa. El objetivo es minimizar el total de tardanza de trabajos (*minimizing total tardiness*). Se determina un límite inferior. Se propone un algoritmo basado en ramificación y acotación, el cual a través de experimentos computacionales sugiere encontrar soluciones óptimas a problemas de este tipo hasta con 15 trabajos en un tiempo razonable.

Guirchoun, Martineau y Billaut (2005), investigan el problema 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 tiempo de completar todos los trabajos (*minimize the total completion time*). Se demuestra que el algoritmo propuesto es óptimo para el caso planteado. Xie y Wang (2005), analizan la complejidad y aproximación del TFH2 para el caso no reanudable cuando se interrumpen las máquinas; se contemplan

restricciones de disponibilidad determinísticas. Proveen algunos algoritmos de aproximación con límites finitos de desempeño en el peor caso para algunas variaciones del problema.

Xie y Wang (2005), consideran este problema con restricciones de disponibilidad. Se discute la complejidad del problema y proponen algoritmos de lista con el objetivo de minimizar el C_{max}.

Por último, Allaoui y 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 son reanudados si se interrumpen. Proponen un algoritmo de ramificación y acotación para minimizar el C_{max}. Se calcula el peor caso de 3 heurísticas: algoritmo LIST, algoritmo LPT y heurística H, propuestos por Lee-Vairaktarakis (1994). El algoritmo consiste en 3 pasos: cálculo de límites, procedimiento de ramificación y eliminación de nodos. Utiliza el mismo enfoque y concepto de Brah y Hunsucker (1991) pero modifica la estrategia de ramificación y elimina nodos para agregar restricciones de disponibilidad de las máquinas.

3.1.2 TFH2 con múltiples máquinas en la primera etapa y una máquina en la segunda etapa

La Tabla 3.2 muestra los algoritmos para el problema del TFH2 con múltiples máquinas en la primera etapa y una máquina en la segunda. Uno de los primeros artículos que tratan los Talleres de Flujo Flexible de dos etapas con múltiples máquinas en la primera etapa y una máquina en la segunda etapa, se debe a Arthanary y Ramaswamy (1971) que desarrollaron un algoritmo de ramificación y acotación. Proporciona la solución exacta a un problema pequeño.

Más adelante Gupta (1988), analizó el caso de un TFH2 con 2 máquinas en la primera etapa y una máquina en la segunda etapa con el objetivo de minimizar el *makespan*. Además demostró que este problema es *NP*-completo y que la resolución de este tipo de problema implica minimizar el tiempo total de inactividad en la última etapa, como en el taller de flujo simple de dos etapas. Plantea una heurística de lista basada en índices con el objetivo que la máquina asignada al trabajo en la primera etapa quede libre lo

más tarde posible (regla basada en el tiempo de proceso más largo). Las técnicas que presentó Gupta se basan en la Regla de Johnson.

Posteriormente, Vignier, Billaut, Proust y Tkindt (1996), propusieron una modelización analítica y dos heurísticas para resolver el problema del TFH2 con una máquina en la segunda etapa, con el objetivo de minimizar la tardanza máxima.

El siguiente año, Gupta, Hariri y Potts (1997) trataron el mismo problema pero con el objetivo de minimizar el *makespan*, proponiendo un algoritmo nuevo de ramificación y acotación junto con nuevos límites superiores e inferiores. También propusieron 4 algoritmos heurísticos, uno de ellos basado en búsqueda local descendente.

Ling-Huey (2003), examina el mismo problema pero con restricciones de tiempos de espera limitados para cada trabajo. El objetivo es minimizar el *makespan*. Se propone un modelo de programación entera y un algoritmo heurístico.

Tabla 3.2 Algoritmos para el TFH2 con múltiples máquinas en la primera etapa y una máquina en la segunda

Año	Autor	Notación	Método
1971	Arthanary y Ramaswamy	$FF2,((PM^{(1)}),P1^{(2)}))//C_{\max}$	Ramificación y acotación
1988	Gupta	$FF2,((P2^{(1)}),(P1^{(2)}))//C_{\max}$	Heurísticas basadas en la regla de Johnson
1996	Vignier, Billaut, Proust y Tkindt	$FF2,(PM^{(1)},1)/\ pmtn^{(1)},d_i^{(2)}/T_{\max}$	Heurísticas
1997	Gupta, Hariri y Potts	$FF2,((PM^{(1)}),(P1^{(2)}))//C_{\max}$	Heurísticas y Ramificación y acotación
2003	Ling-Huey	$FF2,((PM^{(1)}),(1^{(2)}))/wait_{ij}/C_{\max}$	Heurística y programación entera mixta
2005	Xie y Wang	$FF2,((PM^{(1)}),(1^{(2)}))/M_j/C_{\max}$	Heurísticas de lista
2005	Zhang, Yin, Liu y Linn	$FF2,((PM^{(1)}),(1^{(2)}))//C_{\max}$	Algoritmos heurísticos

Más recientemente, Xie y Wang (2005), analizaron la complejidad y aproximación del TFH2 incluyendo este problema específico con el objetivo de minimizar el *makespan*.

Por último, Zhang, Yin, Liu y Linn (2005), abordaron este problema proponiendo dos algoritmos heurísticos con el objetivo de minimizar el *makespan*. Ambos algoritmos comparten el mismo enfoque pero se diferencian en la manera cómo secuencian los trabajos. Determinaron la secuencia de los trabajos usando las heurísticas de Rajendran y Chaudhuri (1992).

3.1.3 TFH2 con múltiples máquinas en ambas etapas

La Tabla 3.3 muestra los algoritmos para el problema del TFH2 con múltiples máquinas en ambas etapas. Sriskandarajah y Sethi (1989) fueron los primeros en abordar el problema de TFH2 con múltiples máquinas, pero con el mismo número de máquinas en ambas etapas. Utilizan una variante del algoritmo de Johnson aplicada a la descomposición del problema en M subproblemas de taller de flujo simple. El objetivo es minimizar el $C_{\max}$.

Sherali, Sarin y Kodialam (1990), se enfocan en este tipo de problemas motivados por una situación real en la industria papelera donde consideran 10 máquinas en la primera etapa y 12 máquinas en la segunda etapa. Fueron los primeros en considerar tiempos de ajuste dependientes de la secuencia de los trabajos. Dividen el problema en secuenciación y asignación y proponen diversos modelos matemáticos para resolver el problema. Consideran una función de costos de cambio y costos de asignación de los trabajos a las máquinas como criterio de optimización.

Dos años después Rajendran y Chaudhuri (1992), incluyen en su investigación este problema de múltiples máquinas en ambas etapas y proponen un procedimiento de ramificación y acotación. El algoritmo propuesto supera a una heurística de lista (*list scheduling*) basada en la regla de tiempo de proceso más corto, utilizada con fines de comparación.

Lee y Vairaktarakis (1994), analizan el mismo problema abordado por Sriskandarajah y Sethi (1989) donde hay un igual número de máquinas por etapa. Para su resolución utilizan algoritmos de lista. También extienden estos métodos al caso de un número de etapas $k > 2$ y demuestran que esta heurística tiene una cota de error tan buena como la de otras heurísticas conocidas hasta ese entonces.

Tabla 3.3 Algoritmos para el TFH2 con dos etapas y múltiples máquinas en ambas etapas

Año	Autor	Notación	Método
1989	Sriskandarajah y Sethi	$FF2,((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	Heurísticas de lista
1990	Sherali, Sarin y Kodialam	$FF2,((PM^{(i)})_{i=1}^{(2)})/S_{sd}/Costes$	Programación entera mixta
1992	Rajendran y Chaudhuri	$FF2,((PM^{(i)})_{i=1}^{(2)})//F_{\max}$	Ramificación y acotación
1994	Lee y Vairaktarakis	$FF2,((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	Heurística
1996	Guinet y Salomon	$FF2,((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	Heurística y reglas de prioridad
2000	Hong y Wang	$FF2,((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	Lógica difusa
2000	Koulamas y Kyparisis	$FF2,((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	Algoritmos de tiempo lineal para caso particular
2003	Oguz, Ercan, Cheng y Fung	$FF2,((PM^{(i)})_{i=1}^{(2)})/size_{ij}/C_{\max}$	Heurísticas basadas en reglas de prioridad
2003	Soewandi y Elmaghraby	$FF2,((QM^{(i)})_{i=1}^{(2)})//C_{\max}$	Heurística
2004	Hong y Wang	$FF2,((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	Lógica difusa
2005	Kyparisis y Koulamas	$FF2,((QM^{(i)})_{i=1}^{(2)})//C_{\max}$	Heurística
2005	Wang, Xing y Bai	$FF2,((PM^{(1)}),(P(M+1)^{(2)}))/nwt$	Heurística basada en LPT

Guinet y Solomon (1996), proponen una solución con el objetivo de minimizar el $C_{\max}$ dividiendo el problema en secuenciación y asignación. Proponen un modelo de programación entera con variables binarias y algunos límites inferiores para el problema. En este caso, la regla de Johnson muestra buenos resultados para la etapa de asignación.

Posteriormente, Hong y Wang (2000), proponen un algoritmo heurístico difuso en la secuenciación de un TFH2. Se usa un algoritmo LPT difuso para la asignación de trabajos. Se consideran inciertos los tiempos de procesamiento para apegarse más a la realidad de ciertos procesos. Se basan en el trabajo de Sriskandarajah y Sethi (1989). El objetivo es minimizar el $C_{\max}$.

Koulamas y Kyparisis (2000) consideran un TFH de dos y tres etapas, con máquinas paralelas en cada etapa y con el objetivo de minimizar el C_{max}. Desarrollan algoritmos de tiempo lineal para estos problemas.

Más adelante Oguz, Ercan, Cheng y Fung (2003), proponen algoritmos heurísticos basados en reglas de prioridad para el TFH2 con el objetivo de minimizar el C_{max}. Se aplica al procesamiento digital de imágenes en tiempo real.

Soewandi y Elmaghraby (2003), estudian el problema de TFH2 con máquinas uniformes en cada etapa con el objetivo de minimizar el C_{max}. Desarrollan tres procedimientos heurísticos, de los cuales, el primero de ellos resulta eficiente en los experimentos computacionales para el caso de tiempos de procesamientos simétricos. Los otros dos resultaron ser mejores para tiempos de procesamiento no simétrico.

Hong y Wang (2004), tratan el problema considerando tiempos de procesamiento inciertos para acercarse a ciertas situaciones reales. Proponen un algoritmo difuso.

Posteriormente Kyparisis y Koulamas (2005), consideran el TFH2 con máquinas paralelas uniformes. Complementan el trabajo de Soewandi y Elmaghraby (2003), cuyo desempeño es afectado cuando las velocidades de las máquinas varían ampliamente. Motivados por esta observación proponen una alternativa para solucionar este problema. El criterio es minimizar el C_{max}.

Por último Wang, Xing y Bai (2005), enfocan su investigación en la industria del acero, básicamente en los procesos de derretir (*meeting*) y vaciar (*casting*). Las restricciones son que no hay tiempo de espera (*no-wait*) entre las dos operaciones y no debe haber tiempo ocioso (*no-idle*) en las máquinas. El objetivo es minimizar el C_{max}. Se discute la complejidad del problema y se propone un algoritmo heurístico basado en LPT.

3.2 Taller de flujo híbrido con tres etapas.

Uno de los primeros trabajos enfocados en esta clasificación es el de Wittrock (1985), quien desarrolló una serie de algoritmos para el problema FFL (*Flexible Flow Line*) donde algunos trabajos saltan una de las etapas de la producción y donde las máquinas dentro de una etapa son idénticas. En principio considera tres etapas productivas y separa el problema en tres partes: asignación, secuenciación y programación de las operaciones y los resuelve

por este orden. Los problemas ejemplo que resuelven son muy específicos y también las técnicas desarrolladas. Los objetivos son maximizar el número de trabajos procesados por unidad de tiempo (*throughput*) y reducir el WIP (*Work in Process*).

Nuevamente Wittrock (1988) extendió el anterior estudio realizado para el caso donde el almacenamiento (*buffer*) entre etapas es limitado. El tiempo necesario para realizar un trabajo depende del recurso utilizado y del trabajo. Se proponen unas heurísticas en la que primero se asignan los trabajos a las máquinas en cada etapa tratando de minimizar en cada etapa la carga de la máquina cuello de botella, después se secuencian los trabajos en cada máquina y, por último, se calcula el tiempo de inicio de cada trabajo para equilibrar la carga de las máquinas. El criterio de optimización es minimizar el C_{max}.

Este mismo año, Riane, Artiba y Elmagharaby (1998), plantean un estudio de un TFH3 a partir de un caso de una industria de fabricación de tableros, con una máquina en las etapas uno y tres y dos máquinas en la etapa intermedia en la que la asignación de los trabajos se realiza de antemano. Proponen dos algoritmos para su resolución. El primero se basa en aplicar programación dinámica a una secuencia de los trabajos obtenidos, mediante la regla CDS (*Complex Discrete System*), y divide el problema en dos subproblemas en función de que los trabajos se asignen a la primera o segunda máquina de la segunda etapa. Consideran que para cada subproblema la capacidad de la máquina de las etapas uno y tres es la original dividida por el número de trabajos que pertenecen a cada subproblema y multiplicada por los tiempos de proceso en las mismas etapas. El segundo enfoque se basa en un algoritmo de ramificación y acotación. Para ello, definen límites superiores del C_{max}. Este último procedimiento parece ser mejor que el primero.

Koulamas y Kyparisis (2000) consideran un TFH de tres etapas, con máquinas paralelas en cada etapa y con el objetivo de minimizar el C_{max}. Desarrollan algoritmos de tiempo lineal para estos problemas con límites absolutos en el peor caso.

Posteriormente Soewandi y Elmaghraby (2001) proponen soluciones para un FSMP (*Flow Shop with Multiple Processors*) con tres etapas y máquinas idénticas por etapa. El objetivo es minimizar el C_{max}. Las heurísticas propuestas se comparan con nuevos límites inferiores que se desarrollan también en este trabajo.

También en este año Andrés (2001), se centra en el TFH3 con máquinas idénticas e incluye la característica de tiempos de cambio dependientes de la secuencia y propone

métodos de resolución para un problema de una empresa de azulejos. El autor hace varias propuestas de modelos de programación matemática, reglas de prioridad y métodos metaheurísticos, como colonias de hormigas y simulado recocido (*simulated annealing*).

El siguiente año Jin, Ohno, Ito, y Elmaghraby (2002), tratan este problema con máquinas paralelas idénticas aplicado a la industria de tarjetas de circuito impreso, con el objetivo de minimizar el $C_{\max}$. Proponen un procedimiento global que utiliza AGs. Proponen también una programación lineal entera que provee un resultado óptimo para problemas de tamaño pequeño. El procedimiento global está basado en conocidas heurísticas de desempeño probado, lo cual le da cierta solidez. Finalmente los experimentos computacionales muestran la eficiencia de este procedimiento. Al parecer este artículo es el primero que reporta el uso de AGs en un TFH. Para un taller de flujo simple ya Chen (1995) y Reeves (1995) lo habían usado.

Tang, Luh, Liu y Fang (2002) enfocan su atención en la industria del acero para resolver un problema TFH3 con máquinas paralelas idénticas. Usan una combinación de relajación lagrangiana, programación dinámica (*dynamic programming*) y heurísticas. El objetivo es encontrar los requerimientos de entrega y producción "justo a tiempo" (*just-in-time*), tales como, tiempos de cambio de partida, tiempos de remoción, tiempos de entrega, lotes de trabajos y restricciones de precedencia. Experimentos computacionales muestran que tal metodología es eficiente.

Dos años después Babayan y He (2004), desarrollan una metodología general basada en agentes para un TFF de tres etapas minimizando el $C_{\max}$. Realizan un análisis de límites de error usando el límite inferior estimado desarrollado en la literatura como un dato para evaluar esta metodología. Los resultados obtenidos son comparados con las propuestas de Soewandi y Elmaghraby (2001) dando cierta mejora en la mayoría de pruebas del experimento computacional.

Otro estudio más en este año se debe a Bertel y Billaut (2004), que investigan un TFH3 con recirculación. Se pretende minimizar el número ponderado de trabajos retrasados. Se presenta una formulación de programación lineal, límite inferior, se describe un algoritmo voraz (*greedy*) basado en reglas de despacho (*dispatching rules*), así como un AG. Se comparan estos dos algoritmos a través de un experimento computacional. Se concluye que el AG propuesto resulta ser el más eficiente.

Finalmente, Andrés, Albarracín, Tormo, Vicens y García-Sabater (2005), aplican su estudio a la producción de azulejos, donde trabajan en un TFH3 con tiempos de ajuste. Se basan en la filosofía de Tecnología de Grupo (*Group Technology*) agregando un coeficiente de similaridad (*coefficient of similarity*). Los resultados muestran mejoras importantes en un caso real.

La Tabla 3.4 muestra los algoritmos para el problema del TFH3.

Tabla 3.4 Algoritmos para el problema del TFH3

Año	**Autor**	**Notación**	**Método/Comentario**
1985	Wittrock	$HF3,((PM^{(i)})_{i=1}^{(3)})//C_{\max}$	Heurísticas
1988	Wittrock	$HF3,((PM^{(i)})_{i=1}^{(3)})/block/C_{\max}$	Heurísticas
1998	Riane, Artiba y Elmaghraby	$HF3,((P1^{(1)}),(P2^{(2)}),(P3^{(1)}))//C_{\max}$	Programación dinámica y Ramificación y acotación
2000	Koulamas y Kyparisis	$FF3,((PM^{(i)})_{i=1}^{(3)})//C_{\max}$	Algoritmos de tiempo lineal para caso particular
2001	Soewandi y Elmaghraby	$HF3,((PM^{(i)})_{i=1}^{(3)})//C_{\max}$	Heurísticas
2001	Andrés	$HF3,((P4^{(1)}),(P2^{(2)}),(P3^{(3)}))/S_{sd}/C_{\max}$	Heurísticas y metaheurísticas
2002	Jin, Ohno, Ito, y Elmaghraby	$HF3,((PM^{(i)})_{i=1}^{(3)})//C_{\max}$	Algoritmos genéticos
2002	Tang, Luh, Liu y Fang	$HF3,((PM^{(i)})_{i=1}^{(3)})/d_j,brkdwn,S_{nsd},prec/W_j,T_{\max},C_{\max}$	Relajación lagrangiana, programación dinámica y heurísticas
2004	Babayan y He	$HF3,((PM^{(i)})_{i=1}^{(3)})//C_{\max}$	Heurística basada en agentes
2004	Bertel y Billaut	$HF3,((QM^{(i)})_{i=1}^{(3)})/recrc/\bar{D}_w$	Algoritmo genético
2005	Andrés, Albarracín, Tormo, Vicens y García-Sabater	$HF3,((P4^{(1)}),(P2^{(2)}),(P3^{(3)}))/S_{sd}/C_{\max}$	Heurísticas basadas en tecnología de grupo

3.3 Taller de flujo híbrido con *k* etapas

La Tabla 3.5 muestra los algoritmos para el problema del TFH de múltiples etapas. Uno de los primeros estudios de TFH*k* fue realizado por Salvador (1973). Presenta un procedimiento basado en programación dinámica, para la optimización del C_{max} en un TFH*k*, un número diferente de máquinas por etapa y donde los trabajos no esperan entre etapas (*no-wait*). Se aplicó los resultados de los algoritmos a la programación de la producción en una empresa de polimerización de nylon.

Posteriormente, Kochhar y Morris (1987) resuelven un problema muy concreto de líneas de flujo flexibles donde se consideran tiempos de cambio, bloqueo y descompostura aleatoria de las máquinas. Separan el problema en secuenciación y asignación que resuelven en este orden. Los métodos propuestos se basan en búsqueda local descendente y en heurísticas específicas para el problema. El criterio considerado es la minimización del tiempo medio de flujo.

Más adelante, Guinet (1991), enfoca su estudio motivado en la industria de telas. El autor divide también el problema general en subproblemas que resuelve mediante la aplicación de heurísticas concretas para el caso considerado. El objetivo de optimización es la minimización del C_{max} y también la minimización del retraso medio, aunque no se consideran de manera conjunta sino por separado en distintos métodos. Las heurísticas son demasiado específicas para el problema y el autor se limita a resolver ejemplos de problemas reales sin realizar ningún tipo de evaluación objetiva.

Brah y Hunsucker (1991), proponen un procedimiento de ramificación y acotación para el problema de TFH general. Definen dos tipos de cotas mínimas. Una basada en los trabajos y otra basada en las máquinas. El cálculo de estas cotas se obtiene generalizando los razonamientos habituales de taller de flujo al caso de los problemas de TFH. Los autores comparan el rendimiento del procedimiento con una serie de programas generados mediante un sencillo procedimiento de creación de programas sin retraso, obteniendo mejores resultados, sin embargo se observa cómo el tiempo de cálculo crece cuando se resuelven problemas de un gran número de etapas.

Tabla 3.5 Algoritmos para el TFH de múltiples etapas

Año	Autor	Notación	Método
1973	Salvador	$FHm,((PM^{(i)})_{i=m}^{(m)} / nwt / C_{\max}$	Programación dinámica
1987	Kochhar y Morris	$FHm,((PM^{(i)})_{i=1}^{(m)}) / block,$ $brkdwn, S_{nsd} / \bar{F}$	Búsqueda local descendente y heurísticas
1991	Guinet	$FHm,((PM^{(i)})_{i=1}^{(m)}) / S_{sd} / C_{\max}, \bar{T}$	Heurísticas. Caso específico
1991	Brah y Hunsucker	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}$	Ramificación y acotación
1992	Hunsucker y Shah	$FHm,((PM^{(i)})_{i=1}^{(m)}) // \bar{T}, N_T$	Reglas de prioridad
1992	Chandrasekharan y Chaudhuri	$FHm,((PM^{(i)})_{i=1}^{(m)}) // F_{\max}$	Ramificación y acotación
1992	Chandrasekharan y Chaudhuri	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}$	Ramificación y acotación basado en taller de flujo simple
1993	Adler et al.	$FHm,((RM^{(i)})_{i=1}^{(m)}) / S_{sd}$ $/ \sum_{j=1}^{n} w_j T_j$	Reglas de prioridad. Caso específico
1994	Hunsucker y Shah	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}, F_{\max}, \bar{F}$	Reglas de despacho
1995	Aghezzaf et al.	$FHm,((RM^{(i)})_{i=1}^{(m)}) / S_{sd} / C_{\max}, F_{\max}, \bar{F}$	Heurísticas
1996	Guinet y Salomon	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}, T_{\max}$	Heurísticas y reglas de prioridad
1996	Santos, Hunsucker y Deal	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}$	Heurísticas y reglas de prioridad
1998	Nowicki y Smutnicki	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}$	Búsqueda tabú basado en el camino crítico
1998	Portmann et al.	$FHm,((PM^{(i)})_{i=1}^{(m)}) // C_{\max}$	Algoritmo genético con ramificación y acotación
1999	Gourgand, Grangeon y Norre	$FHm,((RM^{(i)})_{i=1}^{(m)}) // C_{\max}$	Recocido simulado

Año	Autor	Notación	Método
1999	Brah y Loo	$FHm,((PM^{(i)})_{i=1}^{(m)})//\bar{F}$	Heurísticas y reglas de prioridad
2000	Botta-Genoulaz	$FHm,((PM^{(i)})_{i=1}^{(m)})/S_{nsd},lags,prec,R_{nsd}/C_{\max}$	Heurísticas
2000	Hong, C. Wang y L. Wang	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Heurísticas y reglas de prioridad
2001	Azizoglu, Cakmak y Kondakci	$FHm,((PM^{(i)})_{i=1}^{(m)})//\sum F_j$	Ramificación y acotación
2001	Kyparisis y Koulamas	$FHm,((PM^{(i)})_{i=1}^{(m)})//\sum w_j C_j$	Heurística basada en reglas de prioridad
2002	H.Djellab, K.Djellab	$FHm,((PM^{(i)})_{i=1}^{(m)})/prec,pmtn/C_{\max}$	Algoritmo polinomial para caso especial
2002	Gupta, Kruger, Lauff, Werner y Sotskov	$FHm,((PM^{(i)})_{i=1}^{(m)})//u_j E_j, v_j T_j, w_j C_j, z_j d_j$	Algoritmos constructivos e iterativos
2002	Wang y Li	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Algoritmo genético y LPT
2003	Alisantoso, Khoo y Jiang	$FHm,((PM^{(i)})_{i=1}^{(m)})/brkdwn/\bar{T}$	Algoritmo inmune
2003	J. Jedrzejowicz y P. Jedrzejowicz	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Metaheurística basada en aprendizaje de población
2003	Wang, Jacob y Rolland	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Redes neuronales
2004	Lee, Kim y Choi	$FHm,((PM^{(i)})_{i=1}^{(m)})//T_{\max}$	Heurística basado en cuellos de botella
2004	Acero-Domínguez y Paternina-Arboleda	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Heurística basado en cuellos de botella
2004	Allaoui y Artiba	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max},\bar{C},T_{\max},\bar{T},N_T,\bar{F},\bar{W}$	Reglas de despacho
2004	Engin y Döyen	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Sistema inmune artificial
2004	Huang, Hong, Horng y Kao	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Algoritmo basado en el enfoque de Palmer y LPT

Año	Autor	Notación	Método
2004	Oguz, Zinder, Ha Do, Janiak y Lichtenstein	$FHm,((PM^{(i)})_{i=1}^{(m)})/Size_{ij}/C_{\max}$	Búsqueda Tabú
2004	Thornton y Hunsucker	$FHm,((PM^{(i)})_{i=1}^{(m)})/zero-buffers/C_{\max}$	Heurística
2005	Tang, Wnxin Liu y Jiyin Liu	$FHm,((PM^{(i)})_{i=1}^{(m)})//N_T,\bar{T},\bar{F}$	Redes neuronales
2005	Janiak, Kozan, Lichtenstein y Oguz	$FHm,((PM^{(i)})_{i=1}^{(m)})//\sum\alpha_j E_j+\sum\alpha_j T_j+\sum\alpha_j W_j$	Algoritmos constructivos y metaheurísticas
2005	Logendran, deSzoeke y Barnard	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Búsqueda tabú
2005	Morita y Shio, 2005,	$FHm,((PM^{(i)})_{i=1}^{(m)})//\sum C_j$	Ramificación y acotación con algoritmos genéticos
2005	Tang, Xuan y Liu	$FHm,((PM^{(i)})_{i=1}^{(m)})//\sum w_j C_j$	Relajación lagrangiana y programación dinámica
2005	Vazquez y Salhi	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max},T_{\max},\sum w_j C_j,\sum w_j T_j$	Algoritmo genético y reglas de despacho
2005	VoB y Witt	$FH16,((QM^{(i)})_{i=1}^{(16)})//C_{\max}$	Reglas de despacho
2005	Xuan y Tang	$FHm,((RM^{(i)})_{i=1}^{(m-1)},1^m)//C_{\max}$	Relajación lagrangiana
2006	Jin, Yang, Ito	$FHm,((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Recocido simulado y búsqueda local
2006	Kyparisis y Koulamas	$FHm,((QM^{(i)})_{i=1}^{(m)})//C_{\max}$	Heurísticas
2006	Ruiz y Maroto	$FHm,((RM^{(i)})_{i=1}^{(m)})/S_{sd},M_j/C_{\max}$	Algoritmo genético

Hunsucker y Shah (1992), evalúan seis sencillas reglas de prioridad en un caso especial del FSMP, el FSMP "restringido" donde se limita el número de trabajos que se encuentran simultáneamente en el taller. Los resultados concluyen que el número máximo de trabajos en el sistema influye en el funcionamiento del mismo, siendo la regla SPT (*Short Processing Time*), la que mejor resultado tiende a dar. El número de máquinas y etapas no es significativo. Experimentos similares utilizando como criterio el retraso medio,

ofrecen un buen resultado aplicando la regla FIFO (*First In First Out*). El objetivo es minimizar el número de trabajos retrasados y minimizar la máxima tardanza.

Chandrasekharan y Chaudhuri (1992a), desarrollaron también un algoritmo de ramificación y acotación para el FSMP pero en este caso el criterio de optimización considerado es la minimización del máximo tiempo de flujo. Ante la imposibilidad de resolver problemas de unos pocos trabajos y máquinas en un tiempo razonable, los autores proponen una sencilla heurística. En un trabajo similar Chandrasekharan y Chaudhuri (1992b), aplican el estudio al mismo problema pero con el criterio de la minimización del C_{max}. Lo interesante de este trabajo es que se aplica un algoritmo de ramificación y acotación para resolver el problema de la secuenciación, es decir, se resuelve un taller de flujo estándar, para luego realizar la asignación de los trabajos en cada etapa a la primera máquina que quede disponible. Esta solución consigue reducir los tiempos CPU del algoritmo, pero a costa de la optimalidad.

Adler et al. (1993), desarrollaron un sistema llamado BPSS (*Bagpak Production Scheduling System*) para la programación de la producción en una empresa de fabricación de bolsas y sacos de papel. El sistema se basa en la identificación de cuellos de botella (*bottleneck*) en la producción y en el uso de reglas de prioridad aplicadas al caso. El sistema es muy específico para la empresa y considera tiempos de cambio de partida dependientes de la secuencia de trabajos y máquinas no relacionadas, aunque no en todas las etapas. El objetivo es minimizar la sumatoria de la tardanza ponderada.

Hunsucker y Shah (1994), estudian un problema de TFH restringido con *k* etapas mediante modelos de simulación. Se estudian el funcionamiento de seis reglas de despacho respecto al C_{max}, tiempo de flujo medio y máximo. Consideran cuatro configuraciones de taller y tres hipótesis de carga del sistema. Concluyen su estudio diciendo que la regla SPT (*Shortest Processing Time first*) es la que mejor funciona para minimizar el flujo medio en el sistema y el tiempo de finalización del último trabajo procesado, mientras que la peor es la LPT (*Largest Processing Time first*). La regla FIFO (*First In, First Out*) también proporciona buenos resultados para sistemas bastante congestionados y las tres funciones-objetivo consideradas.

Aghezzaf et al. (1995), proponen una serie de métodos para dar solución a un TFH3 con máquinas no relacionadas y tiempos de cambio de partida dependientes de la secuencia.

El problema modela la fabricación de alfombras en una empresa real. Los autores construyen un modelo de programación matemática para la secuenciación y asignación y luego descomponen estos dos problemas ya que el modelo resultante es intratable. Se desarrolla una heurística que asigna los trabajos a máquinas en primer lugar y luego se aplican heurísticas de secuenciación en problemas de una sola etapa y máquinas paralelas, de manera que cada una de las tres etapas del problema completo se resuelve independientemente de las otras.

Santos, Hunsucker y Deal (1995), no proponen algoritmos completos, sino una serie de límites inferiores para el problema FSMP con máquinas idénticas en cada etapa. Los autores hacen esta aportación para utilizar estos límites como soluciones de referencia en comparativas y desarrollos, dado que las soluciones óptimas para estos problemas son muy difíciles de obtener.

Guinet y Solomon (1996), tratan el problema FSMP de máquinas idénticas a partir de un modelo mediante programación lineal mixta del problema de TFH general con tiempos de entrega y cuyo objetivo es minimizar la tardanza máxima, proponen un procedimiento en dos fases. En la primera se ordenan los trabajos mediante varias heurísticas tales como la regla CDS (Campbell, Dudek y Smith, 1970), la heurística NEH (Nawaz, Encore y Ham, 1983), la de Townsend (1977) y una heurística de lista que utiliza la mejor regla entre la SPT (*Shortest Processing Time*), EDD (*Earliest Due Date*) y MST (*Minimum Slack Time*) (en todas ellas se utiliza el tiempo de proceso en cada etapa dividido por el número de máquinas de la etapa). En la segunda fase, el procedimiento asigna los trabajos a la máquina que minimiza su tiempo de finalización en la última máquina o el tiempo de finalización en la siguiente etapa. En el estudio se concluye que la heurística NEH es superior a las restantes en la mayoría de los problemas analizados.

Santos, Hunsucker y Deal (1996), realizan un trabajo similar a Guinet y Solomon (1996), donde la solución propuesta es la misma. Utilizan una heurística común para la secuenciación de los trabajos en el taller y luego realizan la asignación a las máquinas dentro de cada etapa siguiendo la regla de prioridad FIFO. El objetivo es minimizar el máximo tiempo de finalización.

Nowicki y Smutnicki (1998), proponen una ampliación del algoritmo de búsqueda tabú (*tabu search*) propuesto para el problema del taller de flujo estándar. Los autores

hacen uso de las propiedades del camino crítico en una secuencia para el FSMP, donde se busca minimizar el tiempo máximo de completar los trabajos. Un trabajo similar de los mismos autores apareció publicado poco después (Nowicki y Smutnicki, 1999).

Portmann et al. (1998) proponen un algoritmo de ramificación y acotación parecido al de Brah y Hunsucker pero con algunas mejoras. Se utiliza un AG hibridizado en la fase de ramificación del algoritmo para mejorar las estimaciones de los límites inferiores y superiores, de esta manera se consigue acotar más el árbol de búsqueda.

Gourgand, Grangeon y Norre (1999), proponen algoritmos heurísticos basados en simulado recocido (*simulated annealing*) para resolver el problema del TFH con máquinas no necesariamente idénticas en cada etapa. Se unen los problemas de secuenciación y de asignación, por lo que los algoritmos propuestos resuelven ambos problemas simultáneamente. Se aplica el algoritmo propuesto en un entorno industrial real. El objetivo es minimizar el C_{max}.

Brah y Loo (1999), secuencian los trabajos de acuerdo a una heurística común y luego hacen la asignación de los trabajos dentro de las etapas con una regla de prioridad. Se investiga cinco heurísticas bajo diferentes estructuras de trabajos. El criterio de optimización utilizado es la minimización del tiempo medio de flujo.

En el siguiente año Botta-Genoulaz (2000), propone seis heurísticas y enfoca su estudio en 5 etapas con máquinas paralelas idénticas con restricciones de precedencia, retraso, tiempos de finalización y otras consideraciones. El criterio de optimización es la minimización de la tardanza máxima.

Hong, C. Wang y L. Wang (2000), extienden el trabajo de Sriskandarajah y Sethi para un TFF con múltiples etapas. Proponen un algoritmo basado en LPT y el algoritmo de Gupta. El objetivo es minimizar el C_{max}.

Azizoglu, Cakmak y Kondakci (2001), desarrollan un algoritmo de ramificación y acotación para un TFF de múltiples etapas con máquinas paralelas idénticas. El objetivo es minimizar el tiempo total de flujo (*total flow time*). Realizan un experimento computacional que muestra que la propuesta es eficiente para problemas de tamaño mediano.

Kyparisis y Koulamas (2001), consideran un taller de flujo de múltiples etapas con máquinas paralelas idénticas, con el objetivo de minimizar el tiempo ponderado de

completar los trabajos (*weighted completion time*). Muestran una extensión de la heurística WSPT (*weighted shortest processing time*).

Djellab H. y Djellab K. (2002), consideran el problema del TFH*k* con restricciones de precedencia y con interrupción en el procesamiento de los trabajos (*preemption*). El objetivo es minimizar el C_{max}. Desarrollan un algoritmo polinomial para este caso especial.

Gupta, Kruger, Lauff, Werner y Sotskov (2002), consideran un TFH*k* con máquinas paralelas idénticas. El criterio de optimización es irregular, basado en la tardanza, adelanto y tiempos de finalización con costos asignados. Proponen algoritmos constructivos basados en técnicas de inserción de trabajos y algoritmos iterativos basados en búsquedas locales. Se usan experimentos computacionales para evaluar la utilidad de los algoritmos propuestos.

Wang y Li (2002), consideran el problema de taller de flujo con múltiples etapas, máquinas paralelas idénticas y con el objetivo de minimizar el C_{max}. Usan el LPT (*longest processing time*) para la asignación de los trabajos y proponen un AG para la secuenciación.

Posteriormente, Alisantoso, Khoo y Jiang (2003), presentan una solución basada en el enfoque de un algoritmo inmune (*immune algorithm*) para el problema de TFF con múltiples etapas. Es aplicado a la manufactura de tarjetas de circuitos impresos PCB (*printed circuit board*) para equipos de comunicación. Bajo el objetivo de minimizar la tardanza se compara con un AG. Después de un análisis de resultados se muestra que los algoritmos inmunes ofrecen calendarios razonables y lógicos, los cuales son una buena opción para máquinas sujetas a averías y órdenes con prisa.

Jedrzejowicz, J. y Jedrzejowicz, P. (2003), consideran un TFH*k* con máquinas paralelas idénticas. Proponen un algoritmo basado en "aprendizaje de población" (*population learning*). El objetivo es minimizar el C_{max}. Después de un experimento computacional concluyen que este tipo de algoritmos representan una efectiva metahurística para resolver este problema.

En este mismo año, Wang, Jacob y Rolland (2003), usan una red neuronal para resolver el problema de TFF de múltiples etapas con el objetivo de minimizar el C_{max}. Se muestran mejoras comparados a otras heurísticas. Su mayor eficiencia es con problemas pequeños.

Más adelante, Lee, Kim y Choi (2004), enfocan en el problema del TFH con máquinas paralelas idénticas con el objetivo de minimizar el total de tardanza de los trabajos. Sugieren un método heurístico llamado “algoritmo enfocado al cuello de botella” (*bottleneck-focused algorithm*). Es aplicado a una industria de marcos de acero. Sus resultados experimentales muestran que este método mejora algunas otras heurísticas.

También, Acero-Domínguez y Paternina-Arboleda (2004), consideran el TFF con múltiples etapas y proponen un algoritmo heurístico basado en la identificación y explotación de la etapa con cuello de botella. A través de un experimento evalúan el desempeño del método propuesto, cuyos resultados son comparables a los de otros investigadores con menos esfuerzo computacional.

En este mismo año Allaoui y Artiba (2004), abordan el TFF con restricciones de tiempos de mantenimiento, tomando en cuenta tiempos de cambio de partida, limpieza y transportación. Integran la simulación y optimización para atacar el problema. Los objetivos son minimizar el tiempo máximo de completar (*maximum completion time*), tiempo promedio de completar los trabajos (*mean completion time*), tiempo promedio de flujo (*mean flow time*), tiempo promedio de espera (*mean waiting time*), tiempo promedio de tardanza (*mean tardiness*), tiempo máximo de tardanza (*maximum tardiness*) y el número de trabajos retrasados (*number of tardy jobs*). Usan las reglas de despacho SPT, LPT y EDD. Se analizan parámetros incluyendo los tiempos de reparación de máquinas.

Engin y Döyen (2004), proponen la técnica de Sistema Inmune Artificial (*Artificial Immune System (AIS)*) para tratar el problema de TFH*k*, con el objetivo de minimizar el $C_{\max}$, Los resultados de los experimentos realizados muestran que esta técnica también es un método eficiente para resolver este tipo de problemas.

Huang, Hong, Horng y Kao (2004), presentan el problema de TFF de grupo con más de dos etapas, con igual número de máquinas en cada etapa, proponiendo un algoritmo basado en el enfoque de Palmer (1965) y LPT. Una característica es que cada trabajo pertenece a un grupo y se procesa grupo por grupo. Resultados experimentales muestran que el algoritmo propuesto resulta bastante aproximado al óptimo.

Oguz, Zinder, Do, Janiak y Lichtenstein (2004), presentan un algoritmo basado en la idea de búsqueda tabú (*tabu search*) y analizan su eficiencia y su variabilidad respecto a los datos de entrada, a través de resultados de experimentos computacionales, considerando

un TFH combinado con tareas de multiprocesamiento simultáneo. Existen restricciones de precedencia. El objetivo es minimizar el C_{max}. Se analiza la complejidad computacional del problema propuesto y se muestra que se resuelve en un tiempo polinomial.

Thornton y Hunsucker (2004), abordan el problema del TFH*k* con una o más máquinas idénticas en cada etapa. Se asume que no hay espacio de almacenamiento (*buffer*) disponible entre etapas. El criterio es minimizar el C_{max}. Se propone una nueva heurística para este caso.

Tang, Liu, W. y Liu, J. (2005), plantean una solución a un TFH dinámico con múltiples etapas, a través de redes neuronales. Se aplica como ejemplo a la industria del acero. Se considera que los trabajos aparecen de manera dinámica.

Janiak, Kozan, Lichtenstein y Oguz (2005), estudian un taller de flujo con máquinas paralelas en cada etapa, con los criterios de adelanto ponderado (*weighted earliness*), tardanza ponderada (*total weighted tardiness*) y tiempo de espera ponderado (*total weighted waiting time*). Se plantean tres algoritmos constructivos y tres metaheurísticas basadas en técnicas de búsqueda tabú y recocido simulado y se muestran algunas condiciones de su optimalidad. Se aplica en la industria de la fundición del hierro.

Logendran, deSzoeke y Barnard (2005), consideran una programación de la producción para un grupo en el contexto de tiempos de cambio de partida dependientes de la secuencia de los trabajos en un TFF con el objetivo de minimizar el C_{max}. Desarrollan tres diferentes algoritmos basados en búsqueda tabú. Resultados experimentales estadísticos revelan que estos algoritmos son eficientes para problemas grandes.

Morita y Shio (2005), consideran el TFH*k* para minimizar el tiempo de completar los trabajos. Se propone un método híbrido de ramificación y acotación con AGs, el cual mejora los métodos basados únicamente en ramificación y acotación reduciendo el número de nodos a buscar.

Tang, Xuan y Liu (2005), investigan el problema de TFH con máquinas paralelas idénticas en múltiples etapas. El objetivo es minimizar la suma de completar todos los trabajos ponderados (s*um of weighted completion times of the jobs*). Se presenta un algoritmo de relajación lagrangiana y un algoritmo de programación dinámica. Al igual que el artículo anterior, se aplica a la producción del acero. De acuerdo a sus experimentos

concluyen que los nuevos métodos generan calendarios cercanos a los óptimos en un tiempo relativamente corto.

Vazquez y Salhi (2005), reportan el desempeño de técnicas para la solución de un TFF usando cuatro variantes de AGs y reglas de despacho como FIFO, SPT, LPT y SDD (*Shorting Due Dates*), así como el SBP (*Shifting Bottleneck Procedure*). Se prueban con cuatro criterios de optimización: minimización del tiempo de completar todos los trabajos, minimización del máximo retraso (*maximum tardiness*), minimización del tiempo de completar todos los trabajos ponderados (*total weighted completion times*) y la minimización del retraso máximo ponderado (*total weighted tardiness*). Se concluye resaltando el notable desempeño de la inclusión de los AGs.

VoB y Witt (2005), consideran un TFH con 16 etapas aplicado a un caso real en una compañía de acero. Se asumen máquinas paralelas uniformes, restricciones de asignación de acuerdo a los trabajos, saltos para los trabajos en ciertas etapas, no disponibilidad de las máquinas, tiempos de cambio de partida dependientes de la secuencia. Se prioriza minimizar el retraso máximo. Se plantea como un problema de lotes. Se aplican reglas de despacho y se extiende el conocido modelo RCPSP (*Resource Constrained Project Scheduling Problem*).

Xuan y Tang (2005), enfocan su investigación en un caso especial de TFH*k* y una producción en lotes en la última etapa, aplicado a la industria del hierro y el acero. Se basan en la técnica de relajación lagrangiana. A través de descomponer en subproblemas de lotes diseñan un algoritmo para problemas de tamaño grande. El objetivo es minimizar un criterio dado con respecto al tiempo de completar todos los trabajos.

Más recientemente, Jin, Yang e Ito (2006), consideran el problema de TFH*k* para minimizar el $C_{\max}$. El procedimiento de optimización se basa en algoritmos metaheurísticos de recocido simulado y en una generalización de búsqueda local (*local search*) llamada VDS (*variable-depth search*). Presentan una serie de nuevos límites inferiores. Muestran la eficiencia del procedimiento a través de experimentos computacionales para dos, tres y cinco etapas.

Kyparisis y Koulamas (2006), consideran un TFF con múltiples etapas y máquinas paralelas uniformes en cada etapa. El objetivo es minimizar el $C_{\max}$. Se desarrolla una clase general de heurísticas con algunas extensiones de otras heurísticas anteriores para el

problema de taller de flujo serial. Se muestra que una de las heurísticas propuestas es asintóticamente óptima.

Por último, Ruiz y Maroto (2006), proponen AGs para un TFH con máquinas no relacionadas, restricciones de elegibilidad de máquinas en cada etapa y tiempos de cambio de partida dependientes de la secuencia. Presentan adaptaciones exitosas de otras metaheurísticas recientes y manejan algunos experimentos con un conjunto de 1320 ejemplos aleatorios y datos reales tomados de compañías que se dedican a la producción de losetas.

3.4 Conclusiones del Capítulo

Las investigaciones en los primeros años de estudio del TFH se inician dando un mayor énfasis en problemas de dos etapas; sin embargo, conforme ha transcurrido el tiempo el número de propuestas para el TFH de múltiples etapas se ha incrementado considerablemente. El criterio de optimización más estudiado es la minimización del C_{max}, sin embargo existen estudios en la minimización del tiempo de flujo, minimización del retraso máximo, minimización de costos asociados a la tardanza, minimización del número de trabajos retrasados. Las configuraciones de máquinas más frecuentes en las publicaciones son paralelas idénticas. Los enfoques más empleados para resolver este problema ha sido el de ramificación y acotación, recocido simulado, AGs y diversas heurísticas.

Entre las industrias donde se ha aplicado esta teoría, se encuentran la industria del acero, electrónica, cerámica, papelera y computación. Sin embargo, lo manifestado por MacCarthy y Liu (1993) y McKay, Allahverdi, Gupta y Aldowaisan (1999), Yang y Liao (1999), Linn y Zhang (1999), Vignier, Billaut y Proust (1999), Cheng, Gupta y Wang (2000), Pinedo y Webster (2002) y Ruiz y Maroto (2006), sigue siendo una realidad: aunque se ha investigado mucho en esta área, todavía existe una brecha importante entre la teoría y las aplicaciones prácticas. Lo rescatable es que muchos modelos teóricos estudiados sirven como base para modelos más complejos. Es deseable que un número mayor de las investigaciones futuras se fundamenten en modelos basados en situaciones reales.

Los métodos que se aplican al problema del TFH, son: óptimos eficientes (extensiones de la regla de Johnson), métodos de enumeración (búsqueda con retroceso, ramificación y acotación, programación dinámica), métodos heurísticos, entre los cuales se tienen los métodos constructivos (voraces, de despacho), métodos de descomposición, métodos basados en índices, de aplicación limitada, redes neuronales; así mismo, métodos metaheurísticos, como, algoritmos GRASP, recocido simulado, búsqueda tabú y algoritmos genéticos. La revisión de los métodos (apartado 2.5), permite visualizar la idoneidad de los mismos, al identificar cuáles métodos han tenido mejores resultados de acuerdo al tipo de problema.

Como se observa en esta revisión del estado del arte, no existe una publicación que contemple las características del ambiente productivo que se abordan en esta tesis de manera integral; por lo cual se concluye que esta situación real no ha sido estudiada de acuerdo a las características del entorno productivo mencionadas en la sección 1.2. La aportación de Ruiz y Maroto (2006) representa una contribución cercana al modelo en estudio. Por los resultados obtenidos en esta publicación, se deduce que los algoritmos genéticos tienen un alto desempeño en problemas de gran complejidad, como es el TFH con máquinas no relacionadas, tiempos de ajuste dependientes de la secuencia y elegibilidad de máquinas.

Capítulo 4

Algoritmos heurísticos basados en el método de dicotomía para el TFH con dos etapas

4.1 Taller de flujo híbrido con dos etapas

Se considera un sistema de recursos con dos grupos de máquinas. Se supone que las máquinas en cada grupo son idénticas, con capacidades y velocidades iguales. Es necesario procesar un conjunto $N = \{1, 2, ..., n\}$ de trabajos. Cada trabajo *j*, $j = \overline{1,n}$, consiste de dos tareas u operaciones: O_j^1 y O_j^2. La primera operación O_j^1 se ejecuta por una de las *M* máquinas del primer grupo durante p_j^1 unidades de tiempo. Después de su terminación se realiza la operación O_j^2, la cual tiene una duración p_j^2, en una de las *m* máquinas del segundo grupo. De tal manera que a cada trabajo *j* le corresponde un par ordenado $p_j = \{p_{Ij}^1, p_{ij}^2\}$, donde $I \in \{1, ..., M\}$ y $i \in \{1, ..., m\}$. 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. No se permiten interrupciones en las operaciones. Todos los trabajos están listos para su ejecución en el momento 0, lo que significa que el instante de entrada es $r_j = 0$, $\forall j = \overline{1,n}$. Es necesario asignar un 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 (TFF2), en cada unas de las cuales se tiene un grupo de máquinas idénticas. Para los dos algoritmos heurísticos presentados en este capítulo (HA_{M+m}, HA_{1+m}), el problema se descompone en dos subproblemas:

1) Asignación (*Assignment*) de trabajos a las máquinas tomando en cuenta las duraciones de las operaciones.
2) Ordenación (*Sequencing*) de los trabajos en cada máquina de tal manera que el tiempo total de ejecución de los trabajos sea mínimo.

Resolver el problema de asignación significa indicar la máquina M_I del primer grupo, $I \in \{1,...,M\}$, y la máquina m_i del segundo grupo, $i \in \{1,...,m\}$, para realizar las operaciones del trabajo *j*, $j = \overline{1,n}$, esto es, determinar la ruta tecnológica de las dos operaciones sucesivas R_j, $R_j \in \{M \times m\}$.

En la manufactura es común que no haya una asignación específica de los pedidos a las máquinas, en las cuales deben ejecutarse, siendo éstas idénticas. Existe una variedad de heurísticas, llamadas "reglas de prioridad" o de despacho (apartado 2.5.2), aplicables a la asignación en este tipo de problemas. Entre estas están (Pinedo, 2002):

SPT S*hortest Processing Time first* (en primera etapa) & *minload*,

SPTB *Shortest Processing Time firs 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*.

En (Romero, 2007) se analiza, cómo afecta cada regla de asignación a $C_{\max}$ en el algoritmo HA_{M+m}, con el propósito de calibrar el algoritmo respecto al algoritmo de asignación.

Una vez ejecutado, el procesamiento de asignación, el modelo TFF se transforma en un caso de TFH con elegibilidad de máquinas no relacionadas y con tiempos de procesamiento p_{Ij}^1 y p_{ij}^2 para la primera y segunda etapa respectivamente. De esta manera, los algoritmos, después de un procesamiento de asignación de trabajos a máquinas, son aplicables tanto para un TFF, como para un TFH.

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 representa un sistema de recursos con dos grupos de máquinas no relacionadas en paralelo, con $m_1 = M$ máquinas en la primera etapa y con $m_2 = m$ máquinas en la segunda. Se requiere obtener el orden de los trabajos en las máquinas, que proporcione el menor tiempo posible de ejecución: $C_{\max} \to \min$. La solución se busca entre las permutaciones de trabajos.

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

1) $1+m$: una máquina en la primera etapa y varias máquinas paralelas en la segunda: $FH2,((1^{(1)}),(RM^{(2)})) \mid prmu \mid C_{\max}$, (Figura 4.1); (4.1)

2) $M+m$: varias máquinas paralelas en ambas etapas (caso general):

$FH2,(RM^{(i)})_{i=1}^{(2)} \mid prmu \mid C_{\max}$ (Figura 4.2); (4.2)

3) $M+1$: varias máquinas paralelas en la primera etapa una máquina en la segunda:

$FH2,((RM^{(1)}),(1^{(2)})) \mid prmu \mid C_{\max}$ (Figura 4.3). (4.3)

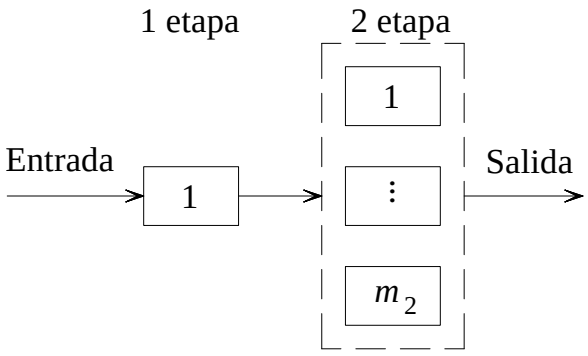


Figura 4.1 TFH2, una máquina en la primera etapa y m_2 máquinas en la segunda

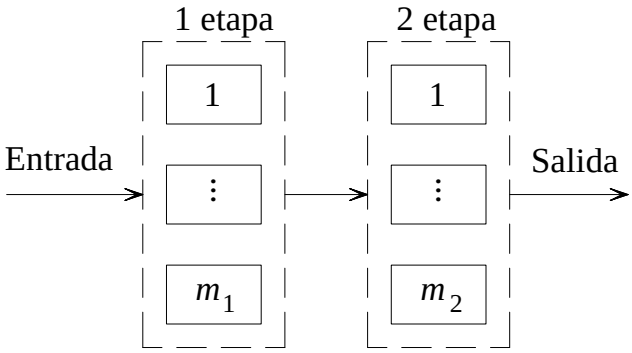


Figura 4.2 TFH2, m_1 máquinas en la primera etapa y m_2 máquinas en la segunda

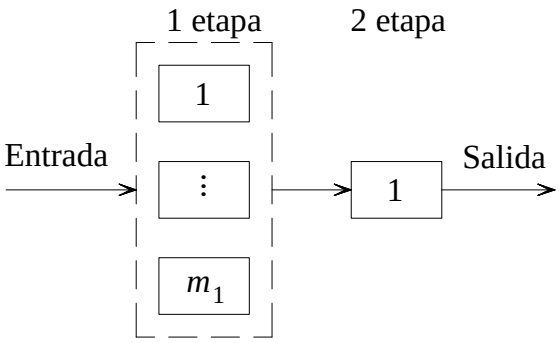


Figura 4.3 TFH2, m_1 máquinas en la primera etapa y 1 máquina en la segunda

El caso particular de un TFH2, cuando $m_1 = m_2 = 1$, es conocido como el problema de Johnson para dos máquinas $F2 \| C_{\max}$. La regla de Johnson encuentra la solución óptima cuando un conjunto determinado de trabajos se procesa sucesivamente en dos máquinas (Johnson, 1954).

El TFH para el caso con sólo dos etapas fue examinado por primera vez por Arthanary y Ramaswamy (1971). Proponen un algoritmo del tipo ramificación y acotamiento.

El problema de minimización de $C_{\max}$ en un TFH2 pertenece a los **NP**-difíciles. Es **NP**-completo cuando el número de máquinas en una de las etapas, m_1 ó m_2, es cualquier valor arbitrario, mientras $\max(m_1, m_2) > 1$ (Gupta, 1988). Por lo tanto, sólo en algunos casos especiales se suele encontrar soluciones óptimas con algoritmos polinomiales o pseudopolinomiales. La solución al problema en general se busca con métodos

aproximados. En el apartado 3.1 se hace una revisión del estado del arte para este tipo de problemas.

En esta tesis se proponen dos algoritmos heurísticos para este problema: 1) HA_{1+m} para el caso (4.1) y 2) HA_{M+m} para el caso (4.2). El último caso (4.3) se resuelve con el algoritmo HA_{M+m} como un caso particular.

Ambos algoritmos usan la dicotomía como esquema externo para su acercamiento al óptimo. A continuación se presentan las implicaciones del modelo:

- El número de máquinas es $M \geq 1,\ m \geq 1$; y el número de trabajos es $n > M$ y $n > m$.
- Todos los trabajos se encuentran disponibles para su ejecución en el momento 0, $\therefore r_j = 0,\ \forall j \in J$.
- Las máquinas están 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 de la ruta tecnológica r_j, ésta debe procesarse hasta su finalización.
- Los trabajos son independientes entre sí, es decir, no hay relaciones de precedencia de ningún tipo entre los trabajos.
- Los tiempos de ajuste no dependen de la secuencia de los trabajos y son incluidos en los tiempos de procesamiento de las operaciones.
- Se asume una capacidad de almacenamiento ilimitada entre las máquinas. Nunca un trabajo queda bloqueado entre dos máquinas.
- Los tiempos 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 no significativos.

4.2 Aplicación de dicotomía para la optimización combinatoria

4.2.1 Método

El concepto de dicotomía tiene aplicaciones en varias áreas matemáticas. Se utiliza en el método de bisección para la búsqueda de raíces de ecuaciones algebraicas y en la búsqueda binaria, la cual es uno de algoritmos básicos de matemática discreta. Representa la división sucesiva de un intervalo inicial en dos partes con el propósito de localizar la raíz de una ecuación ó un valor específico 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 programación de la producción (Yaurima, et al., 2006). En este caso, la dicotomía representa un método que integra la naturaleza numérica de bisección con la naturaleza discreta de la búsqueda binaria. A continuación se presenta un análisis de los métodos basados en el concepto de dicotomía y la forma en la cual se aplica en la optimización combinatoria.

El método de bisección se basa en el Teorema de los Valores Intermedios (Burden, et al., 1985).

Teorema de Bolzano. Sea f una función continua en cada punto del intervalo cerrado $[a,b]$ y $f(a)$ y $f(b)$ tienen signos opuestos, es decir, $f(a)\cdot f(b)<0$. Entonces existe al menos un $m\in(a,b)$ tal que $f(m)=0$.

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

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, de acuerdo al intervalo donde ocurre un cambio de signo.

El método aplica el concepto de dicotomía mediante un proceso infinito que consiste en ir estrechando de manera sistemática el intervalo, en el cual una función continua cambia de signo. Al definir la exactitud deseada $\varepsilon>0$ de la solución, se restringe el número de iteraciones a un valor finito y se obtendrá un intervalo $[b_i,a_i]$ arbitrariamente

pequeño, que contenga la solución, es decir, $(b_i - a_i) \leq \varepsilon$. La Figura 4.4 ilustra el procedimiento descrito.

El método de búsqueda binaria representa un método común en la programación matemática. Se aplica para localizar un elemento *a* en un arreglo finito $X = \{x_1, \ldots, x_{n/2}, \ldots, x_n\}$ de elementos ordenados en forma creciente. El arreglo de elementos en orden creciente significa una estrategia de estrechamiento del área de ubicación del elemento buscado.

El método consiste en lo siguiente (Rosen, 2004):

Se realiza la comparación del valor del objeto *a* con el valor del elemento central $x_{n/2}$ del arreglo. Entonces se aplica el concepto de dicotomía para partir el arreglo en dos subarreglos, que tienen igual número de elementos. Si el valor $a < x_{n/2}$, la búsqueda continúa con el subarreglo de elementos $X^1 = \{x_1, \ldots, x_{n/2}\}$. En caso contrario ($x_{n/2} \leq a$) se realizará la búsqueda en $X^2 = \{x_{n/2+1}, \ldots, x_n\}$. Este proceso es iterativo finito con intervalos cada vez dos veces menores hasta que se llega a un intervalo indivisible, donde se encuentra el elemento buscado.

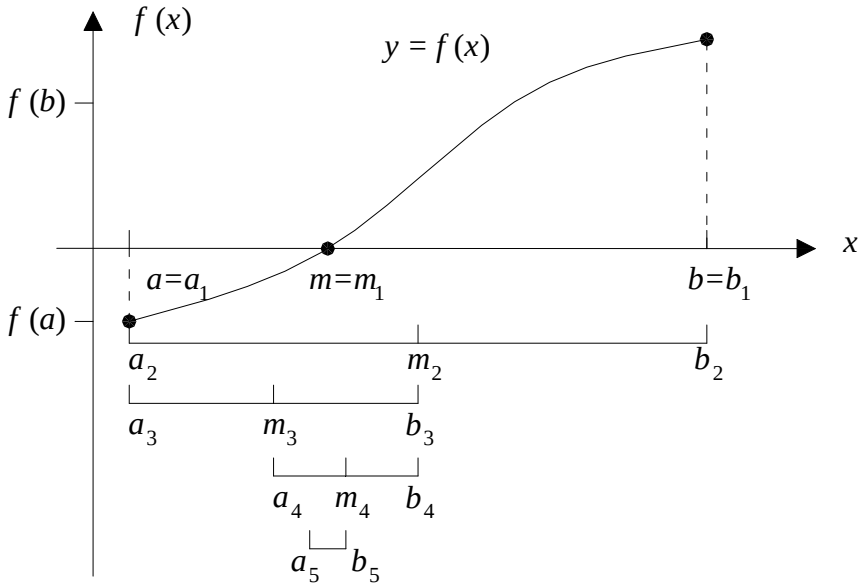


Figura 4.4 Esquema del método de bisección

Ejemplo 4.1 Sea el arreglo $X = \{1,2,3,4,5,6,8,9,10,11,12\}$. Se desea localizar el elemento $a = 8$. La Figura 4.5 representa el árbol binario para el caso.

La dicotomía, aplicada a la optimización combinatoria, representa una combinación del método numérico de la bisección y el discreto de la búsqueda binaria. A continuación se enseñan los detalles del método y se analiza su complejidad.

4.2.2 Convergencia

Sea la funcional[1] $C(x)$, 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 problema C^1.

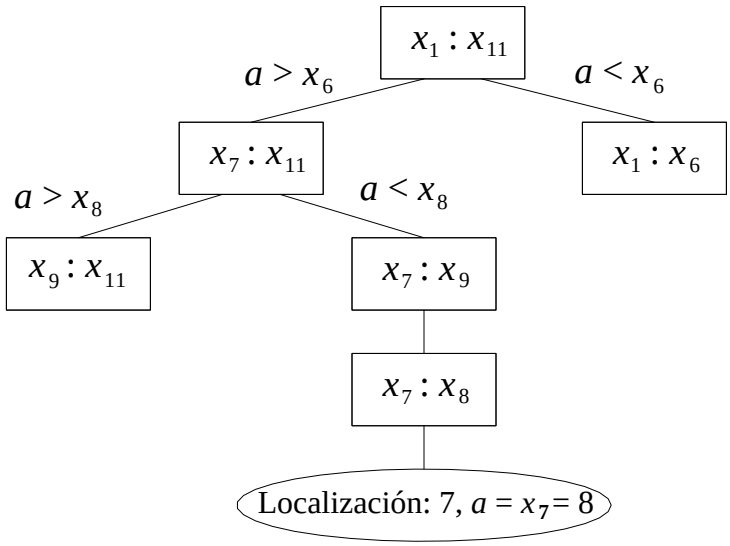


Figura 4.5 Árbol binario

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

Problema C^1: entre todo elemento del conjunto *P* encontrar un elemento $x_1 \in P$, tal que $C(x_1) \le C^1$, donde $C^1 = \frac{C^0_{\min} + C^0_{\max}}{2}$.

Si tal elemento existe, entonces:

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

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

En 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 *s* de la búsqueda de *x**, incluye:

1) El cálculo $C^s = \frac{C^{s-1}_{\min} + C^{s-1}_{\max}}{2}$;

2) La determinación del elemento x_s, tal que $C(x_s) \le C^s$ (si el problema C^s es resoluble).

3) La determinación de los límites $C^s_{\min}$ y $C^s_{\max}$ que cumplen las siguientes condiciones:

$$C^s_{\min} = \begin{cases} C^{s-1}_{\min}, \text{ si el problema } C_s \text{ tiene solución} \\ C^s, \text{ si el problema } C_s \text{ no tiene solución} \end{cases}$$

$$C^s_{\max} = \begin{cases} C^s, \text{ si el problema } C^s \text{ tiene solución} \\ C^{s-1}_{\max}, \text{ si el problema } C^s \text{ no tiene solución} \end{cases}$$

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

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

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

4.2.3 Complejidad

La complejidad temporal del método de dicotomía, se basa en los Teoremas 4.1 y 4.2 para la búsqueda binaria (Aho, et al., 1988).

Teorema 4.1. La búsqueda del elemento x_m, el cual es igual a *x*, en un conjunto de números $x_1 < x_2 < ... < x_n$ requiere $O(\log n)$ operaciones elementales.

La demostración del teorema consiste en la imposibilidad de construir un algoritmo con complejidad menor a $O(\log n)$. Esta conclusión implica el siguiente teorema:

Teorema 4.2. No existe un algoritmo, el cual requeriría un 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 óptimo $C(x^*)$, y de la complejidad temporal del problema C^s. Por tanto, para minimizar el número de operaciones calculatorias, es importante encontrar los límites superior e inferior más exactos de la funcional a minimizar.

Las dificultades principales relacionadas con el uso del método de dicotomía provienen de la complejidad interna del problema C^s, 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, a continuación se formula el problema de empaquetamiento, conocido en la optimización combinatoria. Este problema y sus casos particulares tienen muchas aplicaciones en la práctica, como la asignación de trabajos a 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 N_i, $i = \overline{1,m}$, tales que

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

Supongamos que existe un algoritmo polinomial para la resolución del problema *C*, el cual requiere el tiempo $t(d)$ para el tamaño de cadena de entrada *d*. 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:

$C(x^*) \le C \le \sum_{j=1}^{n} p_j$, lo cual implica que $\log C \le \log \sum_{j=1}^{n} p_j$;

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

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

$\sum_{j=1}^{n} \log p_j \le d$, ó bien,

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

Entonces,

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

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

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

y un algoritmo para la resolución del problema de optimización aplicando el método de dicotomía tiene la complejidad temporal $O(t'(d))$, donde $t'(d) = dt(d)$. Como $t(d)$ es un polinomio, entonces $dt(d)$ también lo es.

Con esta razón se llega a 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 problema *C*. Independientemente de la complejidad temporal del algoritmo para la resolución del problema *C*, los pasos de dicotomía 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 programación de la producción donde se utiliza como esquema externo del algoritmo.

4.3 Algoritmo heurístico para la resolución del problema de TFH2 con una máquina en la primera etapa

4.3.1 Modelo

Se considera un sistema de recursos con una máquina en la primera etapa y múltiples máquinas en la segunda. Para todo trabajo *j*, $j = \overline{1,n}$, $n > m$, está definida su ruta tecnológica R_j, $R \in \{1, i\}$, $i = \overline{1,m}$, como una sucesión de dos tareas, u operaciones. La primera operación de un trabajo *j* es realizada por la máquina de la primera etapa durante p_j^1 unidades de tiempo. La segunda operación de duración p_j^2 es asignada a la máquina *i*, $i \in \{m\}$, de la segunda etapa. Así, a todo trabajo *j*, corresponde un par ordenado $p_j = \{p_j^1, p_j^2\}$. Las implicaciones del modelo son enumerados en el apartado 4.1. Es necesario determinar un orden de los trabajos que garantice un tiempo mínimo de ejecución.

En el caso considerado, los *n* trabajos están pre-asignados a las máquinas, lo que implica que las *m* máquinas de la segunda etapa no son idénticas, o existe una elegibilidad completa de las máquinas. En otras palabras, el conjunto de trabajos $N = \{j_l \mid 1 \le l \le n\}$ está partido en *m* disjuntos N_i, donde N_i es un subconjunto de los trabajos, cuya segunda

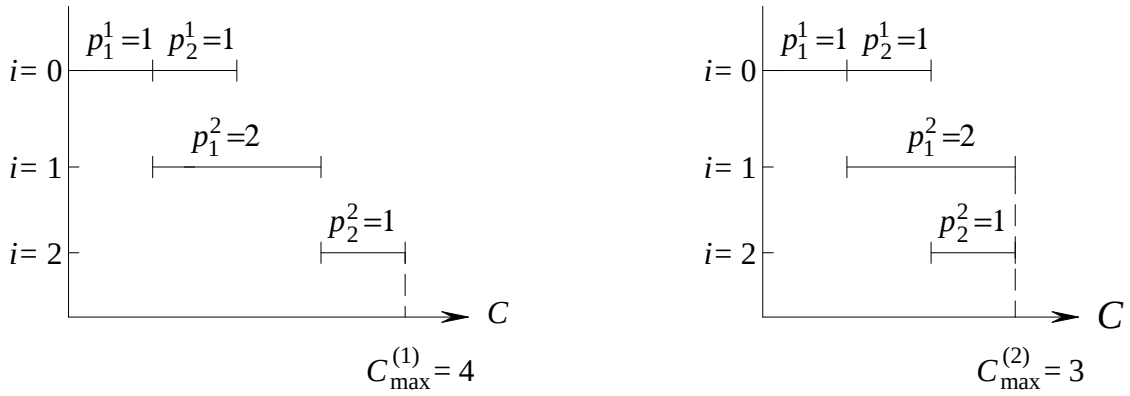
operación está asignada a la máquina i del segundo nivel, $i \in \{1,2,...,m\}$; $N = \bigcup_{i=1}^{m} N_i$, $|N| = n$, $N_i \neq \varnothing$, $N_\mu \cap N_\nu = \varnothing$, $\mu \neq \nu$.

Este problema representa un caso particular de un TFH2 con una máquina en la primera etapa y m máquinas paralelas diferentes en la segunda etapa. Este caso también es considerable como un taller de flujo con $1+m$ etapas sucesivas, cuya solución es una permutación (*permutation flowshop*). Para la demostración de la última afirmación, a la única máquina de la primera etapa se asigna el número 0, y a las máquinas de la segunda etapa, se asignan los números 1, 2, …, m. Ahora todo trabajo j_l, $l = \overline{1,n}$, representaremos en forma de una sucesión de $1+m$ operaciones $\{O_{0l}, O_{1l}, ..., O_{il}, ..., O_{ml}\}$. Para cualquier índice l, el tiempo de ejecución de la operación O_{0l} es p_l^1. El tiempo de ejecución de la operación O_{il} es p_l^2 si $j_l \in N_i$, $i \in \{m\}$; las demás operaciones tienen duración cero. En tal caso, el sistema con $1+m$ máquinas representa un taller de flujo simple, puesto que la operación O_{0l} del trabajo con el índice l debe ejecutarse en la máquina 0, y la operación O_{il}, en la máquina i, después de terminar la operación $O_{(i-1)l}$ en la máquina $i-1$.

A diferencia del caso general de un taller de flujo con $1+m$ etapas sucesivas en serie, aquí, de acuerdo con las condiciones de funcionamiento del sistema, el trabajo j_l, representado por la sucesión de $1+m$ operaciones, no llega a la máquina i, si la duración de la operación O_{il} es cero.

Ejemplo 4.2. Para $j_1 \in N_1$, $j_2 \in N_2$, $p_1^1 = p_2^1 = p_2^1 = 1$, se deben ordenar dos trabajos $j_1 = (1,2,0)$ y $j_2 = (1,0,1)$ en 3 máquinas en serie.

Si cada trabajo llega para su ejecución en la primera, segunda y tercera máquina sucesivamente, el tiempo mínimo es $C_{\max}^{(1)} = 4$ (Figura 4.6a). Sin embargo, cuando el trabajo j_2 no llega a la segunda máquina, el valor buscado para la misma sucesión de trabajos es $C_{\max}^{(2)} = 4$ (Figura 4.6b).



(a) (b)

Figura 4.6 Gráfica de Gantt para un taller de flujo simple con $1+m$ etapas:
a) Cada trabajo llega para su ejecución en la primera, segunda y tercera máquinas sucesivamente;
b) el trabajo j_2 no llega a la segunda máquina.

De acuerdo con el análisis anterior, a continuación se muestra la descripción formal del problema.

Una matriz con $1+m$ renglones 0, 1, 2, ..., *m*, y *n* columnas 1, 2, ..., *n* corresponde a una sucesión arbitraria $\pi=(j_1, j_2,..., j_n)$ de los componentes $j_l=(p_{j_l}^1, p_{j_l}^2)\in N$, $|N|=n$. Los elementos de la matriz de duraciones de operaciones $b_{0\,j_l}$, $b_{0\,j_l}$, $i=\overline{1,m}$, $l=\overline{1,n}$, se definen de la siguiente manera:

$$b_{0\,j_l}=p_{j_l}^1;$$

$$b_{ij_l}=\begin{cases} p_{j_l}^2 & \text{si } j_l\in N_i; \\ 0, & \text{en otros casos.}\end{cases}$$

Ejemplo 4.3. Sea el sistema de recursos $1+2$ máquinas, y sean los trabajos $j_1=(1,4)$, $j_2=(5,9)$, $j_3=(3,6)$, $j_4=(5,5)$, $j_5=(4,10)$, $j_6=(8,7)$; $N=\bigcup_{i=1}^{2} N_i$, donde $N_1=\{j_1, j_2, j_3\}$ y $N_2=\{j_4, j_5, j_6\}$. La matriz de duraciones de operaciones para este ejemplo es:

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left[\begin{array}{cccccc} 1 & 5 & 3 & 5 & 4 & 8 \\ 4 & 9 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 10 & 7 \end{array}\right] \end{array}$$

Se busca la permutación π^* del conjunto $\{1,2,...,n\}$ que minimice el valor de la funcional

$$C_{\max}(\pi) = \max_{1\le i\le m}\max_{1\le l\le n}\left(\sum_{k=1}^{l} b_{0j_k} + \sum_{k=l}^{n} b_{ij_k}\right) = \max_{1\le i\le m}\max_{1\le l\le n}\left(\sum_{k=1}^{l} p^1_{j_k} + p^2_{j_l} + \sum_{k=l+1}^{n} b_{ij_k}\right) \quad (4.4)$$

4.3.2 Evaluación de límites

Es evidente que para una permutación óptima π^* se cumple la desigualdad

$$\sum_{j=1}^{n} p^1_j + \min_{1\le j\le n} p^2_j \le C(\pi^*) \le C(\pi_0)$$

A continuación se describe el algoritmo heurístico HA_{SI} basado en la regla de Johnson para la construcción de una solución inicial π_0 del problema (4.4) que se toma como un límite superior.

Algoritmo HA_{SI}

Paso 1. Para todo subconjunto $N_i = \{j_{i_k} \mid k = \overline{1,n_i}, i = \overline{1,m}\}$ se busca la permutación $\sigma_i = (i_1, i_2, ..., i_{n_i})$ que minimice la función:

$$L(\sigma_i) = \max_{1\le k\le n_i}\left(\sum_{r=1}^{k} p^1_{i_r} + \sum_{r=k}^{n_i} p^2_{i_r}\right).$$

En el resultado, para cada subconjunto N_i, $i = \overline{1,m}$, se crea la permutación $\sigma_i = (i_1, i_2, ..., i_{n_i})$ que representa la solución óptima del problema de Johnson $2\times n$ con las duraciones de las operaciones $p^1_{i_r}$ y $p^2_{i_r}$, $r = \overline{1,n_i}$.

Paso 2. Se calcula el peso del componente situado en la permutación σ_i, $i = \overline{1,m}$, $s = \overline{1,n_i}$, en la posición *s* del lado izquierdo:

$$F_{i_s} = \sum_{r=s}^{n_i} p_{i_r}^2 \ .$$

Paso 3. El conjunto *N* se acomoda en el orden decreciente de F_{i_s}, $i = \overline{1,m}$, $s = \overline{1,n_i}$. Se obtiene la sucesión de los *n* valores de F_{i_s} que coincide con la sucesión de los componentes $j \in N$, $j = \overline{1,n}$, en la permutación π_0.

Los elementos de la permutación π_0 se enumeran sucesivamente. Ahora $\pi_0 = \{1, 2, ..., n\}$. El peso correspondiente al elemento, el cual pertenece al subconjunto N_i, se denota por F_{ij} y se sitúa en la posición *j* de π_0. El tiempo de ejecución de los trabajos, el cual corresponde a π_0 se determina según la fórmula:

$$C(\pi_0) = \max_{1 \le q \le n} \left(\sum_{j=1}^{q} p_j^1 + F_{ij} \right) = \max_{1 \le q \le n} \left(\sum_{j=1}^{q} b_{0j} + \sum_{j=q}^{n} b_{ij} \right), \ i \in \{1, 2, ...m\}.$$

(4.5)

A continuación, el algoritmo se ilustra mediante un ejemplo.

Ejemplo 4.4. Sea un sistema de recursos con $1+2$ máquinas. El conjunto de trabajos es: $j_1 = (1,4)$, $j_2 = (5,9)$, $j_3 = (3,6)$, $j_4 = (5,5)$, $j_5 = (4,10)$, $j_6 = (8,7)$; $N = \bigcup_{i=1}^{2} N_i$, $N_1 = \{j_1, j_2, j_3\}$ y $N_2 = \{j_4, j_5, j_6\}$. Se busca una solución inicial π_0 del problema (4.4).

Los pasos del algoritmo son los siguientes.

1. Aplicando la regla de Johnson para los subconjuntos N_1 y N_2, se obtienen dos permutaciones: $\sigma_1 = \{j_1, j_3, j_2\}$, $\sigma_2 = \{j_5, j_4, j_6\}$. Los valores de $L(\sigma_i)$, $i = \overline{1,2}$ son:

$L(\sigma_1) = \max(1+19, 4+15, 9+9) = 20$;

$L(\sigma_2) = \max(4+22, 9+12, 17+7) = 26$.

2. Se calculan los pesos para todo elemento del conjunto *N*, según su posición en la permutación σ_1 o σ_2:

$F_{1_1} = 4+6+9 = 19$;

$F_{1_2} = 6+9 = 15;$

$F_{1_3} = 9;$

$F_{2_1} = 10+5+7 = 22;$

$F_{2_2} = 5+7 = 12;$

$F_{2_3} = 7.$

3. Los pesos F_{i_s} se colocan en orden decreciente. Se obtiene la permutación $\pi_0 = \{j_5, j_1, j_3, j_4, j_2, j_6\}$.

La gráfica de Gantt de la solución del problema que corresponde a la permutación π_0 se muestra en la (Figura 4.7).

La matriz de duración de operaciones para tal permutación es:

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccc} 5 & 1 & 3 & 4 & 2 & 6 \end{array} \\ \begin{bmatrix} 4 & 1 & 3 & 5 & 5 & 8 \\ 0 & 4 & 6 & 0 & 9 & 0 \\ 10 & 0 & 0 & 5 & 0 & 7 \end{bmatrix} \end{array}$$

El tiempo de ejecución de los trabajos según el orden definido por la permutación π_0 es:

$$C(\pi_0) = \max(4+22, 5+19, 8+15, 13+12, 18+9, 26+7) = 33.$$

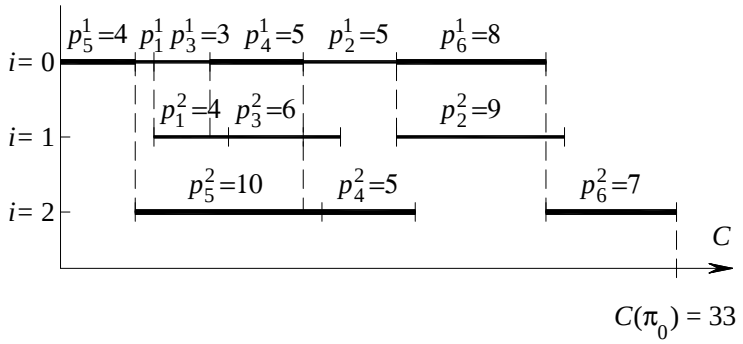


Figura 4.7 Gráfica de Gantt para el procesamiento de los trabajos en la sucesión π_0.

Ahora se evalúa el tiempo necesario para la construcción de la solución inicial $C(\pi_0)$. La obtención de los σ_i , $i=\overline{1,m}$, ocupa el tiempo $O(n_i \log_2 n_i)$. La ejecución de los *m* procedimientos de Johnson requiere $O(n\log_2 n)$. Para encontrar un valor de F_{i_s}, $i=\overline{1,m}$, $s=\overline{1,n_i}$, es necesario ejecutar n_i operaciones de comparación. Por lo tanto, el tiempo de cálculo de los *n* valores F_{i_s} se evalúa como $O(n)$. La mayor parte de los cálculos ocupa un ordenamiento de *n* pesos F_{i_s}, por eso la complejidad temporal del algoritmo de construcción de π_0 se evalúa como $O(n\log_2 n)$.

4.3.3 Algoritmo HA_{1+m}

El algoritmo descrito a continuación funciona de acuerdo con el esquema del método de dicotomía, el cual requiere la formulación de (4.4) como un problema de decisión:

Dada una partición del conjunto *N* de los componentes $j_l=(p_l^1, p_l^2)$, $l=\overline{1,n}$, $p_l^1, p_l^2 \in Z^+$, en *m* no vacíos disjuntos N_i, $i=\overline{1,m}$. Se requiere determinar ¿existe para un límite $C^s \in Z^+$ tal permutación π_s de *n* componentes j_l que $C(\pi_s) \le C^s$? (Yaurima, et al., 2006).

El siguiente algoritmo en la iteración *s* construye la permutación π_s de longitud $C(\pi_s) \le C^s$ si π_s existe, y viceversa: el algoritmo no encuentra la permutación π_s para la cual se cumple la condición $C(\pi_s) \le C^s$, si ésta no existe.

El valor de C^s se concluye en un intervalo $(C_{\min}, C(\pi_0)]$, donde

$$C_{\min} = \max\left[\max_{1\le i\le m} L(\sigma_i), \sum_{j=1}^{n} p_j^1 + \min_{1\le j\le m} p_j^2\right] - 1 .$$

Para el ejemplo considerado $C_{\min} = [20, 26, 26+4] - 1 = 29$.

El procedimiento de la construcción de π_s incluye los siguientes pasos.

Algoritmo HA_{1+m}

Paso 1. El conjunto N contiene n componentes $j_k = (p_k^1, p_k^2)$, H^s es una lista vacía, $N^s = N \setminus H^s$, $A = \sum_{r=1}^{n} p_r^1$, $F_i = 0$, $i = \overline{1,m}$; $l = n$.

Paso 2. Se fija $k = l$. Los componentes del conjunto N^s de manera arbitraria son enumerados con los números 1, 2, ..., k.

Paso 3. Si para cada p_r^2, $r = \overline{1,k}$, se cumple la desigualdad

$$C^s < A + p_r^2 + F_i,\ j_r = (p_r^1, p_r^2) \in N_i, \tag{4.5}$$

entonces fin: no existe tal permutación π_s que $C(\pi_s) \le C^s$.

Paso 4. Entre todos los elementos del conjunto N^s que no cumplan la desigualdad (4.5) se busca un elemento $j_l = (p_l^1, p_l^2)$ con el valor máximo de p_l^1. El elemento elegido se ubica en la posición l de la permutación buscada π_s y ocupa la primera posición en la lista $H^s = \{j_l\} \bigcup H^s$, $N^s = N^s \setminus \{j_l\}$, $A = A - p_l^1$; $F_i = F_i + p_l^2$ si $j_l = (p_l^1, p_l^2) \in N_i$; $l = l - 1$.

Paso 5. Si $l > 0$, entonces ir al Paso 2. En caso contrario, está construida la permutación π_s que cumple la desigualdad $C(\pi_s) \le C^s$.

Una búsqueda exitosa termina en el Paso 3 después de la determinación de un elemento que ocupa la primera posición en π_s. Los Pasos 3 y 4 representan la base del algoritmo. En el Paso 3 el algoritmo elige los elementos del conjunto N^s que pretenden ocupar la posición l en la permutación buscada π_s. Las operaciones posteriores llevan a la elección de un elemento único dirigido a ocupar la posición l en π_s. Cuando varios elementos tienen un valor máximo igual de p_l^1, la lista H^s incluye cualquiera de éstos. El algoritmo para en el Paso 3, cuando el problema de construcción de π_s para un límite fijo no tiene solución.

Recordando el Ejemplo 4.4: $j_1 = (1,4)$, $j_2 = (5,9)$, $j_3 = (3,6)$, $j_4 = (5,5)$, $j_5 = (4,10)$, $j_6 = (8,7)$; $N_1 = \{j_1, j_2, j_3\}$ y $N_2 = \{j_4, j_5, j_6\}$.

Según los cálculos anteriores, el límite inferior es $C_{\min} = 29$, y el límite superior es $C(\pi_0) = 33$; La solución óptima $C(\pi^*) \in (29, 33]$.

La primera iteración del algoritmo: $s = 1$

Se busca la solución que no supera el punto medio del intervalo: $C^1 = \lceil (29 + 33)/2 \rceil = 31$.

Siguiendo los pasos del algoritmo se obtiene la permutación $\pi_1 = \{1,3,5,6,2,4\}$ para la cual se cumple la desigualdad $C(\pi_1) \le C^1 = 31$. La gráfica de la solución para la solución obtenida π_1 se muestra en la Figura 4.8.

Usando la dicotomía, se busca la mejora de la solución disminuyendo el límite superior del intervalo, al cual actualmente pertenece $C(\pi^*)$.

Segunda iteración del algoritmo: $s = 2$.

Ahora el límite superior del $C(\pi^*)$ es $C^2 = C(\pi_1) = 31$; y la solución óptima se ubica en el intervalo $C(\pi^*) \in (29, 31]$. Al repetir las acciones del algoritmo desde el principio, se encuentra la solución del problema $\pi_2 = \{5, 2, 6, 3, 4, 1\}$ con $C(\pi_2) = 30$.

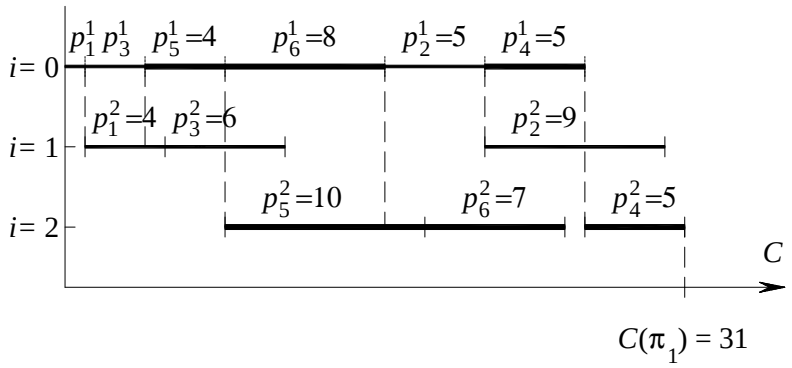


Figura 4.8 Gráfica de Gantt para el procesamiento de los trabajos en la sucesión π_1, obtenida en la segunda iteración del algoritmo.

Cabe mencionar que la solución π_2 es óptima puesto que $C(\pi_2)$ coincide con el límite inferior del $C(\pi^*)$: $\sum_{j=1}^{n} p_j^1 + \min_{1\le j\le n} p_j^2 = 26 + 4 = 30$.

La gráfica de Gantt para la solución obtenida π_2 se muestra en la Figura 4.9.

4.4 Algoritmo heurístico para la resolución del problema de TFH2 con múltiples máquinas en cada etapa

4.4.1 Modelo

Se considera un caso general para el sistema de recursos de dos etapas. Si las máquinas de cada etapa son identicas entre sí, el problema de minimización del tiempo máximo de completar un conjunto de *n* trabajos 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. Tecnológicamente la asignación significa la definición de la ruta tecnológica R_j para cada trabajo *j* en las máquinas de cada etapa.

Es necesario minimizar el tiempo cuando el último trabajo abandona el taller. Las notaciones e implicaciones del modelo se han presentado en el apartado 4.1.

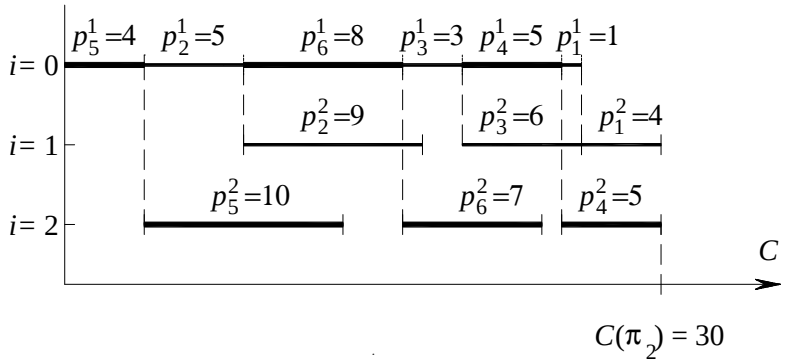


Figura 4.9 Gráfica de Gantt para la solución óptima $\pi_2 = \pi^*$, obtenida en la segunda iteración del algoritmo.

La asignación de los trabajos a las máquinas implica que con respecto a la primera etapa el conjunto de trabajos $N = \{1, 2, ..., n\}$ ahora se representa como una partición del conjunto *N* por *M* disjuntos N_I, tal que $N = \bigcup_{I=1}^{M} N_I$, $N_I \neq \varnothing$, $N_\mu \bigcap N_\nu = \varnothing$, $\mu \neq \nu$. Aquí N_I es un subconjunto de los trabajos, cuya primera operación ejecuta la máquina *I*, $I \in \{1, 2, ..., M\}$.

Analógicamente, con respecto a la segunda etapa, el conjunto *N* se divide por *m* disjuntos N_i, tal que $N = \bigcup_{i=1}^{m} N_i$, $N_i \neq \varnothing$, $N_\mu \bigcap N_\nu = \varnothing$, $\mu \neq \nu$. Aquí N_i es un subconjunto de los trabajos, cuya segunda operación ejecuta la máquina *i*, $i \in \{1, 2, ..., m\}$.

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

Sea G_{Ii} el conjunto de trabajos con la misma ruta tecnológica , $I \in \{1, ..., M\}$, $i \in \{1, ..., m\}$. Se forma el conjunto G_I, $I = \overline{1, M}$, 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, $i = \overline{1, m}$. Sea $B_i = \sum_{j \in Q_i} p_{ij}^2$. Entonces, $G_I = \bigcup_i G_{Ii}$, $G_{Ii} \cap G_{Il} = \varnothing$, $i \neq l$, $i, l \in \{1, 2, ..., m\}$, $I = \overline{1, M}$; $Q_i = \bigcup_I G_{Ii}$, $Q_{Ii} \cap Q_{Li} = \varnothing$, $I \neq L$, $I, L \in \{1, 2, ..., M\}$, $i = \overline{1, m}$. De tal manera, $\bigcup_I Q_i = \bigcup_i G_i$.

4.4.2 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 $C(\Omega)$ en 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 $C(\Omega')$, el valor correspondiente a una solución válida Ω', el cual se determina mediante un algoritmo que representa una modificación del algoritmo de Johnson (Burtseva, et al., 2005).

Los pasos del algoritmo de búsqueda de $C(\Omega')$ son los siguientes:

Paso 1. Sea *G* la lista de los elementos $j = (p_{Ij}^1, p_{ij}^2)$, ordenados en forma decreciente de los valores p_{ij}^2; $A_I = 0$, $I = \overline{1,M}$; $F_i = 0$, $i = \overline{1,m}$; $j = 0$.

Paso 2. $j = j + 1$. Para el elemento $j = (p_{Ij}^1, p_{ij}^2)$ se pone $A_I = A_I + p_{Ij}^1$.

Paso 3. Si $A_I \ge F_i$, entonces $F_i = A_I + q_j^i$, de lo contrario, $F_i = F_i + p_{ij}^2$.

Paso 4. Si $j < n$, entonces ir al Paso 2.

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

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

4.4.3 Algoritmo HA_{M+m}

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

Dado que las máquinas de la primera etapa funcionan de manera independiente una de otra, entonces las primeras operaciones de los trabajos se ejecutan durante el tiempo $A_K = \max(A_I \mid 1 \le I \le M)$. En cualquier solución válida, incluyendo la óptima, el trabajo que 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 la 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 la segunda etapa. En caso contrario, el tiempo de ocupación, no se cambiará.

De nuevo, se determina $\max(A_I \mid 1 \le I \le M)$, después se busca el conjunto G_L, $L = \{1, 2, ..., m\}$ que contiene este trabajo. El procedimiento se repite hasta que los valores de A_I, $I = \overline{1, M}$, sean iguales a cero. La dificultad principal en la búsqueda de la solución óptima $\Omega*$ 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 (Yaurima, et al., 2006; Leyva, et al., 2007). 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 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* 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* $I = \overline{1, M}$. Además, las acciones del algoritmo determinan las *m* 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, $i = \overline{1, m}$.

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

A continuación se presenta un algoritmo eficiente que actúa dentro del esquema del método de dicotomía. En cada iteración *s* el algoritmo forma *M* permutaciones π_L^s, $L = \overline{1, M}$ y *m* permutaciones Ψ_k^s, $k = \overline{1, m}$ de solución del problema $C^s = C(\Omega^s)$, a las cuales unívocamente corresponde la permutación Ω^s (Burtseva & Yaurima, 2004).

Algoritmo HA$_{M+m}$:

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

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

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

$$C^s < A_L^s + p_{ij}^1 + B_i, \; j \in G_{Li}, \tag{4.6}$$

entonces ir al Paso 9.

Paso 7. Entre todos componentes del conjunto G_L^s, que no cumplen la desigualdad (4.6), 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. $s = s + 1$, $C^s = \left\lceil (C_{\min} + C^s)/2 \right\rceil$.

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

$$C^s < B_k^s + p_{Ij}^1 + A_I, \; J_j \in Q_{Ik}, \tag{4.7}$$

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

Paso 13. Entre todos los elementos del conjunto Q_k^s, que no cumplen la desigualdad (4.7), 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 $m_1 = 1$ ó $m_2 = 1$, 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.

Ejemplo 4.5

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 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 la 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 (4.6):

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

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

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

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

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:

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

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

Se coloca el elemento $j_5 = (p_{25}^1, p_{15}^2)$ en la última posición de la permutación π_2^1; $G_2^1 = \{j_6\}$, $A_2^1 = 11-2 = 9$, $B_1 = 1$, $l = 1$, $n_2 = 1$.

Otra vez se busca

$\max(A_1^1, A_2^1) = \max(6,9) = A_2^1 = 9$

y se verifica el cumplimiento de la desigualdad (4.6) para el elemento j_6:

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

De tal manera, el elemento $j_6 = (p_{26}^1, p_{26}^2)$ se ubica en la permutación π_2^1 antes del elemento

$j_5 = (p_{25}^1, p_{15}^2)$; $G_2^1 = \varnothing$, $A_2^1 = 0$.

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

Se repite el Paso 4 del algoritmo:

$\max(A_1^1, A_2^1) = \max(6,0) = A_1^1 = 6$.

Para los tres elementos restantes j_1, j_2, j_4 del conjunto G_1^1 se tienen las siguientes desigualdades:

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

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

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

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

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

se coloca en la tercera posición de la permutación π_1^1.

Repitiendo las operaciones, 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 se repiten 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:

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

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 (4.7) para los elementos del conjunto $Q_1^2 = \{j_1, j_2, j_5\}$:

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

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

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

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:

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

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

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

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

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:

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

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

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 $\Psi_1^2;\ Q_1^2 = \varnothing$, $B_1^2 = 1-1 = 0,\ A_2 = 11,\ l_1 = 3.$

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 4.10 muestra gráficamente el TFH2 construido.

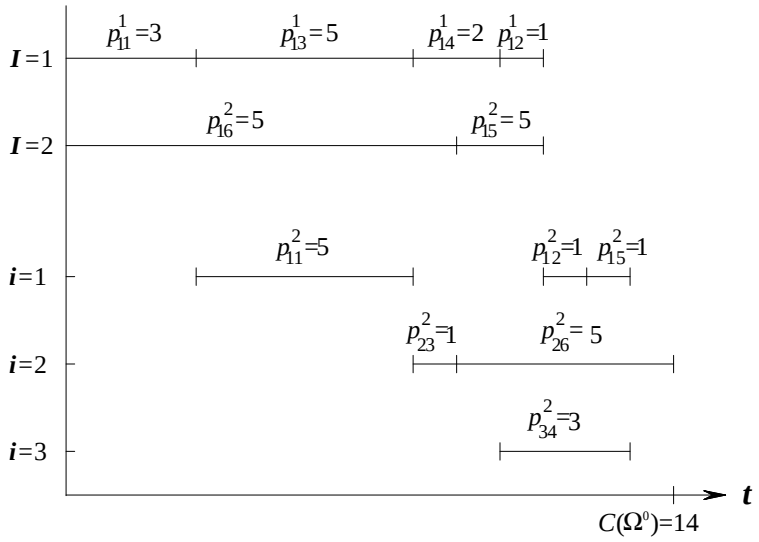


Figura 4.10 TFH2 de longitud óptima

4.5 Conclusiones del capítulo

El problema de TFH2 es **NP**-difícil y por tanto no existe un algoritmo polinomial que obtenga una solución óptima para un caso general. Los algoritmos exactos se desarrollan generalmente sólo para modelos particulares o de tamaño reducido. Existe una variedad de métodos para encontrar una solución exacta o aproximada: métodos de programación matemática (programación dinámica, ramificación y acotación), métodos heurísticos (modificaciones del algoritmo de Johnson, algoritmos voraces) y metaheurísticos (recocido simulado, búsqueda tabú, AGs, 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. Sin embargo, el costo de tiempo de solución es más alto que en el caso de los algoritmos de heurísticas simples.

El algoritmo HA_{1+m} presentado en esta sección se aplica para el TFH2 con una máquina en la primera etapa y el HA_{M+m} para el caso general del TFH2 cuando cada etapa tiene múltiples máquinas. Los algoritmos se aplican en ciertas circunstancias del modelo, para grupos de pedidos con tiempos de ajuste de las máquinas no dependientes de la

secuencia de los trabajos. Estos tiempos de ajuste se incluyen en los tiempos de procesamiento. Los algoritmos requieren una asignación previa de los trabajos a las máquinas. Esta se realiza mediante reglas de prioridad, como: LPT, SPT, algoritmo de Johnson modificado. Se proponen expresiones para determinar el límite inferior en cada caso. Por el límite superior se toma una solución válida del problema. Se proponen algoritmos para calcular una solución válida en cada caso.

Ambos algoritmos se basan en el método de dicotomía, el cual representa un esquema eficiente y económico para algoritmos de optimización combinatoria, en particular, para la programación de la producción. La solución óptima se localiza en un intervalo especificado por el límite superior e inferior, que representa el intervalo inicial para la búsqueda dicotómica. La exactitud de la solución se evalúa numéricamente por el tamaño del intervalo que proporciona la última iteración válida.

La aplicación del método de dicotomía en optimización combinatoria resulta en un procedimiento integral que combina cualidades del método numérico de bisección y el método discreto de búsqueda binaria. Se analizan la convergencia y complejidad del método de dicotomía y se muestra que éste depende del intervalo inicial y de la complejidad del problema de decisión a resolver. A partir del análisis de complejidad del método de dicotomía y los algoritmos HA_{1+m} o HA_{M+m} se concluye que éstos poseen una alta eficiencia. Los algoritmos propuestos son aplicables en un entorno industrial así como en sistemas de tiempo real.

Capítulo 5

Algoritmo genético para un TFH con tiempos de ajuste y restricciones de elegibilidad

5.1 Problema de TFH con tiempos de ajuste y restricciones de elegibilidad

Se considera un modelo de TFH con múltiples etapas, donde existen máquinas no relacionadas, tiempos de ajuste dependientes de la secuencia de trabajos y elegibilidad de máquinas. Algunos aspectos particulares del TFH han recibido mucha atención de numerosos investigadores.

Muchos artículos consideran casos particulares de un TFH con sólo dos o tres etapas. En 1973, uno de los primeros trabajos con *m* etapas fue presentado por Salvador (1973). Se aplica programación dinámica para un taller de flujo con múltiples procesadores sin tiempo de espera.

El número de artículos es escaso si se abordan problemas de este tipo con máquinas no relacionadas. Se conocen seis estudios de modelos con máquinas no relacionadas y múltiples etapas (Adler, 1993; Aghezzaf, 1995; Gourgand, 1999; Allaoui & Artiba, 2004; Ruiz & Maroto, 2006; Zandieh, et al., 2006). Tres de ellos toman en cuenta tiempos de ajuste de las máquinas dependientes de la secuencia (Aghezzaf, 1995; Zandieh, et al., 2006; Ruiz & Maroto, 2006). Solamente en el artículo de Ruiz y Maroto (2006) dentro del mismo problema se consideran las mencionadas restricciones más la de disponibilidad de máquinas. La restricción de disponibilidad de máquinas añade una nueva dimensión a un problema de planificación (Lee, 2004). Shmidt (2000), revisa algunos resultados en este

campo de disponibilidad de máquinas, aunque en ningún caso se han considerado las restricciones mencionadas de manera integral.

Algunas publicaciones consideran entornos de producción de la vida real. En el *Bagpak Production Scheduling System* (BPSS) desarrollado por Adler para la producción de bolsas de papel, se toman en cuenta tiempos de ajuste y máquinas no relacionadas en algunas etapas (Adler, et al., 1993). El BPSS está basado en la aplicación de reglas de prioridad. Aghezzaf propone algunos métodos para resolver un problema de planificación en la manufactura de alfombras, donde se consideran tres etapas y tiempos de ajuste dependientes de la secuencia de los trabajos. La solución se basa en la descomposición del problema, heurísticas y programación entera (Aghezzaf, et al., 1995). Gourgand presenta algunos algoritmos basados en recocido simulado aplicados a un TFH en una industria real (Gourgand, et al., 1999). Jin, et al. (2002), consideran un TFH3 para un caso específico en la producción de tarjetas de circuito impreso. Hay *n* tipos de PCBs a ser calendarizados. Cada tipo debe ser ensamblado en serie de tres etapas con máquinas idénticas en cada etapa. El objetivo es encontrar un calendario que minimice el tiempo de completar todos los trabajos. Allaoui y Artiba manejan un TFH con restricciones de mantenimiento para optimizar algunos objetivos basados en tiempo de flujo (*flow time*) y tiempo de finalización. En este modelo son tomados en cuenta tiempos de ajuste, limpieza y transportación (Allaoui & Artiba, 2004). Un TFH con tiempos de ajuste dependientes de la secuencia es considerado por Tang y Zhang (2005). En este modelo, todos los trabajos pasan por la misma ruta y hay al menos dos máquinas idénticas en cada etapa y al menos una etapa tiene más de dos máquinas. Zandieh, et al. (2006) proponen un algoritmo inmune para este problema pero con máquinas idénticas en cada etapa.

Ruiz y Maroto (2006) proponen un AG para un TFH complejo aplicado a la producción de textiles y losetas. Se toman en cuenta máquinas paralelas no relacionadas, tiempos de ajuste dependientes de la secuencia y restricciones de disponibilidad de máquinas. Se muestra que el algoritmo propuesto es más efectivo entre otros algoritmos relacionados con los cuales se comparó (Nawaz, et al., 1983; Goldberg, 1989; Osman, 1989; Widmer, 1989; Chen, et al., 1995; Murata, et al., 1996; Aldowaisan, 2003; Rajendran & Ziegler, 2004). De acuerdo a la revisión de la literatura, esta última publicación es la

única a la fecha que aborda un TFH con máquinas paralelas no relacionadas, tiempos de ajuste dependientes de la secuencia y restricciones de elegibilidad.

Un AG es una técnica que se usa para encontrar soluciones en problemas de optimización (Holland, 1975). En un AG se genera un conjunto de soluciones candidatas, codificadas como cromosomas (*chromosome*) (llamadas también genomas o individuos), éstos a su vez están compuestos de genes. El conjunto de individuos forman la población inicial (*population*). Se evalúa cada solución o individuo de la población y se le asigna un valor de acuerdo a su aptitud (*fitness*). Los individuos se desarrollan o evolucionan iterativamente generación tras generación a través de operadores genéticos, buscando soluciones óptimas o aproximadas.

Una generación comienza con el mecanismo de selección (*selection*), el cual escoge individuos de la población de manera aleatoria. Los individuos se cruzan para generar nuevos individuos o descendientes (*offspring*) a través de una operación llamada cruzamiento (*crossover*). Así mismo, los individuos se someten a un proceso llamado mutación (*mutation*), en el cual se cambian algunos genes de los cromosomas, de tal manera que una nueva combinación de genes aparezca en la nueva generación. La población evoluciona generación tras generación hasta cumplir un criterio de paro (*stopping*) (Luger, 2002). La eficiencia de un AG depende de la elección de sus operadores y parámetros; entre ellos se tienen: el criterio de paro, método de selección, tamaño de la población, tipo de cruzamiento y tipo de mutación; estos dos últimos con sus respectivas probabilidades (Figura 5.1).

Uno de los primeros trabajos relacionados con el problema del taller de flujo y con AG se debe a Cleveland y Smith (1989). En este trabajo se investiga el problema de secuenciación de sectores (*sector scheduling problem*), el cual es un complejo problema de programación de la producción. Los autores, en una primera aproximación, hacen una simplificación y resuelven un problema muy parecido al taller de flujo mediante un AG, aplicando operadores que se habían utilizado con éxito para el problema del comerciante viajero. En forma general y más explícito se aplica un AG en el artículo de Chen et al. (1995). Su algoritmo incorpora varias modificaciones con respecto a un AG estándar. También de los primeros, Reeves (1995) publicó otro AG para el taller de flujo con características novedosas.

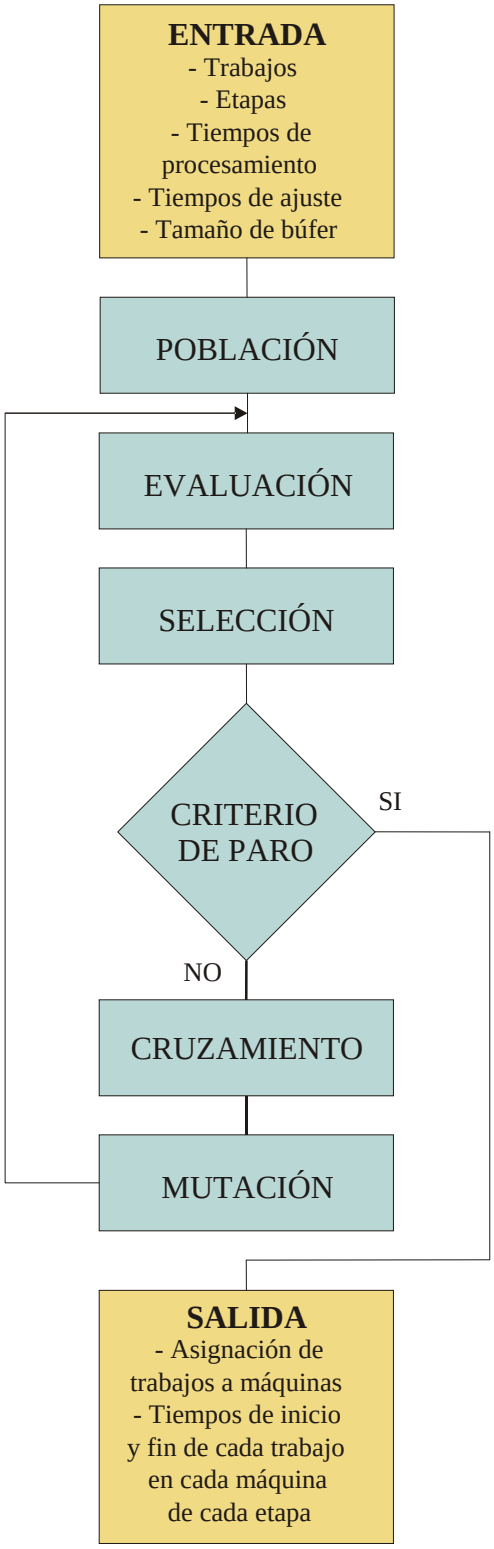


Figura 5.1 Algoritmo Genético

En el presente capítulo se propone un AG, llamado GA_{BC}, para minimizar el tiempo de ejecución de un conjunto de trabajos para un TFH con máquinas no relacionadas, tiempos de ajuste dependientes de la secuencia y disponibilidad / elegibilidad de máquinas. Este algoritmo incluye algunas innovaciones; entre ellas, un nuevo operador de cruzamiento, un especial criterio de paro y condiciones de reinicio; con estas innovaciones se consiguieron mejores resultados.

5.2 Modelo

Se considera el siguiente problema del TFH:

Un conjunto *N* de *n* trabajos, $N = \{1, 2, ..., n\}$ disponibles al tiempo inicial 0 tiene que ser procesado consecutivamente en un conjunto *M* de *m* etapas, $M = \{1, 2, ..., m\}$ con el objetivo de minimizar el tiempo total de procesamiento. En cada etapa $i \in M$, hay un conjunto $M_i = \{1, 2, ..., m_i\}$ de máquinas paralelas no relacionadas, $|M_i| \geq 1$. Cada trabajo debe ser procesado exactamente por una máquina en cada etapa. Sea $p_{i_l, j}$ el tiempo de procesamiento del trabajo $j \in N$, en la máquina $l \in M_i$, dentro de la etapa *i*. Se supone que el tiempo de ajuste de cada máquina depende de la secuencia de los trabajos. Sea $S_{i_l, j, k}$ el tiempo de ajuste en la máquina *l*, en la etapa *i*, para procesar el trabajo $k \in N$, después de haber procesado el trabajo *j*. En la etapa *i*, E_{ij} es un conjunto de máquinas elegibles capaces de procesar el trabajo *j*, $1 \leq E_{ij} \leq m_i$. Esta restricción es conocida como "*machine eligibility constraints*" y se denota por M_j.

Usando la notación de tres campos, el problema considerado es:

$FHm, ((RM^{(i)})_{i=1}^{(m)} \mid S_{sd}, M_j \mid C_{\max}$.

5.3 Codificación

La secuencia de trabajos se representa como un individuo llamado cromosoma. En la codificación se usa una cadena de enteros 1, 2,..., *n*, donde cada entero representa el número de trabajo.

A continuación se presenta el algoritmo GA_{BC}.

Algoritmo 5.1 GA_{BC}:

Entrada: Población de P_{size} individuos.

Salida: Una secuencia de *n* elementos.

```
01. generate_population
02. While stopping_criterion = false do
03.     For i = 0 to P_size
04.         evaluate_objective_function(i)
05.     select_individuals_by_tournament_selecction
06.     keep_the_best_individual_found
07.     If iterations_without_improvement = 25 then
08.         regenerate_population
09.         If regenerate = 10 then
10.             stopping_criterion = true
11.         Else
12.             regenerate = regenerate +1
13.         iterations_without_improvement = 0
14.     Else
15.         If actual_minimum_makespan = previous_minimum_makespan
16.             Increment iterations_without_improvement
17.   crossover_with_probability_P_c
18.   mutation_with_probability_P_m.
```

5.4 Inicialización y evaluación

La población se forma por P_{size} individuos. De acuerdo al algoritmo GA_{BC}, la función *generate_population* genera el conjunto de individuos con sus genes de manera aleatoria y uniforme. La función *evaluate_objective_function* calcula la aptitud (*makespan*) de cada individuo. Muchos autores separan las decisiones de secuenciación y asignación en problemas de TFH (Sherali, et al., 1990; Rajendran & Chaudhuri, 1992). El algoritmo presentado toma un procedimiento propuesto por Ruiz y Maroto (2006), por la siguiente razón: En un TFH sin tiempos de ajuste y restricciones de disponibilidad, la asignación del trabajo a la primera máquina disponible podría resultar en el mejor tiempo de completar el trabajo. En un TFH con máquinas no relacionadas, si la primera máquina disponible es muy lenta para un trabajo dado, asignarlo a esta máquina resulta en un tiempo retrasado, comparado con la asignación a otras máquinas. Ahora, con la consideración de tiempos de ajuste se empeoraría el resultado. Para resolver esta dificultad, el trabajo se asigna a la máquina que lo finaliza en el menor tiempo posible en cada etapa, tomando en consideración las diferentes velocidades, tiempos de ajuste y disponibilidad de máquinas.

El cálculo del tiempo de completar los trabajos es el siguiente:

Sea π una permutación; sea $\pi_{(j)}$ el trabajo en la posición *j* de la permutación, $j \in N$. Cada trabajo se procesa en cada etapa, por lo tanto, se consideran *m* tareas u operaciones por trabajo. Sea L_{i_l} el último trabajo que fue asignado a la máquina *l* dentro de la etapa *i*, $l \in M_i$. $S_{i_l, L_{i_l}, \pi_{(j)}}$ representa el tiempo de ajuste de la máquina *l* en la etapa *i* cuando está por procesar el trabajo $\pi_{(j)}$ después de haber procesado el trabajo previo asignado a la máquina *l* (L_{i_l}).

El tiempo $C_{i,\pi_{(j)}}$ de completar el trabajo $\pi_{(j)}$ en la etapa *i* se calcula de acuerdo a la fórmula:

$$C_{i,\pi_{(j)}} = \min_{1 \le l \le m_j} \{\max\{C_{i,L_{i_l}} + S_{i_l,L_{i_l},\pi_{(j)}}; C_{i-1,\pi_{(j)}}\} + p_{i_l,\pi_{(j)}}\}.$$

Una vez que los trabajos son asignados a las máquinas en las etapas, el tiempo máximo de completar los trabajos se calcula como:

$$C_{\max} = \max_{j=1}^{n}\{C_{m,\pi_{(j)}}\}.$$

5.5 Reinicio y criterio de paro

Se aplica un mecanismo de reinicio basado en el esquema propuesto por Alcaraz, et al. (2003): Después de ordenar por aptitudes en orden ascendente de C_{max}, se mantiene el mejor 20% de individuos, el 50% del resto se reemplaza por mutaciones *Insert* de los primeros 20%. El resto se genera nuevamente de manera aleatoria. Se introduce una modificación para mejorar la calidad de las soluciones, la cual reemplaza el primer 50% del resto por mutaciones *Insert* del mejor individuo. Los detalles de este procedimiento se muestran a continuación.

Algoritmo 5.2 Reinicio:

01. En cada generación *i*, almacenar el mínimo tiempo de completar los trabajos (C_{max}).
02. **Si** $mak_i = mak_i$-1 **entonces**
03. *countmak*=*countmak* +1.
04. **De lo contrario** hacer *countmak* = 0.
05. **Si** *countmak* > *Gr* (número de generaciones sin mejora) **entonces**
06. Ordenar la población en orden ascendente de C_{max}.
07. Dejar el primer 20% de individuos de la lista (éstos son los mejores individuos).
08. Del restante 80% de individuos, 50% de ellos se reemplazan por la mutación SHIFT del mejor individuo y 50% son reemplazados por nuevos individuos generados aleatoriamente.
09. *regeneration* = *regeneration*+1.
10. *Countmak* = 0.
11. **De lo contrario**
12. Continúan las generaciones.

Como se muestra en (Reeves, 1995), un AG estable, donde los descendientes reemplazan los peores individuos de la población, proporciona mejores resultados que AG regulares.

Las evaluaciones paran cuando el mínimo tiempo de completar los trabajos no ha cambiado por 25 veces (Gr = 25) y después que se han ejecutado 10 regeneraciones o reinicios. Finalmente, el algoritmo se ejecuta 2 veces conservando el mejor resultado. Este criterio permite obtener buenas soluciones y tiempos de ejecución razonables.

5.6 Selección, Cruzamiento y Mutación

Para la selección de padres, se considera el torneo binario (*binary tournament*): De la población se toman aleatoriamente dos individuos y se selecciona el primero con mayor aptitud.

Para evitar la convergencia a un óptimo local, se incorpora un operador de mutación, el cual permite introducir material genético perdido y variabilidad en la población. Se consideran tres operadores de mutación (*insert*, *swap* y *switch*), los cuales se usan más frecuentemente en la literatura (Michalewicz, 1996; Gen, 1997; Ruiz & Maroto, 2006):

1. INSERT. Se seleccionan aleatoriamente dos puntos. Si punto1 < punto2, entonces el gen correspondiente a punto1 se coloca en punto2 y todos los genes desde punto1 hasta punto2 se recorren una posición hacia punto1. Si punto1 > punto2, entonces la operación de corrimiento se realiza hacia punto2 (Figura 5.2).

2. SWAP. Se seleccionan dos puntos aleatoriamente y sus genes se intercambian (Figura 5.3).

3. SWITCH. Se realiza un intercambio simple entre dos genes adyacentes. Se selecciona aleatoriamente un punto. El gen correspondiente se intercambia con su sucesor inmediato a la derecha. Si se selecciona el último gen, entonces éste se intercambia con el primero (Figura 5.4).

Un operador de cruzamiento genera nuevas soluciones a partir de dos secuencias. El objetivo es generar soluciones con mejores valores para $C_{\max}$.

Se consideran cinco operadores conocidos; cuatro de ellos han sido aplicados a la programación de la producción y uno (TP) es de uso general:

1. OBX - *Order Based Crossover* (Gen, 1997).

Este operador se basa en una máscara binaria. Los valores de la máscara igual a 1 indican que la correspondiente secuencia de elementos son copiados del padre 1 al hijo. El resto de elementos son copiados del padre 2. Los valores de la máscara son generados de manera aleatoria y uniforme en todas las operaciones de cruzamiento (Figura 5.5).

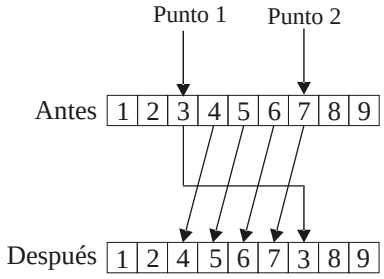


Figura 5.2 Mutación *Insert*

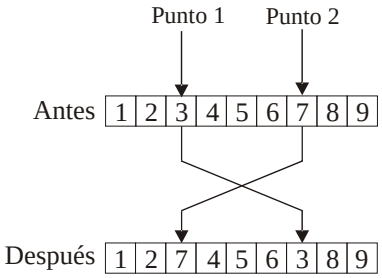


Figura 5.3 Mutación *Swap*

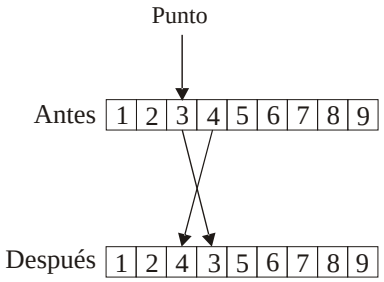


Figura 5.4 Mutación *Switch*

2. PPX - *Precedence Preservative Crossover* (Bierwirth, et al., 1996).

Está basado en una máscara binaria. Los valores de la máscara igual a 1 indican que los elementos correspondientes son copiados del padre 1 al hijo y los valores igual a 0 indican que los elementos son copiados del padre 2, de acuerdo a cada valor de la máscara uno a la vez (de manera alternada) (Figura 5.6).

3. OSX - *One Segment Crossover* (Guinet & Salomon, 1996; Allaoui & Artiba, 2004).

Se escogen dos puntos aleatoriamente. Los elementos se copian del padre 1 desde la posición 1 hasta el primer punto. Se copian los elementos del padre 2 del primer punto al segundo punto. Por último, los elementos se copian del padre 1 a partir del segundo punto a la última posición, considerando solo los elementos no copiados (Figura 5.7).

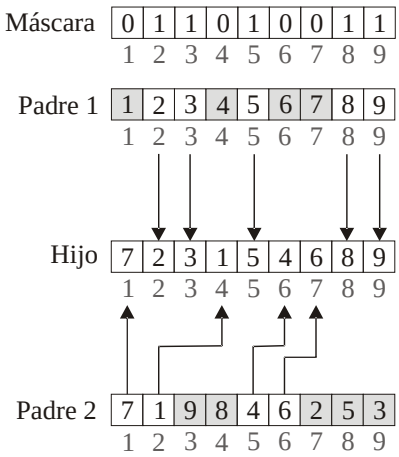


Figura 5.5 Operador de cruzamiento OBX

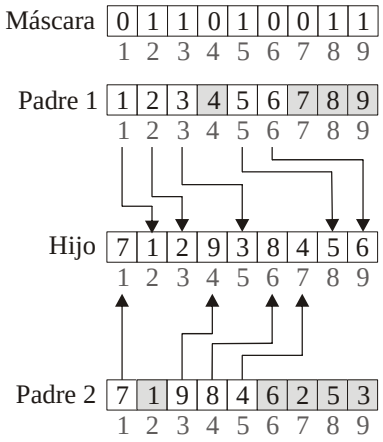


Figura 5.6 Operador de cruzamiento PPX

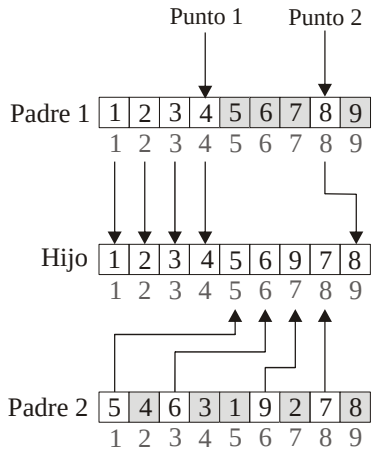


Figura 5.7 Operador de cruzamiento OSX

4. TP - *Two Point* (Michalewicz, 1996).

Se escogen dos puntos aleatoriamente. Los elementos se copian del padre 1 desde la primera posición al primer punto y del segundo punto a la última posición. Los elementos se copian del padre 2 del primer punto al segundo punto (Figura 5.8).

5. SB2OX - Similar Block 2-Point Order Crossover (Ruiz & Maroto, 2006).

Los bloques comunes en ambos padres (al menos dos trabajos idénticos consecutivos) se copian a los hijos; entonces se definen dos puntos de corte aleatorios y la sección entre estos dos puntos se copia directamente a los hijos. Finalmente, los elementos no copiados en los hijos se copian en el orden correspondiente desde los respectivos padres (Figura 5.9).

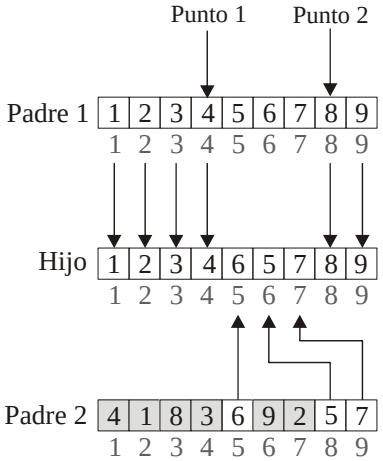


Figura 5.8 Operador de cruzamiento TP

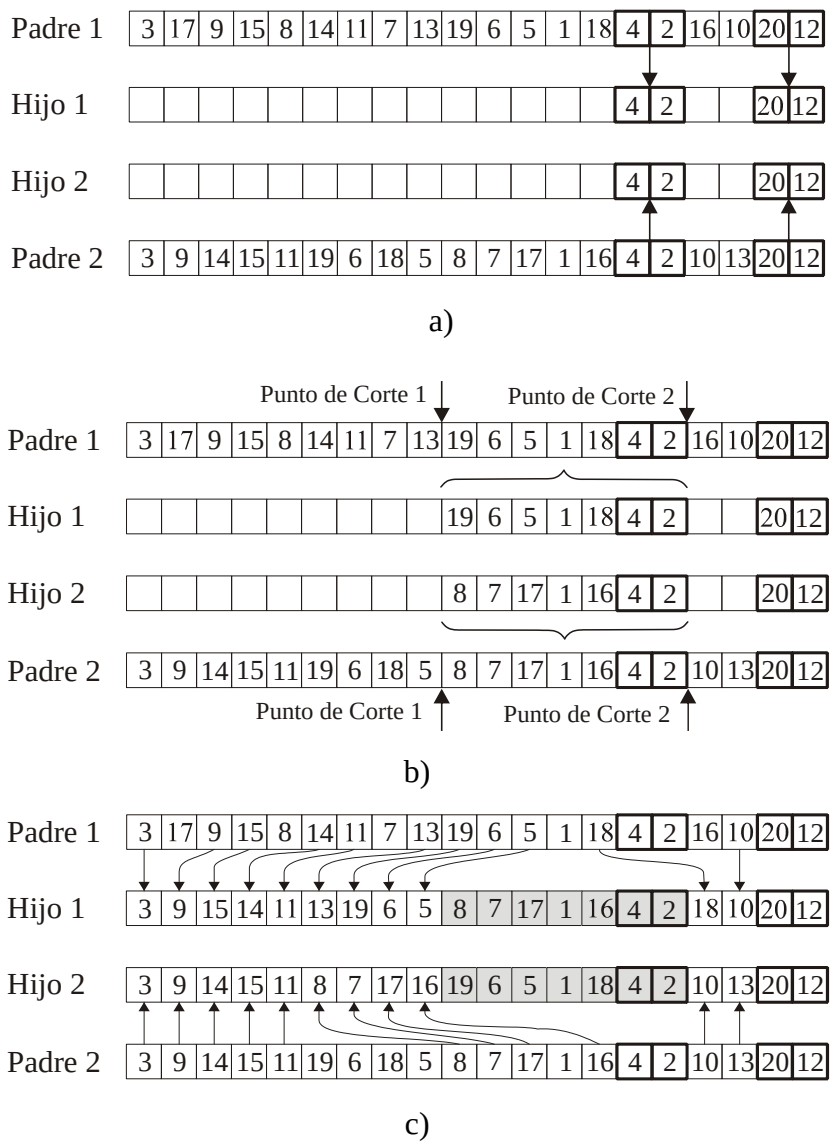


Figura 5.9 Operador de cruzamiento SB2OX (Ruiz, 2003):

a) Los bloques con al menos dos posiciones de valores idénticos en ambos padres, se copian directamente a los hijos;

b) Se copian directamente los elementos entre los dos puntos de corte;

c) Se heredan las posiciones faltantes del padre en su orden relativo

Con el propósito de incrementar la calidad de la solución en este modelo, se desarrollan tres operadores de cruzamiento. El operador TPI (*Two Point Inverse*) es una modificación del operador TP invirtiendo la herencia de los padres. El segundo operador

llamado OBSTX (*Order Based Setup Time Crossover*), es una extensión del operador OBX tomando en cuenta los tiempos de ajuste en la herencia del padre 2. Por último, se desarrolló el operador ST2PX (*Setup Time Two Point Crossover*) que considera el tiempo de ajuste en base a la herencia que realiza el operador TP (Yaurima, et al., 2007).

A continuación se describen los detalles de estos tres nuevos operadores:

1. TPI (*Two Point Inverse*).

Se toman aleatoriamente dos puntos en la secuencia. Los elementos de la posición 1 al primer punto se copian del padre 2. Los elementos del primer punto al segundo punto se copian del padre 1. Los elementos del segundo punto a la última posición se copian del padre 2 (Figura 5.10).

2. OBSTX (*Order based Setup Time Crossover*).

Está basado en una máscara binaria. El valor 1 de la máscara indica que el correspondiente elemento del padre es copiado al hijo. El valor 0 de la máscara indica que el elemento del padre 2 es copiado al hijo de acuerdo al mínimo valor del tiempo de ajuste correspondiente a una máquina elegida aleatoriamente de la primera etapa (Figura 5.11).

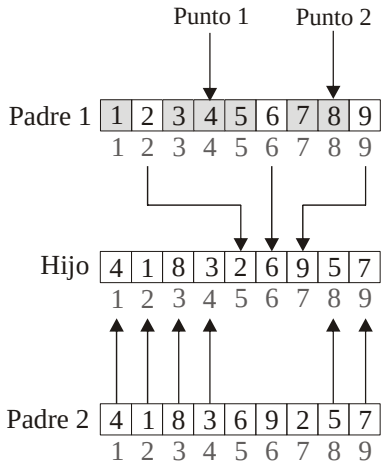
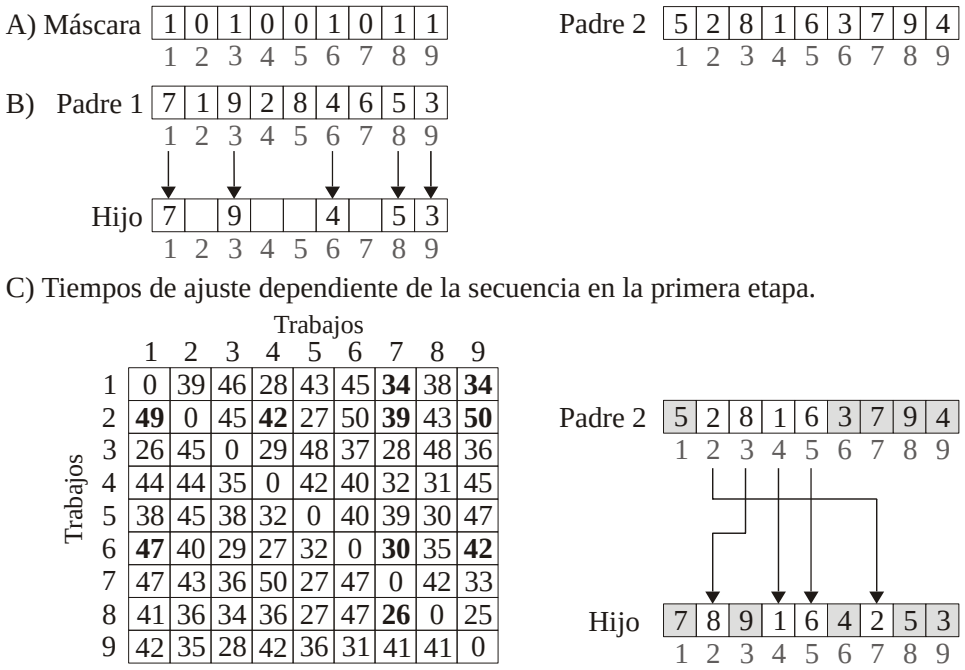


Figura 5.10 Operador de cruzamiento TPI



Después del Trabajo 7: min[39(2),26(8),34(1),30(6)]=26, entonces el Trabajo 8 es elegido.
Después del Trabajo 9: min[50(2),34(1),42(6)]=34, entonces el Trabajo 1 es elegido.
Después del Trabajo 1: min[49(2),47(6)]=47, entonces el Trabajo 6 es elegido.
Después del Trabajo 4: min[42(2)]=42, finalmente el Trabajo 2 es copiado.

Figura 5.11 Operador de cruzamiento OBSTX

3. ST2PX (*Setup Time Two Point Crossover*).

Al igual que el operador anterior, éste toma en consideración el tiempo de ajuste dependiente de la secuencia. Se escogen aleatoriamente dos puntos en la secuencia. Los elementos de la posición 1 al primer punto y del segundo punto a la última posición, se copian del padre 1. Los elementos del primer punto al segundo punto se copian del padre 2 de acuerdo al mínimo tiempo de ajuste dependiente de la secuencia de una máquina de la primera etapa elegida aleatoriamente. Los detalles de este operador se muestran a continuación.

Algoritmo 5.3 Operador de cruzamiento ST2PX:

Entrada: Dos individuos (*padre_1*, *padre_2*)

Salida: Un individuo (*hijo*)

01. Definir el *primer_punto* y *segundo_punto* aleatoria y uniformemente

02. **Desde** *i* = 1 **hasta** *primer_punto* **hacer**

03. Copiar elemento *i* desde *padre_1* a *hijo*

07. **Desde** *j* = *segundo_punto* **hasta** última posición, **hacer**

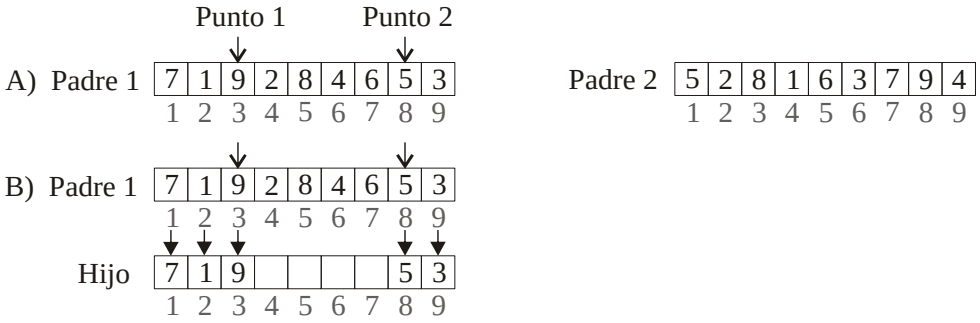
08. Copiar elemento *j* desde *padre_1* a *hijo*

04. **Desde** *k* = *primer_punto* **hasta** s*egundo_punto* **hacer**

05. Escoger aleatoriamente una máquina de la primera etapa

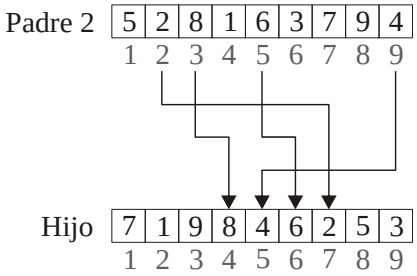
06. Tomar la mejor posición (desde el *primer_punto* hasta el *segundo_punto*) del padre 2 para la posición *k* del hijo, de acuerdo al tiempo de ajuste dependiente de la secuencia de la máquina elegida.

La Figura 5.12 ilustra los pasos principales del operador de cruzamiento ST2PX.



C) Tiempos de ajuste dependiente de la secuencia en la primera etapa.

Trabajos	1	2	3	4	5	6	7	8	9
1	0	39	46	28	43	45	34	38	34
2	49	0	45	**42**	27	**50**	39	**43**	**50**
3	26	45	0	29	48	37	28	48	36
4	44	44	35	0	42	40	32	**31**	**45**
5	38	45	38	32	0	40	39	30	47
6	47	40	29	**27**	32	0	30	**35**	**42**
7	47	43	36	50	27	47	0	42	33
8	41	36	34	36	27	47	26	0	**25**
9	42	35	28	42	36	31	41	41	0



Después del Trabajo 9: min[50(2),25(8),42(6),45(4)]=25, Entonces el Trabajo 8 es elegido.
Después del Trabajo 8: min[43(2),35(6),31(4)]=31, Entonces el Trabajo 4 es elegido.
Después del Trabajo 4: min[42(2),27(6)]=27, Entonces el Trabajo 6 es elegido.
Después del Trabajo 6: min[50(2)]=50, finalmente el Trabajo 2 es copiado.

Figura 5.12 Operador de cruzamiento ST2PX

Se asume que el primer punto es elegido en la posición 3, y el segundo en la posición 8 (Figura 5.12A). Los elementos desde la posición 1 hasta la posición 3 se copian desde el padre 1. Los elementos de la posición 8 a la posición 9 (última posición) son copiados también del padre 1 (Figura 5.12B). El resto de las posiciones del hijo se llenan con los mejores elementos del padre 2, tomando en cuenta los tiempos de ajuste (Figura 5.12C). Cuatro trabajos (2, 8, 4, 6) pretenden ser procesados en la posición 4 del hijo, después de haber procesado el trabajo 9. Así, cuatro tiempos de ajuste (50, 25, 45, 42) son comparados. El trabajo 8 con mínimo tiempo de ajuste (25) se escoge para la posición 4 del hijo.

5.7 Configuración de parámetros de entrada

La configuración de parámetros de entrada y las instancias de referencia fueron tomadas de Ruiz y Maroto (2006) y son las siguientes: El número de etapas es 5, 10, y 20. El número de trabajos es 20. Para cada configuración se tienen 10 problemas diferentes. Los tiempos de procesamiento son uniformemente distribuidos en el rango [1, 99].

Se definen tres principales grupos de instancias de entrada, donde se establece un número de máquinas por etapas. En el primer grupo hay un número distribuido de manera uniforme entre 1 y 3 máquinas por etapas. En el segundo grupo se asigna un número fijo de 2 máquinas por etapa, y en el tercer grupo, 3 máquinas por etapa. Dentro de cada grupo hay 4 diferentes subgrupos de problemas con diferentes configuraciones de tiempos de ajuste dependientes de la secuencia. Los tiempos de ajuste se obtienen de acuerdo a la distribución uniforme en los siguientes rangos U[1, 9], U[1, 49], U[1, 99] y U[1, 124], que corresponden al 10%, 50%, 100% y 125% de los tiempos de procesamiento promedio.

Entonces, se tienen 12 diferentes configuraciones (3 grupos por 4 subgrupos) con 30 problemas cada uno. Respecto a la eligibilidad de máquinas, para todas las configuraciones y para todos los trabajos en todas las instancias se establece un 25% de probabilidad para $p_{i_l,j} = -1$, el cual indica que la máquina l en la etapa i es no elegible para procesar el trabajo j, $\left|E_{ij}\right| \geq 1$. En total se definen 12 diferentes configuraciones o grupos con 110 problemas cada uno. El primer grupo es referido como SSD10_P13, donde se tienen 110 problemas con 1 a 3 máquinas por etapa y donde los tiempos de ajuste son en promedio el

10% de los tiempos de procesamiento. La configuración más grande es la siguiente: 20 trabajos, 20 etapas, con 3 máquinas por etapa. De esta manera, se usa una matriz de tamaño 20×60 para los tiempos de procesamiento y 60 matrices de tamaño 20×20 para los tiempos de ajuste.

5.8 Evaluación experimental

El algoritmo GA_{BC} se compara con el algoritmo de referencia (Ruiz & Maroto, 2006) en el experimento computacional, usando los siguientes parámetros:

Cruzamiento: ST2PX

Mutación: *Swap*

Probabilidad de cruzamiento: 0.8

Probabilidad de mutación: 0.4

Población: 200.

Se utiliza el torneo binario y el criterio de paro descrito en la sección 5.5.

El desempeño del algoritmo se calcula a partir de la siguiente fórmula:

$$IRMS = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$$

donde *IRMS* es el incremento relativo sobre la mejor solución conocida; Heu_{sol} es el valor de la función objetivo obtenido por el algoritmo y $Best_{sol}$ es la mejor solución conocida.

Cada instancia para $Best_{sol}$ es evaluada 500,000 veces con parámetros estándar propuesto en Ruiz y Maroto (2006):

- Selección por *Ranking* (Goldberg, 1989; Michalewicz, 1996),
- Cruzamiento OP (*One point order crossover*) (Michalewicz, 1996),
- Probabilidad de cruzamiento P_c=0.6,
- Probabilidad de mutación P_m=0.01,
- Tamaño de la población P_{size}=50,
- Número de generaciones sin mejora G_r=50.

En (Ruiz & Maroto, 2006) el algoritmo GA_H es comparado con 9 diferentes metaheurísticas adaptadas: *NEH heuristic*, *simulated annealing*, *tabu search*, *ant-base algorithms* (Nawaz et el., 1983; Osman & Potts, 1989; Widmer & Hertz, 1989; Reeves, 1995; Chen, et al., 1995; Murata, et al., 1996; Aldowaisan & Allahvedi, 2003; Rajendran & Ziegler, 2004). Se concluye que el algoritmo GA_H es el mejor para este problema.

Los resultados de la comparación del algoritmo GA_H con el algoritmo GA_{BC} para los 12 grupos de instancias se muestran en la Tabla 5.1. Como se aprecia, después de cierto umbral de complejidad del problema, el algoritmo GA_{BC} muestra resultados satisfactorios con el incremento del tamaño del problema; esto se observa claramente en las Figuras 5.13-5.15.

Las Figuras 5.13 a 5.15 muestran los resultados de las instancias de 1 a 3 máquinas por etapa, dos máquinas por etapa y 3 máquinas por etapa respectivamente. El eje *y* corresponde al IRMS y el eje *x* indica en cada par de números separados por un guión, el número de etapas y el porcentaje del valor del tiempo de ajuste referido en el apartado 5.7.

En el experimento más complejo (SSD125_P3), que se refiere a un 125% de tiempos de ajuste respecto a tiempos de procesamiento y 3 máquinas por etapa, el algoritmo GA_H muestra resultados por debajo de la mejor solución en un promedio de 3.67% (penúltima fila de la Tabla 5.1). Los números negativos en la tabla significan que estos resultados superan a la mejor solución conocida ($Best_{sol}$). El algoritmo GA_{BC} muestra superioridad respecto al IRMS en un rango de 0.16% a 13.35% (Tabla 5.1).

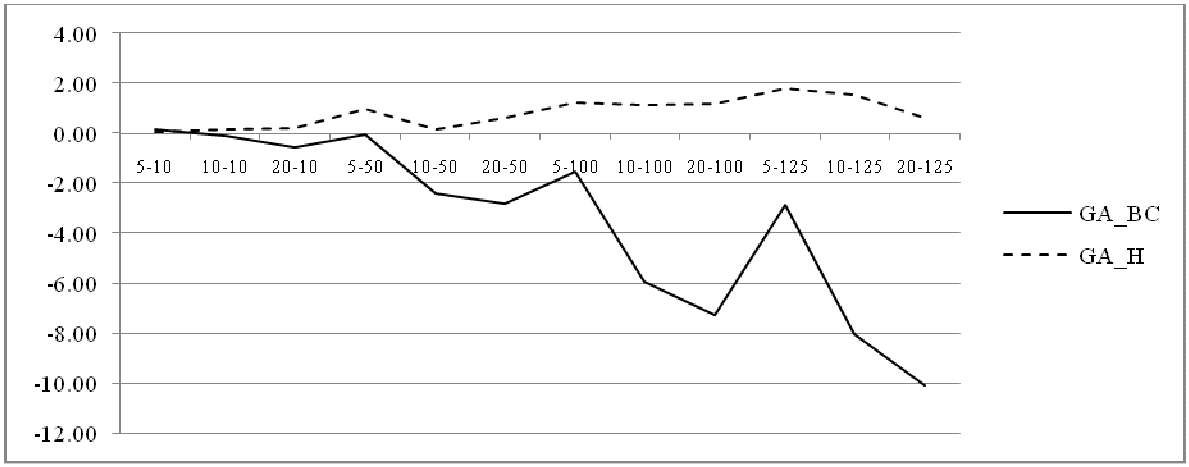


Figura 5.13 Resultados para instancias de 1 a 3 máquinas por etapa. 20 trabajos

Tabla 5.1 Resultados de los algoritmos GA_H y GA_{BC} respecto al IRMS

	SSD10_P13		SSD10_P2		SSD10_P3	
Instancias	GA_{BC}	GA_H	GA_{BC}	GA_H	GA_{BC}	GA_H
20 x 5	0.18	0.06	-0.41	2.64	-1.25	3.06
20 x 10	-0.10	0.16	-0.72	1.76	-0.45	2.34
20 x 20	-0.55	0.22	0.09	1.23	-0.54	1.03
Promedio	-0.16	0.15	-0.35	1.88	-0.74	2.14

	SSD50_P13		SSD50_P2		SSD50_P3	
Instancias	GA_{BC}	GA_H	GA_{BC}	GA_H	GA_{BC}	GA_H
20 x 5	-0.03	0.99	-5.38	3.10	-7.81	4.22
20 x 10	-2.40	0.16	-6.48	1.38	-6.05	2.86
20 x 20	-2.81	0.63	-4.58	1.13	-5.17	1.13
Promedio	-1.75	0.59	-5.48	1.87	-6.34	2.74

	SSD100_P13		SSD100_P2		SSD100_P3	
Instancias	GA_{BC}	GA_H	GA_{BC}	GA_H	GA_{BC}	GA_H
20 x 5	-1.50	1.25	-9.77	4.09	-12.76	5.89
20 x 10	-5.92	1.13	-12.42	2.40	-11.86	2.20
20 x 20	-7.25	1.19	-10.13	1.64	-10.02	1.81
Promedio	-4.89	1.19	-10.77	2.71	-11.55	3.30

	SSD125_P13		SSD125_P2		SSD125_P3	
Instancias	GA_{BC}	GA_H	GA_{BC}	GA_H	GA_{BC}	GA_H
20 x 5	-2.83	1.80	-11.44	3.96	-13.55	5.69
20 x 10	-8.05	1.56	-14.67	2.07	-14.35	3.67
20 x 20	-10.10	0.64	-12.66	1.46	-12.13	1.65
Promedio	-6.99	1.33	-12.92	2.50	-13.35	3.67
Promedio	-3.45	0.82	-7.38	2.24	-7.99	2.96

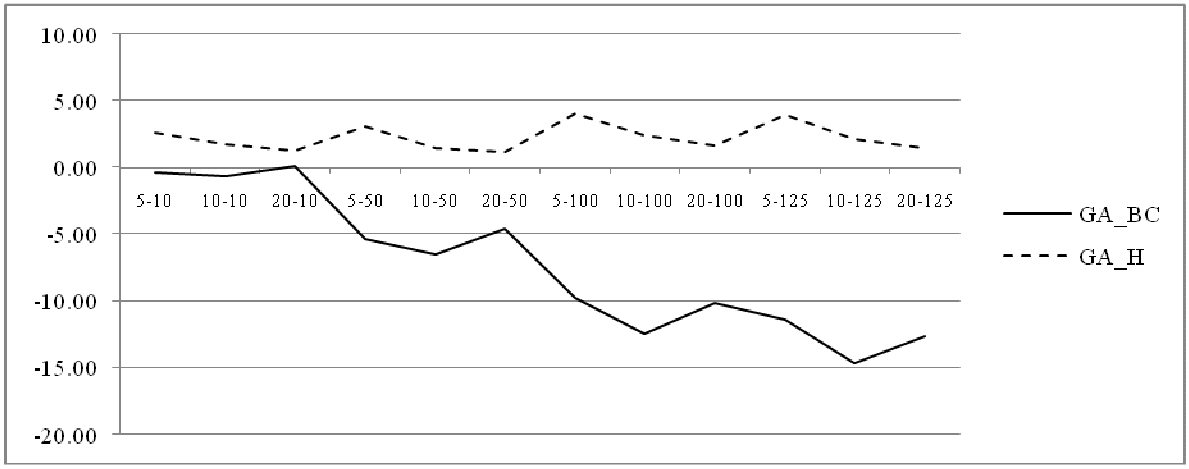


Figura 5.14 Resultados para instancias de 2 máquinas por etapa. 20 trabajos

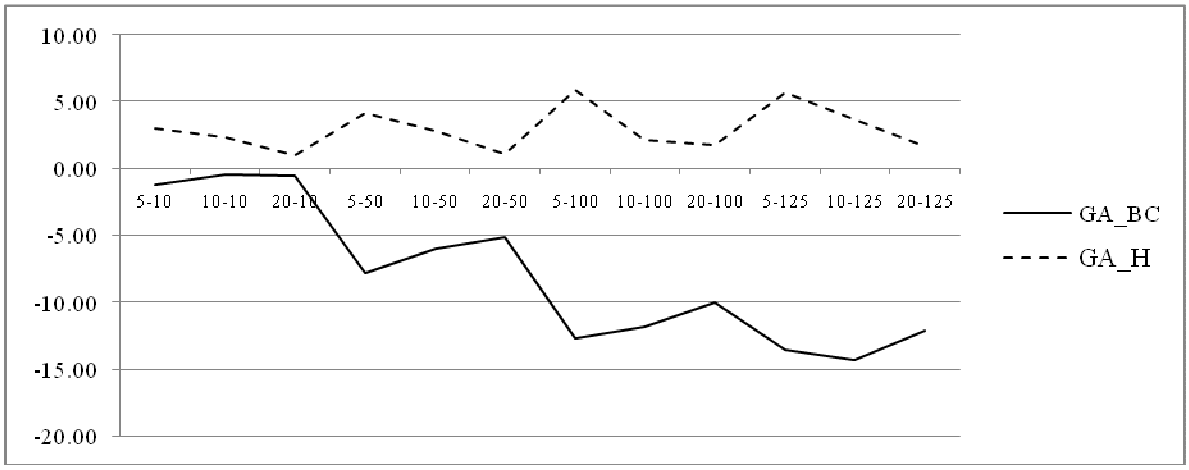


Figura 5.15 Resultados para instancias de 3 máquinas por etapa. 20 trabajos

5.9 Conclusiones del capítulo

Se presenta el algoritmo GA_{BC} para un TFH con múltiples etapas, tiempos de ajuste dependientes de la secuencia, máquinas no relacionadas en cada etapa y elegibilidad de máquinas. Se introducen tres nuevos operadores de cruzamiento para mejorar la calidad de la solución del algoritmo propuesto. El algoritmo es comparado con el mejor AG conocido en la literatura para este problema. Los resultados experimentales obtenidos muestran que en comparación con el conjunto de datos de referencia el algoritmo propuesto tiene buen desempeño en problemas complejos, permitiendo soluciones de alta calidad. Para

problemas de tamaño mediano y grande especialmente, se muestra como los resultados del algoritmo GA_{BC} incrementa la calidad de sus resultados alcanzando un promedio global del 6.27% del IRMS, que indica una superioridad en ese porcentaje respecto a la mejor solución conocida. Se refiere el éxito del algoritmo al criterio de paro y regeneración descritos en el apartado 5.5 que permite salir de óptimos locales; así como al nuevo operador de cruzamiento ST2PX que toma en cuenta una característica importante del modelo como es el tiempo de ajuste dependiente de la secuencia.

Los resultados presentados en este capítulo muestran una visión general de la metodología. En el siguiente capítulo se presenta el desarrollo de este algoritmo con un extensivo análisis estadístico y se evalúa su robustez frente a complejos y grandes problemas considerando el entorno real de una industria electrónica de televisiones.

Capítulo 6

Algoritmo genético para un TFH con tiempos de ajuste, elegibilidad de máquinas y búfer limitado

6.1 Problema del TFH con tiempos de ajuste, elegibilidad de máquinas y búfer limitado

Se aborda un modelo considerando el ambiente de producción de una planta industrial de fabricación de televisiones (Apartado 1.2) que representa un TFH con varias etapas, máquinas no relacionadas (*unrelated machines)*, tiempos de ajuste dependientes de la secuencia de los trabajos (*sequence dependent setup time*), disponibilidad / elegibilidad de máquinas (*machine availability constraints*) y búfer limitado en cada máquina (*limited buffer*).

En el capítulo anterior se hace una reseña amplia de los antecedentes del algoritmo propuesto. En el presente capítulo se agrega la característica de búfer limitado, lo cual aumenta la complejidad del problema.

Una de las primeras publicaciones relacionadas con el taller de flujo y el búfer limitado es el trabajo de Papadimitriou y Kanellakis. Los autores demuestran que el problema del taller de flujo de dos máquinas con búfer limitado es *NP-completo* (Papadimitriou & Kanellakis, 1980). Un análisis detallado de los primeros resultados para un problema con esta restricción es realizado por Weng (2000). Los estudios son dedicados a diferentes aspectos de rebasar el búfer limitado. En Norman (1999), Sawik (2000), Sawik (2002), Wang, et al. (2006), se discuten problemas relacionados con la minimización del

tiempo de completar todos los trabajos. Weng busca la capacidad óptima del búfer para cada máquina; sus resultados muestran que el tamaño del búfer es no más de 4 hasta para 100 trabajos y 20 máquinas (Wang, et al., 2006). El autor enfatiza que el tamaño del búfer es más significativo cuando hay más trabajos. Witt presenta tres heurísticas para controlar el inventario en proceso (Witt, 2007). Son tomados en cuenta diferentes consumos de espacio dentro del almacenamiento limitado. Otros problemas complejos que combinan búfer limitado y tiempos de ajuste dependientes de la secuencia son estudiados por Norman (1999). Un procedimiento de búsqueda tabú es usado para encontrar la solución.

Cómo se observa, las características consideradas: máquinas paralelas no relacionadas, tiempos de ajuste dependientes de la secuencia, elegibilidad de máquinas y búfer limitado, son bastante conocidas; sin embargo no han sido consideradas todas ellas juntas en el mismo problema.

6.2 Modelo

Se considera el siguiente problema del TFH: Un conjunto *N* de *n* trabajos, $N = \{1, 2, ..., n\}$ disponibles al tiempo inicial 0 tiene que ser procesado en un conjunto *M* de *m* etapas de producción consecutivas, $M = \{1, 2, ..., m\}$ con el objetivo de minimizar el tiempo total de procesamiento. En cada etapa $i \in M$, hay un conjunto $M_i = \{1, 2, ..., m_i\}$ de máquinas paralelas no relacionadas, donde $|M_i| \geq 1$. Cada trabajo debe ser procesado por exactamente una máquina en cada etapa. Sea $p_{i_l, j}$ el tiempo de procesamiento del trabajo $j \in N$, en la máquina $l \in M_i$, dentro de la etapa *i*. Se considera que el tiempo de ajuste de cada máquina depende de la secuencia de los trabajos. Sea $S_{i_l, j, k}$ el tiempo de ajuste en la máquina *l* de la etapa *i* para procesar el trabajo *k*, $k \in N$, después de haber procesado el trabajo *j*. Para la etapa *i*, E_{ij} es un conjunto de máquinas elegibles capaces de procesar el trabajo *j*, $1 \leq E_{ij} \leq m_i$. Esta restricción es denotada por M_j conocida también como "*machine eligibility constraints*". Para cada máquina $l \in M_i$ se tiene un búfer limitado para almacenar temporalmente los trabajos, el cual es llamado "inventario en proceso" (*work in*

process). Una máxima capacidad frente a la máquina l se define como $b_{i,l}$, donde $1 \le |b_{i,l}| \le n$.

Gourgand et al. (1999) muestra que para el problema dado, el número posible de soluciones es $n!\left(\prod_{i=1}^{m} m_i\right)^n$. Además, Gupta y Tunc (1998) comprueban que un TFF con solo dos etapas *(m = 2)* es *NP-Difícil* aún si una de las dos etapas contiene una sola máquina. Siendo el TFH una generalización de un TFF, se tiene que un TFH es también *NP- Difícil*.

El problema considerado se denota de la siguiente manera:

$$FHm,((RM^{(i)})_{i=1}^{(m)} \mid S_{sd}, M_j, Block \mid C_{\max}.$$

6.3 Codificación

Para una secuencia de trabajos se usa una cadena de enteros, la cual es una permutación de números 1,2,...,*n,* y cada entero representa un trabajo.

A continuación se describe el algoritmo GA_{SBC}.

Algoritmo 6.1 GA_{SBC}:

Entrada: Población de P_{size} individuos.

Salida: Una secuencia de *n* elementos.

```
01. generate_population
01. regeneration = 1
02. Mientras stopping_criterion = falso hacer
03.     Desde i=0 hasta P_size
04.         evaluate_objective_function_with_buffer(i)
06.     keep_the_best_individual_found()
07.     Si actual_best_makespan >= previous_best_makespan entonces
08.         incrementar iterations_without_improvement
09.         Si iterations_without_improvement = 25 entonces
09.             Si regeneration = 10 entonces
10.                 stopping_criterion = verdadero
```

```
11.                 De lo contrario
12.                     sort_the_population_in_ascending_order_of_Cmax()
14.                     regenerate_population()

12.                     regeneration = regeneration +1
13.                     iterations_without_improvement = 0
05.     select_individuals_by_binary_tournament_selecction
17.     crossover_ST2PX_with_probability 0.8
18.     mutation_SWAP_with_probability 0.1
```

6.4 Inicialización y evaluación

Al igual que en el capítulo anterior, la población es formada por P_{size} individuos. De acuerdo al algoritmo 6.1, la función *generate_population* genera el conjunto de individuos con sus genes de manera aleatoria y uniforme. La función *evaluate_objective_function_with_buffer* calcula la aptitud (*makespan*) de cada individuo tomando en cuenta el tamaño del búfer en cada máquina. Si el búfer de una máquina se satura, el flujo continúa por otra máquina de la misma etapa, considerando el mismo criterio de elección: Cada trabajo es asignado a la máquina que pueda finalizar el trabajo en el menor tiempo posible en cada etapa, tomando en consideración las diferentes velocidades, tiempos de ajuste, disponibilidad de máquinas y el tamaño del búfer. El cálculo del tiempo de completar los trabajos es descrito en el apartado 5.4.

La función *keep_the_best_individual_found* guarda el mejor individuo encontrado en la anterior evaluación de la población en cada iteración. El resultado final será el último mejor individuo encontrado para la función objetivo.

6.5 Reinicio y criterio de paro

Se aplica un mecanismo de reinicio donde después de ordenar por aptitudes, se mantiene el mejor 20% de individuos, el 50% del resto (80%) es reemplazado por mutaciones SHIFT del mejor individuo. El resto es generado nuevamente de manera aleatoria. Los detalles de

este procedimiento se muestran en el Capítulo 5. Las evaluaciones se detienen cuando el mínimo tiempo de completar los trabajos no ha cambiado por 25 veces ($Gr = 25$) y después que se han ejecutado 10 regeneraciones o reinicios. Este criterio permite buenas soluciones y razonables tiempos de ejecución.

6.6 Selección, Cruzamiento y Mutación

Se considera la selección por torneo binario y los tipos de cruzamientos descritos en el apartado 5.6, cinco de ellos tomados de la literatura:

OBX - *Order Based Crossover* (Gen, 1997),

PPX - *Precedence Preservative Crossover* (Bierwirth, et al., 1996),

OSX - *One Segment Crossover* (Guinet & Salomon, 1996; Allaoui & Artiba, 2004),

TP - *Two Point* (Michalewicz, 1996),

SB2OX - Similar Block 2-Point Order Crossover (Ruiz & Maroto, 2006),

y tres propuestos:

TPI (*Two Point Inverse*),

OBSTX (*Order based Setup Time Crossover*),

ST2PX (*Setup Time Two Point Crossover*).

Se consideran también tres conocidos operadores de mutación descritos en el apartado 5.6 (*insert*, *swap* y *switch*) usados con frecuencia en la literatura (Michalewicz, 1996; Gen, 1997; Ruiz & Maroto, 2006).

6.7 Calibración del algoritmo

La calibración es un procedimiento para elegir los parámetros del algoritmo que proporcionan el mejor resultado en la variable respuesta. Se realiza a través de un experimento computacional. Incluye los siguientes pasos (Montgomery, 1996):

- Elección de los factores que tienen el mayor efecto a la variable respuesta, determinación de los valores de los factores elegidos (niveles de los factores), definición de la hipótesis nula acerca de la influencia de los factores y sus combinaciones, e intervalos de confianza para la variable dependiente.

- Configuración de parámetros de entrada, como número de trabajos, etapas, máquinas por etapa, etc.
- Realización del experimento.
- Análisis residual para la comprobación de idoneidad del modelo.
- Análisis de varianza (ANOVA multifactorial) para aceptar o rechazar la hipótesis nula.
- Elección de la combinación de los niveles de los factores elegidos, la cual proporciona el mejor resultado en la variable respuesta; y en base a esto se obtiene el algoritmo calibrado.

6.7.1 Diseño del experimento

En el experimento se analiza el efecto a la función objetivo (variable respuesta) de los factores: tamaño de la población, tipo de cruzamiento y mutación con sus probabilidades.

Para calibrar el algoritmo se analizan los siguientes niveles de los factores:

Tamaño de la población (P_{size}): 100, 150 y 200;

Operadores de cruzamiento: OBX, PPX, OSX, TP, ST2PX, TPI, OBSTX, SB2OX;

Probabilidad de cruzamiento (P_c): 0.5, 0.6, 0.7, 0.8, 0.9;

Operadores de mutación: Insert, Swap, Switch;

Probabilidad de mutación (P_m): 0.03, 0.05, 0.1, 0.2, 0.4.

En el experimento de calibración del algoritmo se evalúan todas las combinaciones posibles de los niveles de los factores para cada instancia de entrada. Por consiguiente, se tienen $3 \times 8 \times 5 \times 3 \times 5 = 1800$ diferentes combinaciones o algoritmos para cada uno de los 60 problemas que se describen en el apartado 6.7.2.

Como variable respuesta se utiliza el valor de C_{max} en su forma normalizada IRMS (Incremento relativo sobre la mejor solución):

$$IRMS = \frac{Heu_{Sol} - Best_{Sol}}{Best_{Sol}} \cdot 100 ,$$

donde Heu_{Sol} es el valor de la función objetivo del algoritmo y $Best_{Sol}$ es el mejor valor obtenido de las 1800 combinaciones posibles de los factores para el problema.

La hipótesis nula del experimento es: no existe un efecto significativo de los factores y sus interacciones a la variable respuesta. Para el intervalo de confianza se toma el valor de 95% (Ruiz & Maroto, 2006).

6.7.2 Configuración de parámetros de entrada

Los parámetros de entrada corresponden a un entorno de producción real. Se considera el modelo de recursos de 2, 3 y 6 etapas, con 10 problemas para cada valor. El número de máquinas por etapa es aleatorio y se obtiene según distribución uniforme en el intervalo [1, 10]. Para cada configuración se consideran 50 y 100 trabajos con tiempos de procesamiento distribuidos uniformemente en el rango [50, 99]. Por lo tanto, se tienen $3 \times 10 \times 2 = 60$ diferentes instancias o problemas, los cuales se dividen en dos grupos de acuerdo al número de trabajos. El primer grupo está compuesto por 30 instancias de 50 trabajos y el segundo grupo por otras 30 instancias de 100 trabajos (Ver Tabla 6.1). La variable respuesta en el experimento es el porcentaje de incremento relativo sobre la mejor solución conocida (IRMS) para cada uno de los 60 problemas.

Las instancias de entrada se basan en el generador lineal congruente de números pseudoaleatorios enteros propuesto por Taillard (1993) para talleres de flujo simple; son complementadas por Ruiz y Maroto (2006) para TFH y adaptadas para el modelo que se aborda con valores de parámetros de acuerdo al caso real en estudio.

A continuación se presenta la implementación del generador lineal congruente de Taillard (1993), el cual utiliza aritmética modular a través de la fórmula recursiva:

$$X_{i+1} := (16807 X_i) \, \mathbf{mod} (2^{31} - 1).$$

Esta implementación usa enteros de 32 bits y provee una secuencia distribuida uniforme de números en el rango (0,1):

1. Semilla inicial y constantes:

$$X_0 (0 < X_0 < 2^{31} - 1)$$

$$a = 16807,\ b = 127773,\ c = 2836,\ m = 2^{31} - 1$$

Tabla 6.1 Instancias de entrada

Primer Grupo			Segundo grupo		
Instancias	Trabajos	Etapas	Instancias	Trabajos	Etapas
1	50	2	31	100	2
2	50	2	32	100	2
3	50	2	33	100	2
4	50	2	34	100	2
5	50	2	35	100	2
6	50	2	36	100	2
7	50	2	37	100	2
8	50	2	38	100	2
9	50	2	39	100	2
10	50	2	40	100	2
11	50	3	41	100	3
12	50	3	42	100	3
13	50	3	43	100	3
14	50	3	44	100	3
15	50	3	45	100	3
16	50	3	46	100	3
17	50	3	47	100	3
18	50	3	48	100	3
19	50	3	49	100	3
20	50	3	50	100	3
21	50	6	51	100	6
22	50	6	52	100	6
23	50	6	53	100	6
24	50	6	54	100	6
25	50	6	55	100	6
26	50	6	56	100	6
27	50	6	57	100	6
28	50	6	58	100	6
29	50	6	59	100	6
30	50	6	60	100	6

2. Modificación de la semilla $k := \lfloor X_i / b \rfloor$

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

$$\text{If } X_{i+1} < 0 \text{ then } X_{i+1} := X_{i+1} + m$$

3. Nuevo valor de la semilla X_{i+1}

4. Valor actual del generador. X_{i+1} / m

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

El número aleatorio que el generador produce es denotado por $U(0,1)$, donde $0 < U(0,1) < 1$ para cada número generado. 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:

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

Para cada número entero aleatorio generado, se tiene un $a \leq U[a,b] \leq b$ y cada entero entre *a* y *b* tiene la misma probabilidad de ser escogido. El Anexo C incluye un ejemplo de un archivo de instancia.

La Figura 6.1 muestra la primera parte de una instancia donde se observan tres números en la primera línea:

50 es el número total de trabajos,

19 es el número total de máquinas,

3 es el número total de etapas.

En la segunda línea se indica el número de máquinas por etapa. En este caso, como son tres etapas, se observa que la primera etapa tiene 5 máquinas, la segunda etapa 6 máquinas y la tercera 8 máquinas. Las columnas se organizan en pares: el primer elemento del par indica el número de máquina y el segundo elemento, el tiempo que tardaría esa máquina en procesar el trabajo. La primera fila corresponde al trabajo 0, la segunda fila corresponde al trabajo 1, la tercera fila, al trabajo 2, y así sucesivamente. Algunos tiempos de procesamiento tienen valor -1, esto indica que la máquina correspondiente no está disponible para procesar el trabajo indicado por el número de fila.

La Figura 6.2 muestra la sección de tiempos de ajuste. La primera línea contiene el identificador SSD, el cual define el inicio de las matrices de tiempos de ajuste con una matriz por cada máquina. Cada posición en las columnas y filas indica un número de trabajo. Cada dato representa el tiempo de ajuste que requiere la máquina antes de pasar el trabajo que indica el número de columna y después del trabajo que indica el número de fila.

En el ejemplo de la Figura 6.2 se muestra que la máquina 0 requiere 28 unidades de tiempo de ajuste para procesar el trabajo 17 después haber pasado por esa máquina el trabajo número 10.

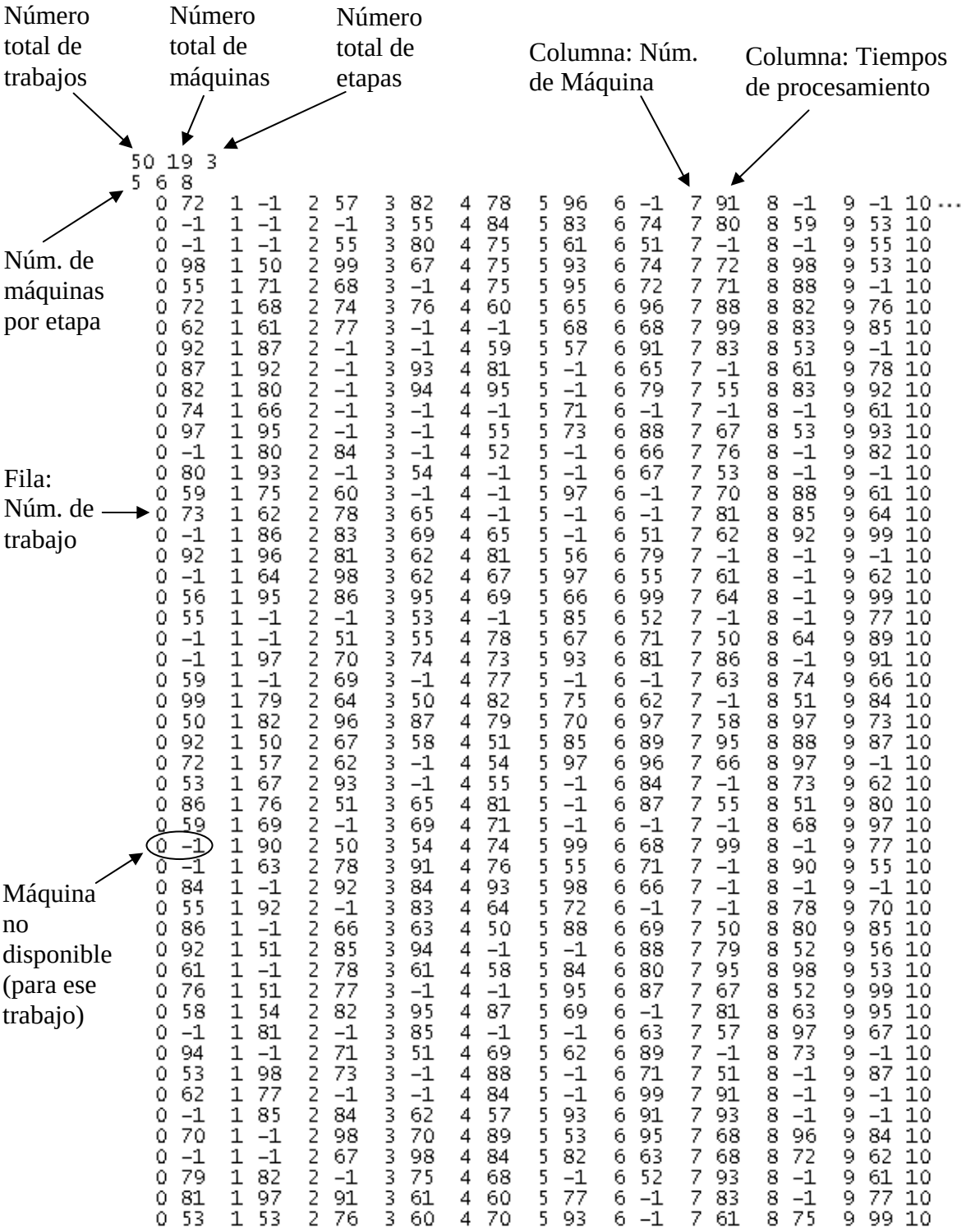


Figura 6.1 Archivo de Instancia. Cabecera y tiempos de procesamiento.

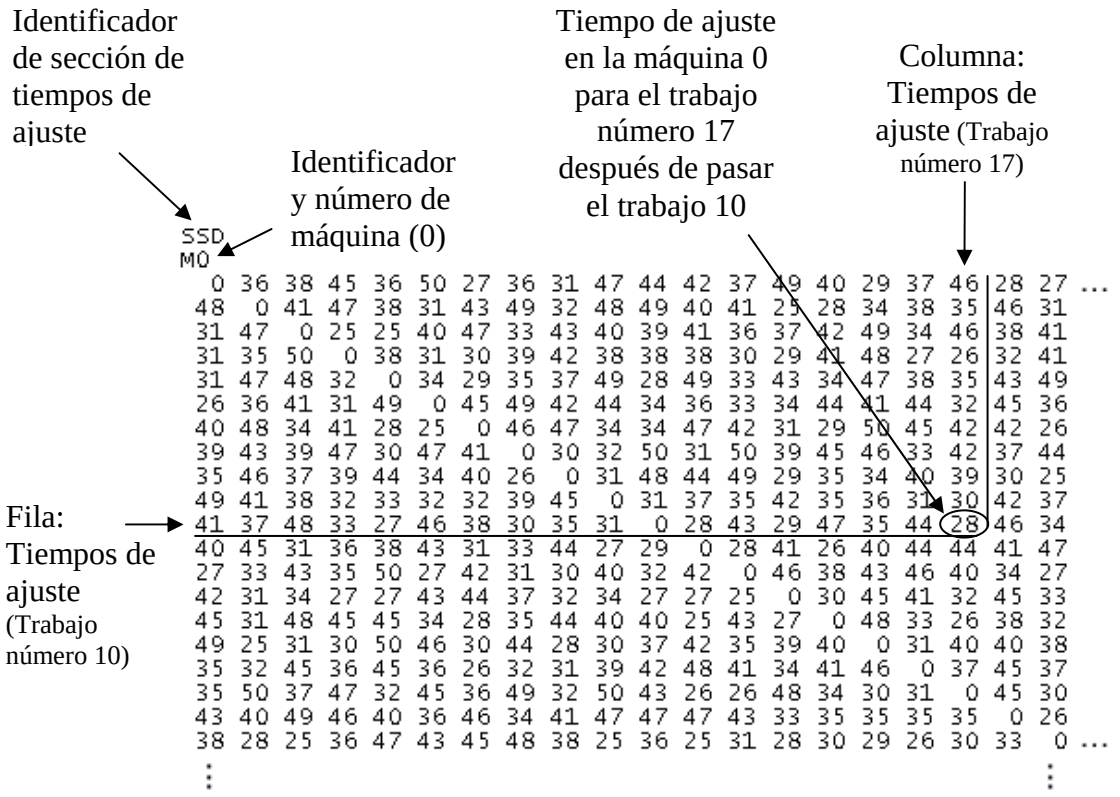


Figura 6.2 Archivo de Instancia. Tiempos de ajuste.

La Figura 6.3 muestra un vector que describe el búfer limitado para una máquina. Los datos se organizan en pares, el primer elemento es el número de máquina y el segundo es el número de trabajos concluidos que permite almacenar temporalmente el espacio físico de esa máquina.

Los tiempos de ajuste están uniformemente distribuidos en el intervalo [25, 50], lo que corresponde al rango [25%, 50%] de los tiempos de procesamiento promedio. Así mismo, hay un 25% de probabilidad de maquinas no disponibles para procesar algún tipo

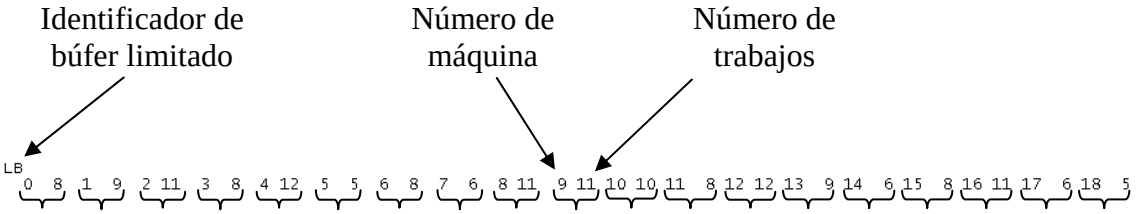


Figura 6.3 Archivo de Instancia. Búfer limitado.

de trabajo, siendo denotado por $p_{i_l,j} = -1$, el cual indica que la máquina *l* en la etapa *i* no es elegible o no está disponible para procesar el trabajo *j*; $|E_{ij}| \geq 1$. El búfer para cada máquina toma valor entre el 25% y 50% del número de trabajos.

La configuración más grande es la siguiente: 100 trabajos, 6 etapas, 10 máquinas por etapa. Se tiene entonces una matriz de 100 × 60 para los tiempos de procesamiento, 60 matrices de 100 × 100 para los tiempos de ajuste y un vector de búferes para 60 máquinas.

El criterio de paro es el siguiente: Después de 25 iteraciones o generaciones sin mejora, se procede a una regeneración de la población hasta un máximo de 10 veces.

6.7.3 Realización del experimento computacional

Los experimentos se han realizado en una PC con procesador Pentium 4, 2.8 GHz y 512 MB de memoria principal y sistema operativo Windows XP. La Tabla 6.2 muestra los resultados del experimento "ta_100_3_1".

La primera columna indica el número de corrida (combinación de factores). Las columnas de la 2 a la 6 contienen los índices de los factores (Pb: Población, Cr: Operador de cruzamiento, Mt: Operador de mutación, Pc: Probabilidad de cruzamiento, Pm: Probabilidad de mutación). Las siguientes columnas muestran los resultados del experimento.

Posteriormente los resultados se han sometido a un Análisis de Varianza Multifactorial (ANOVA) con el 95% de nivel de confiabilidad para verificar la hipótesis nula y determinar los parámetros que tienen mayor influencia.

6.7.4 Análisis residual

Los residuos se calculan de acuerdo a la siguiente fórmula (Montgomery & Runger, 1994):

$$e_i = y_i - \bar{y}_i,\ i = 1, 2, 3, ..., n,$$

donde y_i es el IRMS para la corrida *i* y $\bar{y}_i$ es la media del experimento. El conjunto de resultados e_i se utiliza para verificar la suposición de idoneidad del modelo, la cual se comprueba a través de la normalidad, homocedasticidad e independencia de residuos. El análisis residual se ha realizado con el software STATGRAPHICS (versión 15.1.02).

Tabla 6.2 Resultados del experimento "ta_100_3_1".

Num. de Corrida	Factores					Cmax	Mejor solución	*IRMS*	Residuo
	Pb	Cr	Mt	Pc	Pm				
1	2	3	4	5	6	7	8	9	10
1	100	1	1	1	1	1868	1801	1.03720	-0.00411
2	100	1	1	1	2	1881	1801	1.04442	0.00310
3	100	1	1	1	3	1848	1801	1.02610	-0.01522
4	100	1	1	1	4	1888	1801	1.04831	0.00699
5	100	1	1	1	5	1896	1801	1.05275	0.01143
6	100	1	1	2	1	1893	1801	1.05108	0.00977
7	100	1	1	2	2	1858	1801	1.03165	-0.00967
8	100	1	1	2	3	1873	1801	1.03998	-0.00134
9	100	1	1	2	4	1916	1801	1.06385	0.02254
10	100	1	1	2	5	1890	1801	1.04942	0.00810
11	100	1	1	3	1	1866	1801	1.03609	-0.00523
12	100	1	1	3	2	1855	1801	1.02998	-0.01133
13	100	1	1	3	3	1913	1801	1.06219	0.02087
14	100	1	1	3	4	1893	1801	1.05108	0.00977
15	100	1	1	3	5	1911	1801	1.06108	0.01976
16	100	1	1	4	1	1846	1801	1.02499	-0.01633
17	100	1	1	4	2	1908	1801	1.05941	0.01810
18	100	1	1	4	3	1900	1801	1.05497	0.01365
19	100	1	1	4	4	1887	1801	1.04775	0.00644
20	100	1	1	4	5	1914	1801	1.06274	0.02143
21	100	1	1	5	1	1915	1801	1.06330	0.02198
22	100	1	1	5	2	1903	1801	1.05664	0.01532
23	100	1	1	5	3	1873	1801	1.03998	-0.00134
24	100	1	1	5	4	1907	1801	1.05886	0.01754
25	100	1	1	5	5	1908	1801	1.05941	0.01810
26	100	1	2	1	1	1868	1801	1.03720	-0.00411
27	100	1	2	1	2	1894	1801	1.05164	0.01032
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1783	200	8	3	2	3	1898	1801	1.05386	0.01254
1784	200	8	3	2	4	1861	1801	1.03331	-0.00800
1785	200	8	3	2	5	1842	1801	1.02277	-0.01855
1786	200	8	3	3	1	1894	1801	1.05164	0.01032
1787	200	8	3	3	2	1865	1801	1.03554	-0.00578
1788	200	8	3	3	3	1886	1801	1.04720	0.00588
1789	200	8	3	3	4	1835	1801	1.01888	-0.02244
1790	200	8	3	3	5	1854	1801	1.02943	-0.01189
1791	200	8	3	4	1	1877	1801	1.04220	0.00088
1792	200	8	3	4	2	1864	1801	1.03498	-0.00634
1793	200	8	3	4	3	1856	1801	1.03054	-0.01078
1794	200	8	3	4	4	1832	1801	1.01721	-0.02410
1795	200	8	3	4	5	1852	1801	1.02832	-0.01300
1796	200	8	3	5	1	1859	1801	1.03220	-0.00911
1797	200	8	3	5	2	1820	1801	1.01055	-0.03077
1798	200	8	3	5	3	1836	1801	1.01943	-0.02188
1799	200	8	3	5	4	1837	1801	1.01999	-0.02133
1800	200	8	3	5	5	1843	1801	1.02332	-0.01800

La Figura 6.4 muestra la gráfica de probabilidad normal de residuos para el experimento “ta_100_3_1” (100 trabajos, 3 etapas). Como se aprecia, la gráfica cumple con la suposición de normalidad, pues los residuos se aproximan a una línea recta con una ligera desviación al principio y al final.

La suposición de la homocedasticidad de residuos se verifica para cada factor graficando los residuos frente a sus niveles (Figuras 6.5 – 6.9).

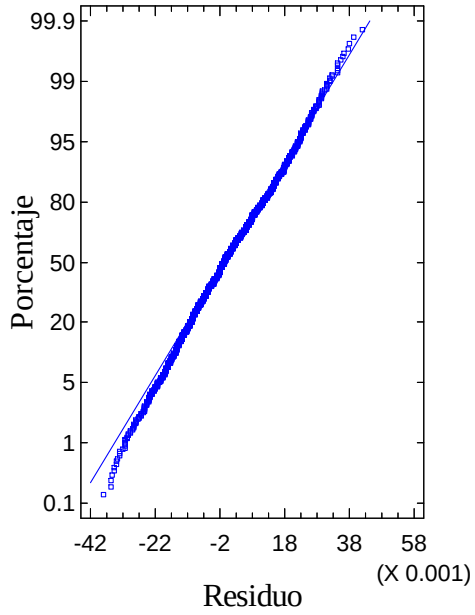


Figura 6.4 Gráfica de probabilidad normal de residuos para el experimento “ta_100_3_1”

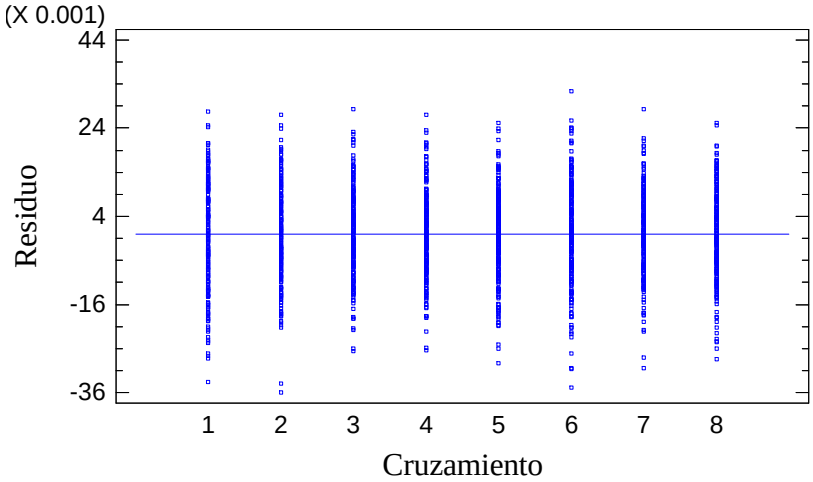


Figura 6.5 Gráfica de residuos contra tipo de cruzamiento en el experimento “ta_100_3_1”.

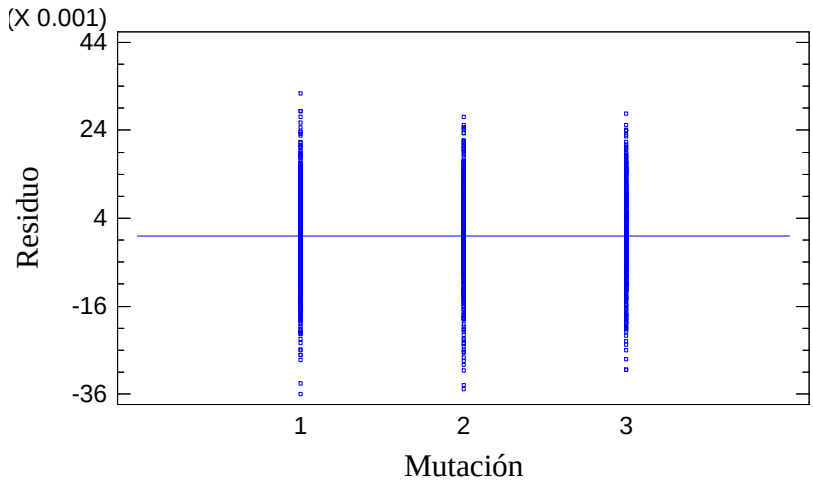


Figura 6.6 Gráfica de residuos contra tipo de mutación en el experimento “ta_100_3_1”.

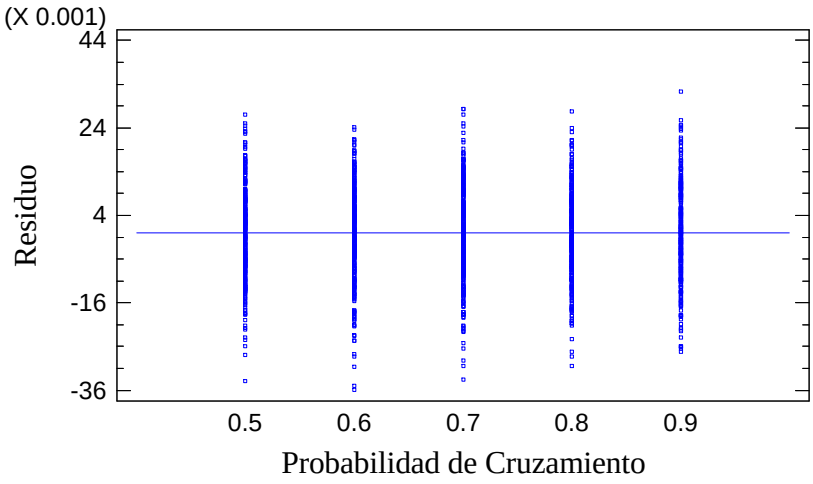


Figura 6.7 Gráfica de residuos contra probabilidad de cruzamiento en el experimento “ta_100_3_1”.

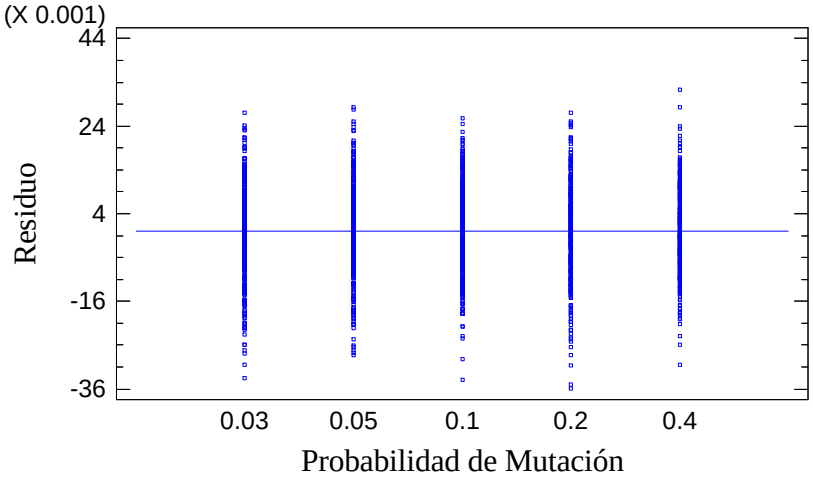


Figura 6.8 Gráfica de residuos contra probabilidad de mutación en el experimento “ta_100_3_1”.

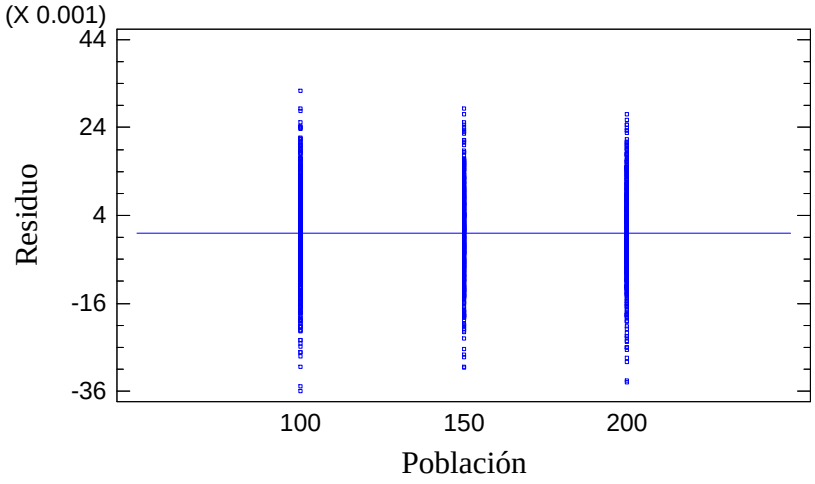


Figura 6.9 Gráfica de residuos contra población en el experimento “ta_100_3_1”.

Como se observa en las Figuras 6.5 - 6.9, las gráficas de los residuos con respecto a los valores ajustados (medias) no muestran una tendencia en la variancia de los residuos a aumentar o disminuir dependiendo del nivel de factor. Así, se cumple la suposición de homocedasticidad de residuos en el experimento.

Finalmente, se comprueba la suposición de independencia graficando los residuos para todas las corridas. La Figura 6.10 muestra la ausencia de una estructura definida o

tendencia en los residuos. La comprobación de idoneidad del modelo para otros experimentos se muestra en el Anexo E.

6.7.5 Análisis de varianza

A través del análisis de varianza se desea probar la hipótesis nula: no existe un efecto significativo de los factores y sus interacciones a la variable respuesta.

Los resultados del experimento se someten a un ANOVA multifactorial para evaluar el efecto de los factores y sus interacciones, los cuales intervienen en el modelo. La Tabla 6.3 muestra los resultados del análisis de varianza a un nivel de confianza del 95%. La contribución de cada factor es medido habiendo quitado los efectos de los otros factores. La primera columna especifica la fuente de variación. El valor de *P-Value* prueba la significancia estadística de los factores o sus interacciones. Los factores cuyo *P-Value* es menor a 0.05 tienen un efecto estadísticamente significativo sobre la variable de respuesta IRMS a un 95% de nivel de confianza.

El análisis de varianza se realizó con los siguientes parámetros: 30 instancias, 100 trabajos, 2, 3 y 6 etapas. La tabla de ANOVA para 50 trabajos se muestra en el Anexo F. Las gráficas de este análisis para 50 trabajos se muestran en el Anexo G.

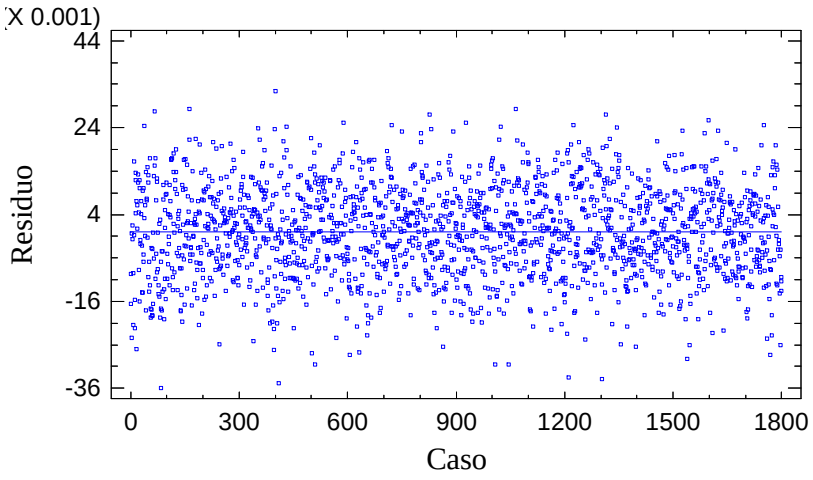


Figura 6.10 Gráfica de residuos frente al orden de ejecución del experimento “ta_100_3_1”.

Tabla 6.3 Tabla ANOVA con resultados del experimento a un nivel de confianza del 95%

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Cross	4.60725	7	0.658179	1452.72	0.0000
B:Mut	0.103056	2	0.0515279	113.73	0.0000
C:Pc	0.00992347	4	0.00248087	5.48	0.0002
D:Pm	0.212351	4	0.0530879	117.17	0.0000
E:Pob	0.347366	2	0.173683	383.35	0.0000
INTERACTIONS					
AB	0.180602	14	0.0129002	28.47	0.0000
AC	0.377783	28	0.0134923	29.78	0.0000
AD	0.289169	28	0.0103275	22.79	0.0000
AE	0.0579243	14	0.00413745	9.13	0.0000
BC	0.0750084	8	0.00937604	20.69	0.0000
BD	0.210307	8	0.0262884	58.02	0.0000
BE	0.00189441	4	0.000473602	1.05	0.3820
CD	0.0974272	16	0.0060892	13.44	0.0000
CE	0.00067277	8	0.0000840963	0.19	0.9930
DE	0.00348887	8	0.000436108	0.96	0.4633
RESIDUAL	24.3949	53844	0.000453067		
TOTAL (CORRECTED)	30.9691	53999			

All F-ratios are based on the residual mean square error.

El *F-Ratio* es la razón entre la media de cuadrados (*Mean Square*) del factor y la media de cuadrados de residuos. En este sentido, un *F-Ratio* alto significa que este factor afecta más a la variable de respuesta. El *P-Value* muestra que los cinco factores contemplados tienen efectos significativos a la variable respuesta, por lo tanto se rechaza la hipótesis nula. Sin embargo, no todas las combinaciones de estos factores tienen significancia; entre las diez interacciones, tres son insignificantes.

Los factores más significativos se muestran en las Figuras 6.11 - 6.14 en el orden de importancia del *F-Ratio*.

La Figura 6.11 muestra los resultados obtenidos para los operadores de cruzamiento. Se aprecia que el operador ST2PX, el cual se propone en esta tesis, es el mejor cruzamiento entre los ocho analizados. Este resultado se explica por el hecho de que este operador de cruzamiento toma en cuenta los tiempos de ajuste de acuerdo a la secuencia de los trabajos. El tiempo de ajuste en este caso es una importante base al momento de la asignación de trabajos a las máquinas paralelas no relacionadas.

La Figura 6.12 muestra la gráfica para el tamaño de la población, donde se aprecia que una población de 200 individuos es estadísticamente más significativa.

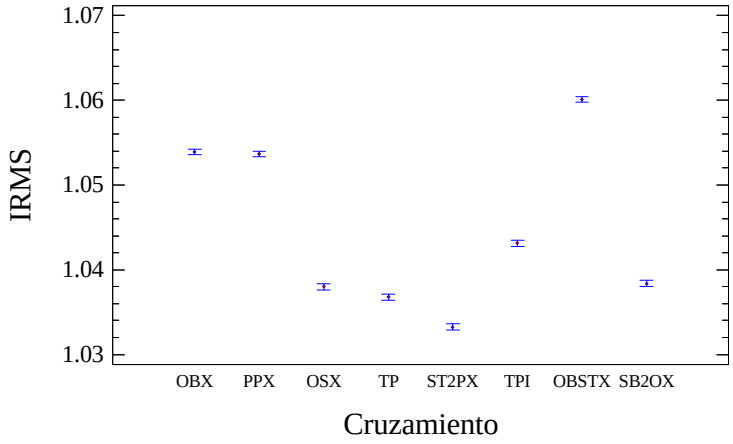


Figura 6.11 Media y 95% de nivel de confianza. Operadores de cruzamiento.

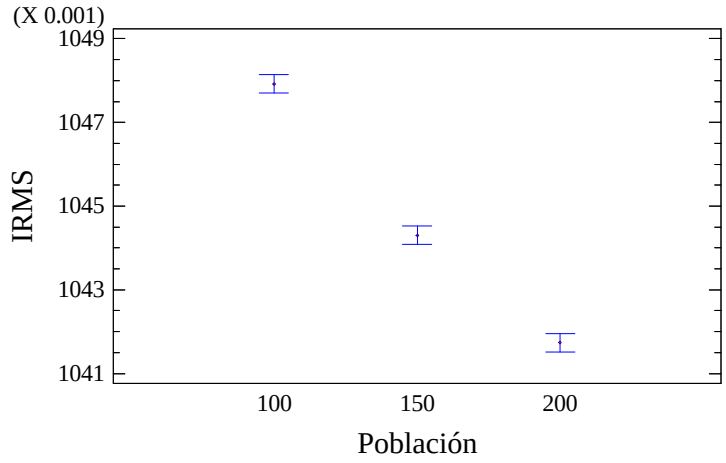


Figura 6.12 Media y 95% de nivel de confianza. Población.

La Figura 6.13 muestra la gráfica de las probabilidades de mutación. La mejor probabilidad de mutación es 0.1.

La Figura 6.14 presenta la gráfica para los tipos de mutación, donde se muestra que el operador “Swap” es el que tiene mejor desempeño en este caso.

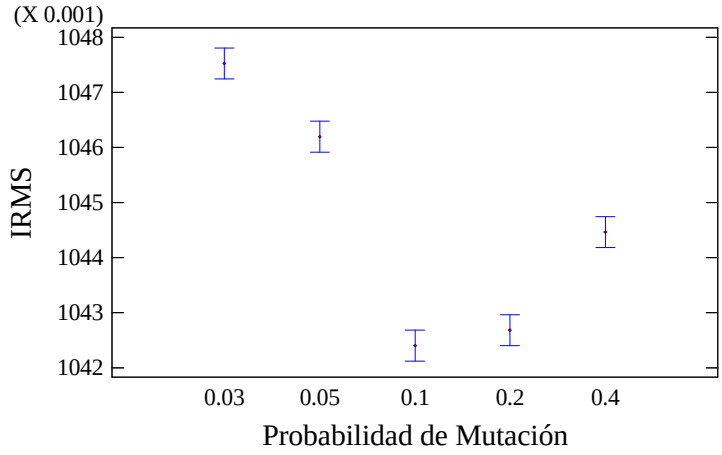


Figura 6.13 Media y 95% de nivel de confianza. Probabilidad de mutación.

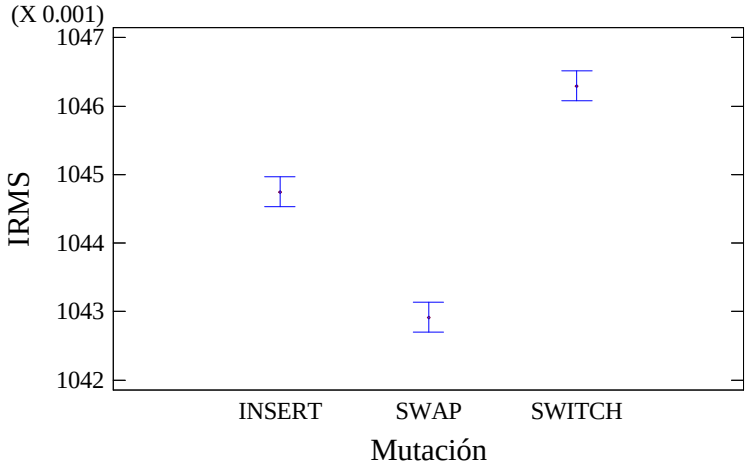


Figura 6.14 Media y 95% de nivel de confianza. Tipos de Mutación.

Siguiendo el orden de importancia en los efectos de la variable de respuesta mostrados por *F-Ratio*, en la Figura 6.15 se muestra la interacción *BD* entre el operador de mutación y la probabilidad de mutación. Enfocando en esta interacción resulta interesante ver que la mutación *Swap* se comporta bien hasta la probabilidad 0.1 y después empeora. Al operador *Insert* le pasa algo similar hasta la probabilidad 0.2 pero con una curva más suave. La mutación *Switch* tiene una tendencia a mejorar conforme aumenta la probabilidad.

La siguiente interacción en orden de importancia es *AC* entre el tipo de cruzamiento y la probabilidad de cruzamiento que muestra la Figura 6.16. Como se observa, el

comportamiento del cruzamiento ST2PX es mejor en todas las probabilidades, con una tendencia a mejorar conforme la probabilidad aumenta.

La Figura 6.17 muestra que especialmente la interacción *AB* entre el cruzamiento ST2PX y la mutación *Swap* se comporta mucho mejor que otras combinaciones.

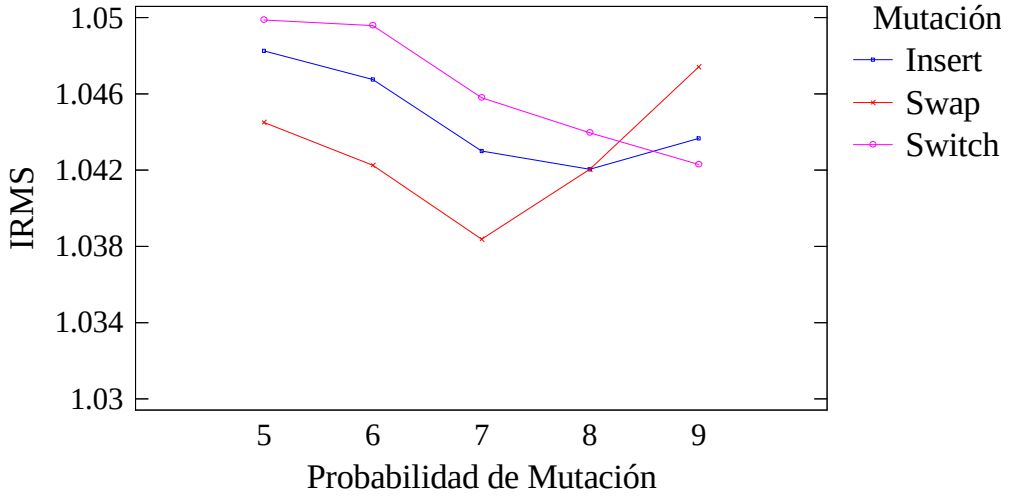


Figura 6.15 Gráfica de interacción entre tipos de mutación y probabilidad de mutación.

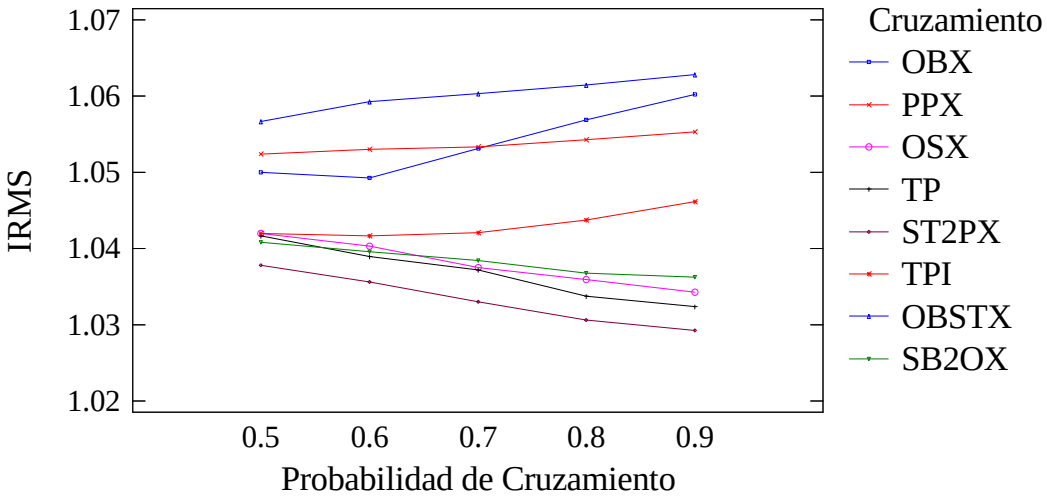


Figura 6.16 Gráfica de interacción entre tipos de mutación y probabilidad de mutación.

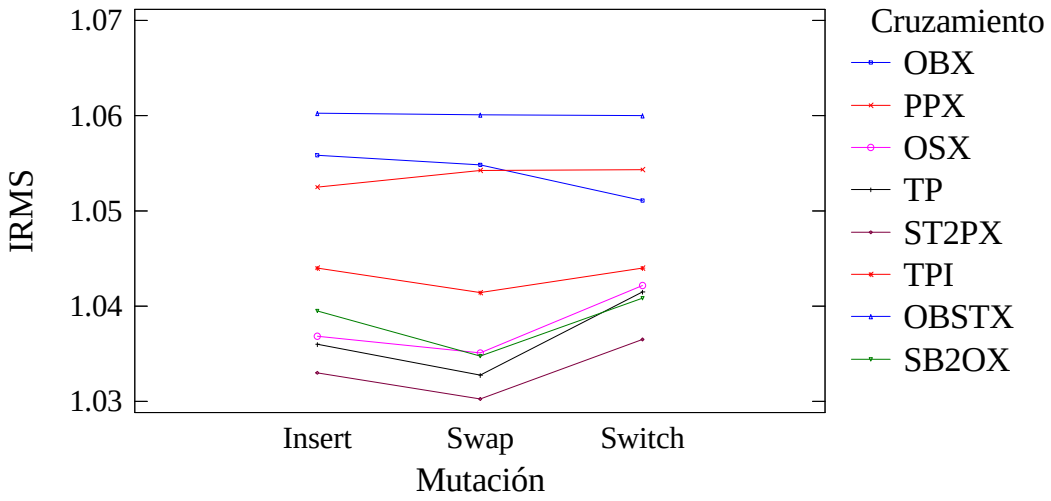


Figura 6.17 Gráfica de interacción entre los operadores de cruzamiento y mutación.

La Figura 6.18 muestra la interacción *AD* entre el operador de cruzamiento y la probabilidad de mutación. Se ve claramente que en todas las probabilidades el algoritmo ST2PX se mantiene con un alto desempeño.

La Figura 6.19 muestra la interacción *BC* entre el tipo de mutación y la probabilidad de cruzamiento. Es interesante ver que confluyen los tres tipos de mutación cerca de la probabilidad 0.9, pero antes de esta probabilidad el operador *Swap* tiene un mejor desempeño.

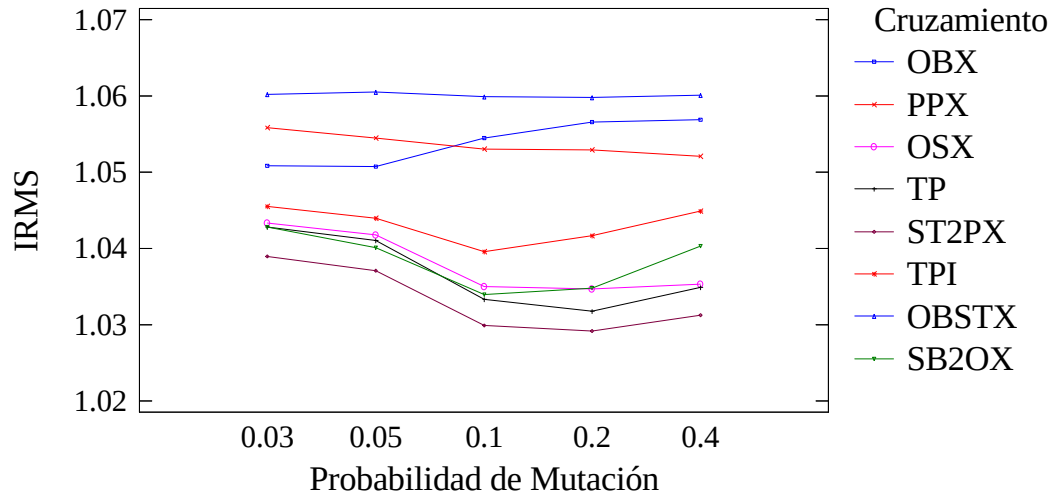


Figura 6.18 Gráfica de interacción entre tipos de cruzamiento y probabilidad de mutación.

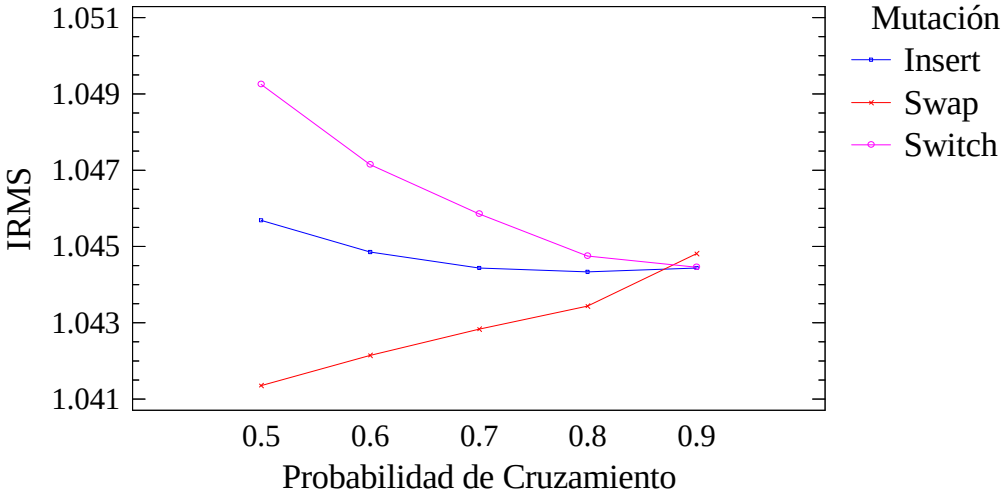


Figura 6.19 Gráfica de interacción entre tipos de mutación y probabilidad de cruzamiento.

La Figura 6.20 muestra la interacción *CD* entre la probabilidad de mutación y la probabilidad de cruzamiento. Se aprecia que hay un mejor comportamiento entre las probabilidades 0.1 y 0.2 respecto a mutación y entre las probabilidades 0.6 a 0.8 respecto a cruzamiento.

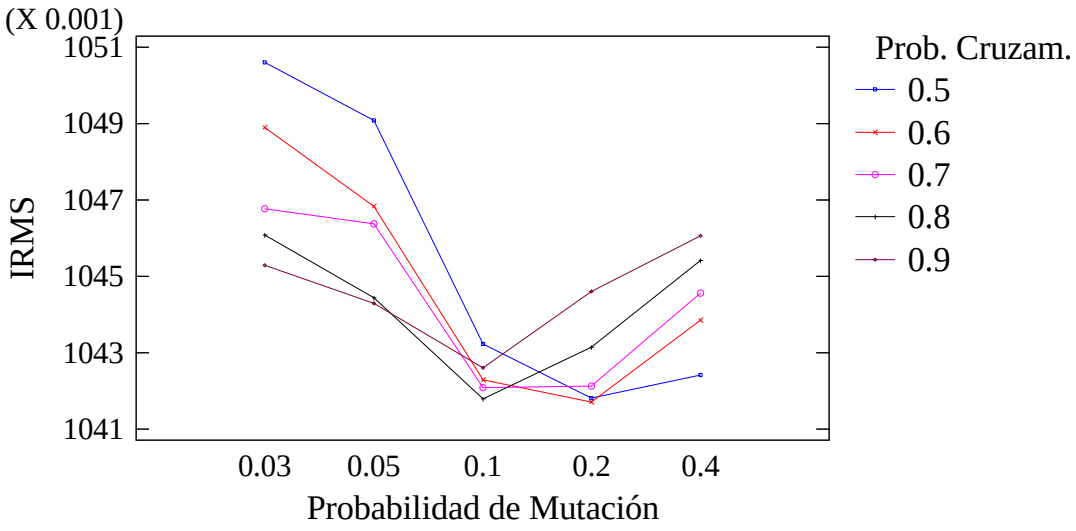


Figura 6.20 Gráfica de interacción entre probabilidad de cruzamiento y probabilidad de mutación.

La Figura 6.21 muestra la interacción *AE* entre el operador de cruzamiento y la población. Se observa claramente que el mejor operador de cruzamiento es el ST2PX para los tres niveles de población.

La Figura 6.22 muestra los resultados obtenidos para el factor de probabilidad de cruzamiento. Como se observa, el mejor resultado se obtiene con la probabilidad de cruzamiento igual a 0.8.

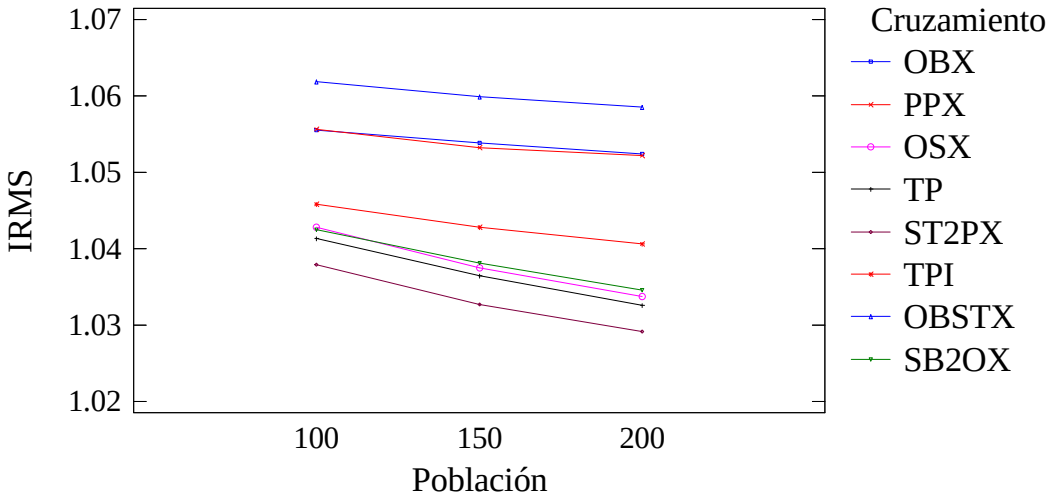


Figura 6.21 Gráfica de interacción entre operadores de cruzamiento y población.

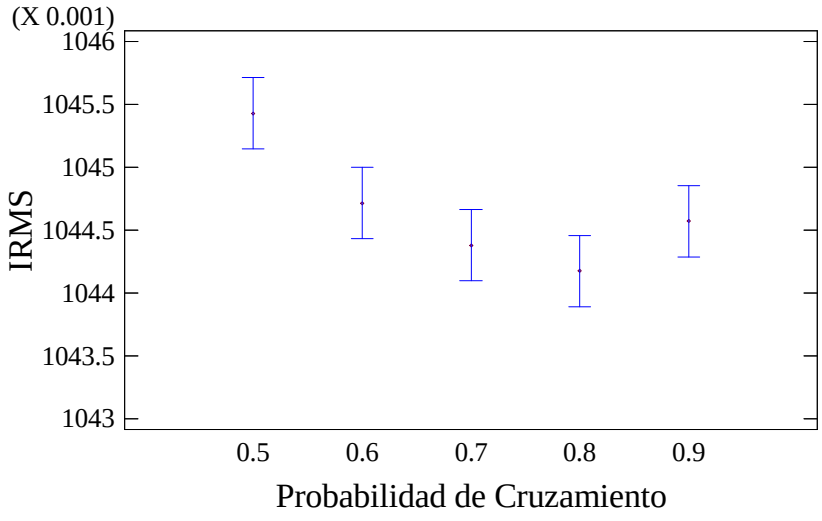


Figura 6.22 Media y 95% de nivel de confianza. Probabilidad de cruzamiento

6.7.6 Resultados del experimento

Después de haber realizado el procedimiento de calibración, se escogen los siguientes niveles de factores que forman el algoritmo propuesto:

Operador de cruzamiento: ST2PX,

Operador de mutación: "*Swap*",

Probabilidad de cruzamiento: 0.8,

Probabilidad de mutación 0.1,

Tamaño de la población: 200,

Se consideran la selección por torneo binario, el criterio de paro y reinicio descritos en el apartado 5.5 (capítulo anterior).

6.8 Evaluación del algoritmo calibrado

Según la revisión de la literatura presentada en el apartado 3.3, a la fecha no se ha publicado un algoritmo para este problema y por consiguiente es imposible hacer una comparación directa. Sin embargo, en (Ruiz & Maroto, 2006) se propone un AG llamado GA_H para resolver un problema similar pero sin la característica de búfer limitado. Este algoritmo ha sido comparado con 9 variantes de métodos metaheurísticos (*NEH heuristic*, *simulated annealing*, *tabu search*, *ant-base algorithms*) (Reeves, 1995; Nawaz et el., 1983; Osman & Potts, 1989; Widmer & Hertz, 1989; Chen, et al., 1995; Murata, et al., 1996; Aldowaisan & Allahvedi, 2003; Rajendran & Ziegler, 2004) y se demuestra que es el mejor para resolver este tipo de problemas. Para efectos de comparación, se introduce una adaptación del algoritmo GA_H a una versión más real del problema, donde se consideran búferes limitados con diferentes tamaños en cada máquina. Esta adaptación recibe el nombre de GA_{HBC}. Los resultados de la comparación entre el algoritmo adaptado GA_{HBC} y del algoritmo calibrado GA_{SBC} se muestra en las Tablas 6.4 y 6.5.

Las Figuras 6.23 y 6.24 muestran la comparación del IRMS recibido por los algoritmos GA_{SBC} y GA_{HBC} para los casos de 50 y 100 trabajos respectivamente. Como se aprecia, el algoritmo GA_{SBC} supera al algoritmo GA_{HBC} en todos los experimentos.

Tabla 6.4 IRMS, 50 trabajos

Instancia		Mejor solución	GA_{SBC}		GA_{HBC}	
			Cmax	**IRMS**	**Cmax**	**IRMS**
50 trabajos 2 etapas (50x2)	1	600	602	0.3333	641	6.8333
	2	1574	1635	3.8755	1698	7.8780
	3	1369	1416	3.4332	1673	22.2060
	4	2531	2531	0.0000	2610	3.1213
	5	2485	2485	0.0000	2547	2.4950
	6	2519	2519	0.0000	2654	5.3593
	7	587	587	0.0000	595	1.3629
	8	5088	5121	0.6486	5191	2.0244
	9	1193	1234	3.4367	1295	8.5499
	10	2460	2505	1.8293	2582	4.9593
				1.3557		**6.4789**
50 trabajos 3 etapas (50x3)	11	985	997	1.2183	1026	4.1624
	12	2518	2572	2.1446	2637	4.7260
	13	1664	1678	0.8413	1792	7.6923
	14	1708	1771	3.6885	1804	5.6206
	15	2476	2532	2.2617	2606	5.2504
	16	5262	5284	0.4181	5331	1.3113
	17	1671	1694	1.3764	1806	8.0790
	18	850	868	2.1176	908	6.8235
	19	1208	1222	1.1589	1306	8.1126
	20	2492	2583	3.6517	2636	5.7785
				1.8877		**5.7557**
50 trabajos 6 etapas (50x6)	21	1134	1185	4.4974	1231	8.5538
	22	2652	2712	2.2624	2862	7.9186
	23	2237	2412	7.8230	2747	22.7984
	24	1350	1350	0.0000	1416	4.8889
	25	5363	5402	0.7272	5434	1.3239
	26	5293	5351	1.0958	5440	2.7773
	27	4796	5045	5.1918	5137	7.1101
	28	1160	1186	2.2414	1243	7.1552
	29	5196	5229	0.6351	5256	1.1547
	30	5318	5344	0.4889	5426	2.0308
				2.4963		**6.5712**

Tabla 6.5 IRMS para 100 trabajos

Instancia		Mejor solución	GA_{SBC}		GA_{HBC}	
			Cmax	**IRMS**	**Cmax**	**IRMS**
100 trabajos 2 etapas (100x2)	31	1187	1229	3.5383	1244	4.8020
	32	1260	1299	3.0952	1329	5.4762
	33	2419	2499	3.3072	2586	6.9037
	34	1451	1487	2.4810	1500	3.3770
	35	4875	5090	4.4103	5185	6.3590
	36	10066	10083	0.1689	10365	2.9704
	37	4847	5016	3.4867	5202	7.3241
	38	3224	3304	2.4814	3429	6.3586
	39	2422	2460	1.5690	2589	6.8951
	40	4904	5057	3.1199	5129	4.5881
				2.7658		**5.5054**
100 trabajos 3 etapas (100x3)	41	1801	1855	2.9983	1894	5.1638
	42	3304	3352	1.4528	3478	5.2663
	43	4987	5228	4.8326	5329	6.8578
	44	2433	2495	2.5483	2576	5.8775
	45	2436	2500	2.6273	2594	6.4860
	46	3279	3333	1.6468	3391	3.4157
	47	10031	10063	0.3190	10308	2.7614
	48	3261	3385	3.8025	3513	7.7277
	49	2462	2535	2.9651	2644	7.3924
	50	1244	1304	4.8232	1463	17.6045
				2.8016		**6.8553**
100 trabajos 6 etapas (100x6)	51	5286	5415	2.4404	5629	6.4888
	52	4996	5150	3.0825	5414	8.3667
	53	10348	10375	0.2609	10632	2.7445
	54	9795	9814	0.1940	10053	2.6340
	55	10530	10600	0.6648	10833	2.8775
	56	2124	2181	2.6836	2277	7.2034
	57	3438	3513	2.1815	3615	5.1483
	58	5150	5363	4.1359	5507	6.9320
	59	10387	10458	0.6835	10547	1.5404
	60	10004	10271	2.6689	10472	4.6781
				1.8996		**4.8614**

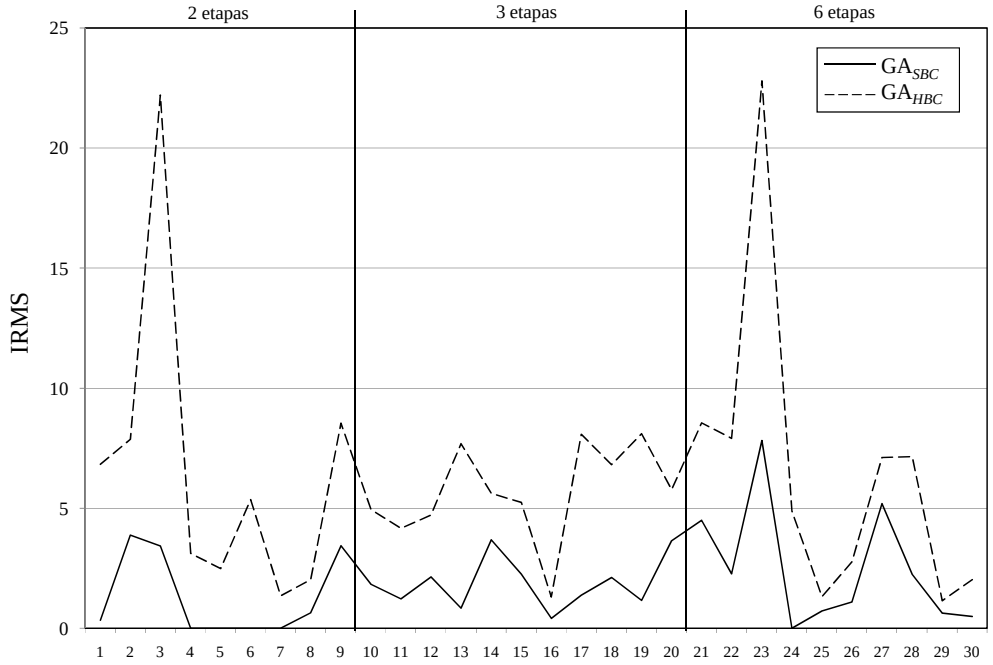


Figura 6.23 Comparación del IRMS recibido por los algoritmos GA$_{SBC}$ y GA$_{HBC}$, 50 trabajos.

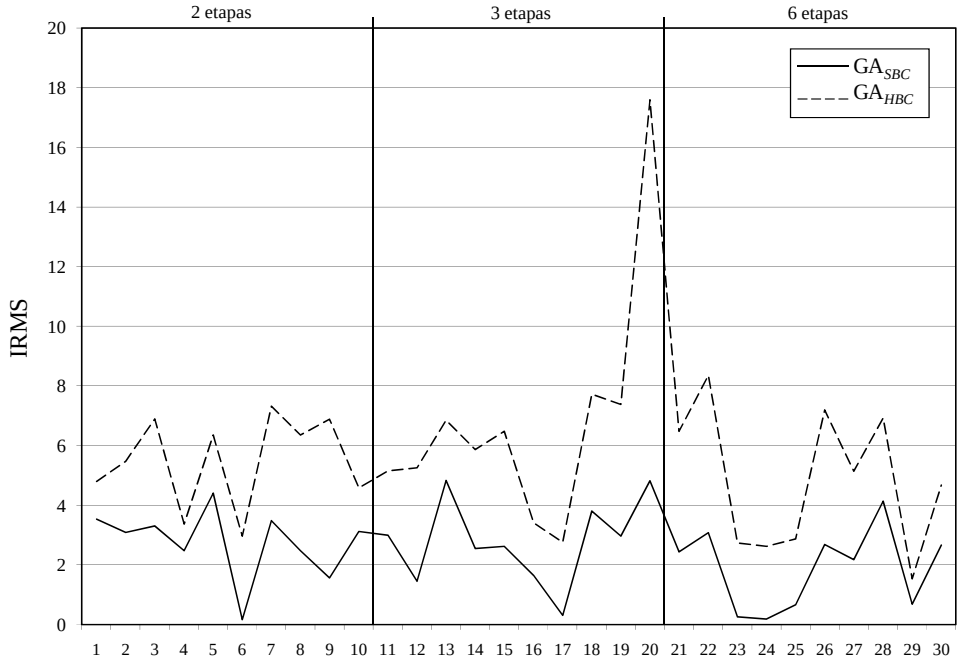


Figura 6.24 Comparación del IRMS recibido por los algoritmos GA$_{SBC}$ y GA$_{HBC}$, 100 trabajos.

La Tabla 6.6 muestra los IRMS promedios de los algoritmos GA_{SBC} y GA_{HBC}. GA_{SBC} supera al algoritmo GA_{HBC} en todos los experimentos en el intervalo entre 99% y 378% dependiendo del caso.

La Tabla 6.7 muestra el incremento relativo promedio del algoritmo GA_{SBC} respecto al algoritmo GA_{HBC} en términos de calidad de solución:

$$\frac{IRMS_{SBC} - IRMS_{HBC}}{IRMS_{HBC}} \cdot 100$$

Los resultados exponen que el algoritmo GA_{SBC} supera al algoritmo GA_{HBC} en todos los experimentos en el rango de 2.58% y 4.60%.

La razón del éxito del algoritmo propuesto se debe al nuevo operador de cruzamiento y al incremento del número de iteraciones debido al novedoso criterio de paro (Algoritmo 6.1, Apartado 6.5), el cual permite encontrar mejores soluciones debido a una regeneración estratégica de la población después de un número de iteraciones sin mejora, saliendo cada vez de los óptimos locales.

Tabla 6.6 IRSM promedio

Instancias de 50 trabajos				Instancias de 100 trabajos			
Instancia	GA_{SBC}	GA_{HBC}	%	Instancia	GA_{SBC}	GA_{HBC}	%
50 x 2	1.3557	6.4789	377.9	100 x 2	2.7658	5.5054	99.1
50 x 3	1.8877	5.7557	204.9	100 x 3	2.8016	6.8553	144.7
50 x 6	2.4963	6.5712	163.2	100 x 6	1.8996	4.8614	155.9
Media	1.9132	6.2686	227.7	Media	2.489	5.7407	130.6

Tabla 6.7 Incremento relativo promedio del algoritmo GA_{SBC} respecto al algoritmo GA_{HBC}

Instancia	%	Instancia	%
50 x 2	-4.60	100 x 2	-2.58
50 x 3	-3.62	100 x 3	-3.70
50 x 6	-3.64	100 x 6	-2.80
Media	-3.95	Media	-3.03

Los tiempos CPU de los algoritmos se presentan en la Tabla 6.8 y las Figuras 6.25 y 6.26. Las instancias 1 a 30 corresponden al grupo de 50 trabajos, y las instancias 31 a 60 corresponden al grupo de 100 trabajos. Como se aprecia en estas gráficas, GA_{SBC} es un poco más lento que GA_{HBC}, sin embargo el incremento de tiempo de cómputo no es significativamente alto.

Tabla 6.8 Tiempo CPU en segundos

Instancia	**GA_{SBC}**	**GA_{HBC}**	**Instancia**	**GA_{SBC}**	**GA_{HBC}**
1	17.98	8.88	31	51.64	28.41
2	17.80	12.3	32	53.56	24.36
3	16.64	8.34	33	50.14	21.28
4	13.70	7.25	34	61.62	26.75
5	14.47	8.38	35	53.98	53.53
6	20.69	8.06	36	65.52	113.47
7	12.58	8.61	37	58.70	112.28
8	10.88	11.52	38	49.77	76.39
9	11.11	7.09	39	55.44	107.88
10	11.98	7.53	40	63.72	127.06
11	20.08	14.36	41	68.78	40.81
12	20.84	14.58	42	66.97	37.62
13	24.38	12.23	43	53.41	27.55
14	16.44	10.55	44	67.56	31.58
15	22.89	15.92	45	59.06	35.47
16	23.52	17.83	46	67.03	31.42
17	19.27	11.16	47	74.31	57.61
18	20.64	15.89	48	61.80	34.12
19	22.30	15.42	49	71.27	35.02
20	19.22	14.44	50	75.22	34.06
21	38.88	34.38	51	106.27	59.56
22	43.14	30.42	52	134.69	73.94
23	38.27	31.23	53	102.42	151.75
24	33.16	27.53	54	101.28	124.25
25	48.80	49.12	55	166.17	117.84
26	36.31	39.27	56	123.33	76.59
27	56.44	53.58	57	131.31	78.64
28	37.48	34.25	58	113.52	96.2
29	64.16	66.81	59	182.45	152.3
30	35.42	57.2	60	204.86	169.06
	43.21	**42.38**		**136.63**	**110.01**

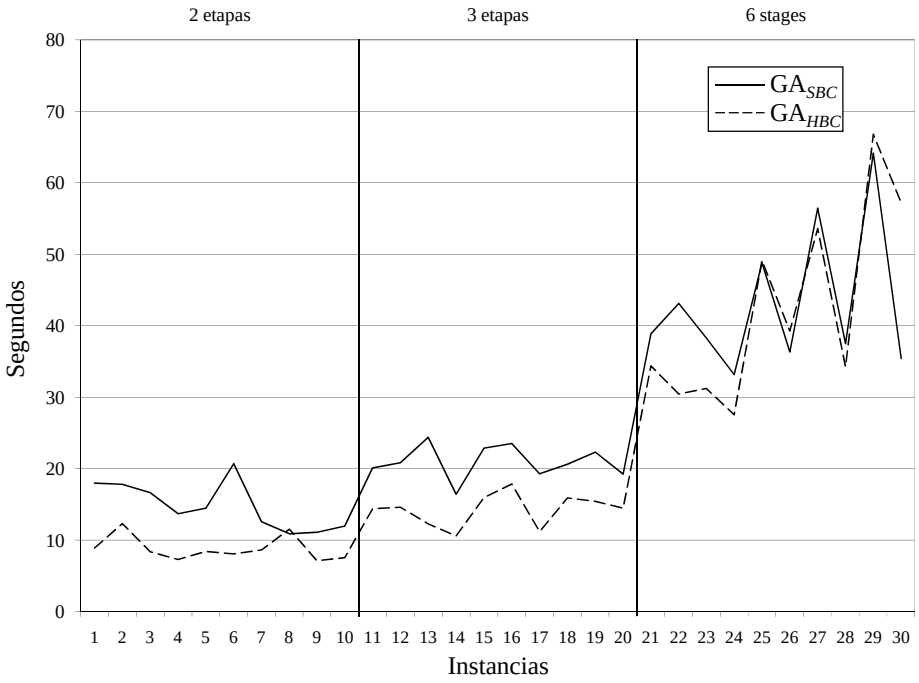


Figura 6.25 Tiempo CPU (en segundos), 50 trabajos.

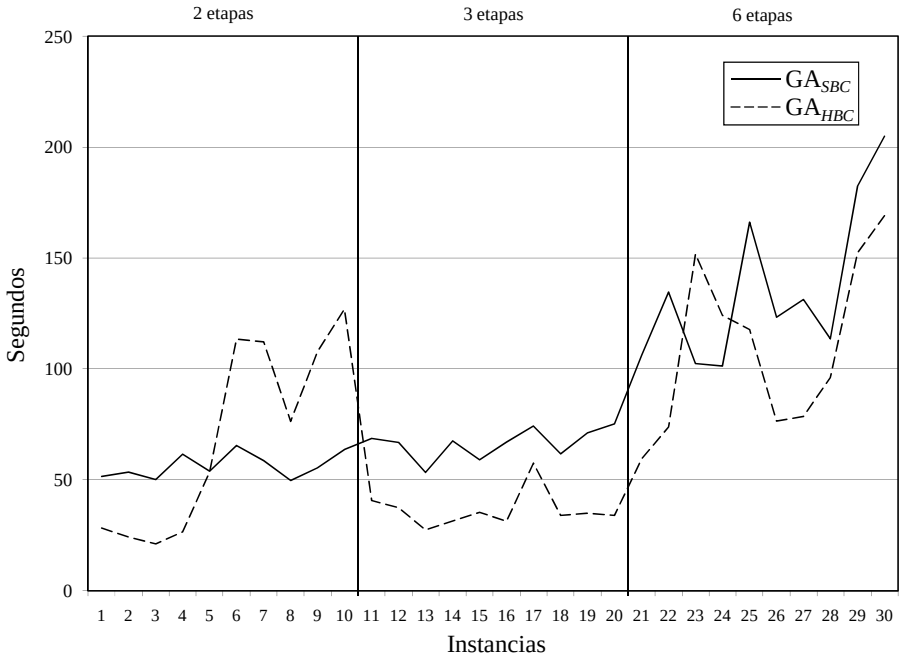


Figura 6.26 Tiempo CPU (en segundos), 100 trabajos

A continuación se comparan los algoritmos GA_{SBC} y GA_{HBC} utilizando la herramienta del análisis de varianza (ANOVA) con un nivel de confianza del 95% (Tabla 6.9). La hipótesis nula es: los resultados obtenidos por los algoritmos GA_{SBC} y GA_{HBC} son estadísticamente iguales.

Como *P-Value* es menor que 0.05, entonces hay una diferencia significativa entre los dos algoritmos, es decir, se rechaza la hipótesis nula. En la Figura 6.27 se comparan gráficamente los resultados de los algoritmos GA_{SBC} y GA_{HBC}. Como se observa, el algoritmo GA_{SBC} es estadísticamente mejor que el algoritmo GA_{HBC} para este problema de TFH.

Los resultados de otra serie de experimentos muestra el comportamiento de los algoritmos en intervalos de tiempo fijo. Esta situación tiene importancia bajo circunstancias de tiempos limitados para la obtención de resultados. Se considera cuatro intervalos de tiempo basados en los tiempos CPU de los experimentos previos: tiempo CPU mínimo reducido en un 30%; tiempo CPU mínimo; tiempo CPU máximo; tiempo CPU máximo incrementado en un 30%.

Tabla 6.9 Tabla ANOVA con resultados de comparación de los algoritmos GA_{SBC} y GA_{HBC}

Analysis of Variance for IRMS - Type III Sums of Squares

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Alg	158,604	1	158,604	31,11	0,0000
RESIDUAL	295,719	58	5,0986		
TOTAL (CORRECTED)	454,323	59			

All F-ratios are based on the residual mean square error.

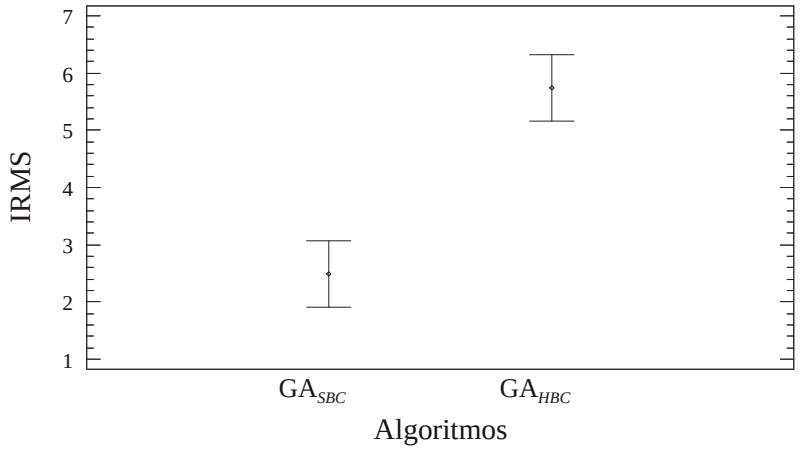


Figura 6.27 Media y 95% de nivel de confianza. Algoritmos

El desempeño de los algoritmos con el mismo tiempo CPU se muestra en la Tabla 6.10. Se observa que al incrementarse el tiempo, el algoritmo GA_{SBC} obtiene mejores resultados, mientras que el algoritmo GA_{HBC} no muestra una mejora significativa.

Tabla 6.10 IRMS promedio con el tiempo CPU especificado

Instancia	Tiempo CPU	GA_{SBC}	GA_{HBC}
50 x 2	5.5	2.39	6.15
	7.1	2.12	6.79
	20.7	1.09	5.36
	26.9	0.51	5.79
50 x 3	8.1	3.04	6.58
	10.6	2.66	6.98
	24.4	1.84	6.20
	31.7	1.16	6.16
50 x 6	21.2	2.96	6.59
	27.5	2.59	6.68
	66.8	2.82	5.26
	86.9	2.88	5.67
100 x 2	16.4	4.07	5.43
	21.3	3.32	5.86
	127.1	1.21	4.63
	165.2	0.81	4.70
100 x 3	21.2	4.31	7.09
	27.6	3.60	7.02
	75.2	1.54	6.63
	97.8	2.04	6.34
100 x 6	45.8	2.97	5.68
	59.6	2.40	5.02
	204.9	1.39	4.03
	266.3	1.18	3.67

6.9 Conclusiones del capítulo

Se presenta un AG eficiente GA_{SBC} que resuelve un complejo problema de TFH con tiempos de ajuste dependientes de la secuencia de trabajos, máquinas paralelas no relacionadas, restricciones de disponibilidad de máquinas y búfer limitado. Este problema es común en la industria electrónica de televisiones, en particular, en la línea de auto

inserción de componentes en tableros de circuito impreso, sin embargo, hasta el momento no ha sido publicado en la literatura. Por lo tanto, el modelo es nuevo y original.

El algoritmo GA_{SBC} utiliza como base el algoritmo GA_{BC} descrito en el capítulo 5 y desarrollado para un modelo similar pero sin búfer limitado. Emplea elementos originales introducidos en el desarrollo del AG GA_{BC}: un operador de cruzamiento, condiciones de reinicio y criterio de paro.

La restricción de búfer limitado se resuelve de la siguiente manera: En cada máquina se verifica el número de trabajos en su búfer. Si el búfer se satura, el flujo continúa por otra máquina de la misma etapa, considerando el mismo criterio de elección: cada trabajo es asignado a la máquina que pueda finalizar el trabajo en el menor tiempo posible en cada etapa, tomando en consideración las diferentes velocidades, tiempos de ajuste, disponibilidad de máquinas y el tamaño del búfer.

El algoritmo GA_{SBC} ha sido calibrado experimentalmente aplicando 1800 algoritmos alternativos a un conjunto de 60 problemas y tomando en cuenta 5 factores: tipo de cruzamiento, tipo de mutación, probabilidad de cruzamiento, probabilidad de mutación y población. Se confirma la idoneidad del modelo a través del análisis residual: normalidad, homocedasticidad e independencia de residuos. De acuerdo a los resultados de la calibración se evalúan los efectos que producen los factores y sus combinaciones a la variable respuesta (ISRM) y se eligen los mejores niveles para cada factor, específicamente: ST2PX como tipo de cruzamiento, tipo de mutación Swap, probabilidad de cruzamiento 0.8, probabilidad de mutación 0.1 y población de 200. Estos niveles son también favorables en los efectos de las combinaciones de los factores.

Con el propósito de evaluar el algoritmo GA_{SBC}, se realiza la adaptación de un algoritmo de referencia GA_{H} que resuelve un problema similar al problema que se aborda pero sin búferes limitados. La adaptación recibe el nombre GA_{HBC}.

Se han realizado dos experimentos computacionales para comparar el algoritmo calibrado GA_{SBC} con GA_{HBC}.

En el primer experimento se obtiene el IRMS de los dos algoritmos con los parámetros característicos para el ambiente de producción que se considera: número de

trabajos 50 y 100; TFH con 2, 3 y 6 etapas; de 1 a10 máquinas por etapa; en total 60 instancias.

A partir de los resultados del experimento se realiza un análisis comparativo de los algoritmos. El IRMS de GA_{SBC} superó el de GA_{HBC}. El algoritmo GA_{SBC} es mejor que GA_{HBC} en todas las instancias y lo supera por un margen entre 99% y 378%; en promedio, 328% para las instancias de 50 trabajos y 231% para las instancias de 100 trabajos. El algoritmo GA_{SBC} demostró un mayor acercamiento a la mejor solución respecto a GA_{HBC} en base al incremento relativo promedio, entre un 2.58% y 4.60%. La comparación del tiempo CPU requerido muestra que el algoritmo GA_{SBC} es ligeramente más lento, en promedio 1 segundo para 50 trabajos y 27 segundos para 100 trabajos, lo que comprueba que el incremento de tiempo no es significativamente alto.

Los resultados del experimento se han sometido a un análisis de varianza unifactorial con un 95% de nivel de confianza considerando el algoritmo como un factor con dos niveles: GA_{HBC} y GA_{SBC}. El valor de *P-Value* obtenido confirmó que hay diferencia estadísticamente significativa entre los resultados producidos por los algoritmos, resultando ser mejor el algoritmo GA_{SBC} .

El segundo experimento es realizado para investigar el comportamiento del algoritmo bajo circunstancias de tiempo limitado. Los resultados confirman que el algoritmo GA_{SBC} obtiene mejores resultados con los mismos intervalos de tiempo.

Resumiendo lo anterior se concluye que el algoritmo GA_{SBC} muestra los mejores resultados en comparación con el algoritmo de referencia.

Finalmente, cabe mencionar que el algoritmo GA_{SBC} cubre las características del modelo y por lo tanto es aplicable al entorno de producción real que sirvió como caso de estudio. Sin embargo, la implantación del algoritmo para la programación de la producción en esta industria, requiere las siguientes consideraciones:

1) La empresa maneja casilleros o estantes para almacenar temporalmente tableros del mismo tipo entre las etapas. El número de tableros en cada estante es fijo. Cada estante debe ser considerado como una unidad de trabajo.

2) El espacio físico en cada máquina para almacenar temporalmente el WIP (*Work in Process*) o trabajo en proceso, debe ser considerado en base al número de estantes que el espacio almacena.

3) El tiempo de pasar un trabajo de una máquina a otra, debe incluirse en los tiempos de procesamiento. Aunque dicho tiempo es relativamente corto comparado con los tiempos de procesamiento, la suma total (de esos pequeños tiempos) resulta merecedora de ser tomada en cuenta para el logro de los objetivos de minimizar el tiempo total de la producción.

Capítulo 7

Conclusiones y futuras líneas de trabajo

7.1 Conclusiones y aportaciones

En esta tesis se aborda un complejo problema de programación de la producción, común en la industria electrónica de televisiones, específicamente en la sección de auto inserción, donde máquinas automatizadas insertan componentes en tableros de circuito impreso a través de procesos que implican de dos a seis etapas.

Se ha formalizado el modelo de la planta productiva que se describe a través de un TFH con máquinas paralelas no relacionadas, tiempos de ajuste dependientes de la secuencia de los trabajos, restricciones de elegibilidad de máquinas y espacio limitado para el almacenamiento temporal de los trabajos.

Se ha realizado una extensa revisión del estado del arte. Las investigaciones dedicadas a problemas de programación de la producción inician con la publicación de S. M. Johnson (1954) quien propone una solución óptima al problema de procesar un conjunto de trabajos en dos máquinas sucesivas. Cuando el número de máquinas sucesivas es 3 ó más, el problema es de tipo **NP**-Difícil. A través de los años aparecen contribuciones en el campo de las técnicas exactas (aplicándose sin embargo a problemas muy concretos y de tamaño reducido), técnicas heurísticas (que proponen soluciones aproximadas) y más recientemente ha proliferado el uso de técnicas más avanzadas como son los algoritmos metaheurísticos (recocido simulado, búsqueda tabú, búsqueda local iterativa, GRASP, algoritmos genéticos).

El TFH se empieza a estudiar desde el año 1971. Se inicia dando énfasis en problemas de dos etapas. Con el transcurrir del tiempo el número de propuestas para el TFH se ha incrementado, así como el número de etapas y mayores restricciones abordando cada vez problemas mucho más complejos. El criterio de optimización más estudiado es la minimización del C_{max}. Las configuraciones de recursos más frecuentes en las publicaciones son grupos de máquinas paralelas idénticas debido a la insuficiencia de métodos de resolución de problemas más complejos. En los últimos años ha aumentado el número de publicaciones hacia entornos de producción reales. Sin embargo, sigue siendo una realidad que aunque se ha investigado mucho en esta área, existe todavía una brecha importante entre la teoría y las aplicaciones prácticas.

El modelo de producción que se aborda en esta tesis no había sido considerado en la literatura. Para su solución se proponen: dos algoritmos heurísticos basados en el método de dicotomía para el caso de dos etapas y un AG para el modelo completo y caso general.

El problema de TFH2 es **NP**-Difícil y por tanto no existe un algoritmo polinomial que obtenga una solución óptima para un caso general. El algoritmo HA_{1+m} presentado en esta tesis se aplica al TFH2 con una máquina en la primera etapa y el algoritmo HA_{M+m} para el caso general del TFH2 cuando las dos etapas tienen múltiples máquinas. Estos algoritmos requieren una asignación previa mediante reglas de prioridad y son aplicables en circunstancias específicas del modelo cuando los tiempos de ajuste no dependen de la secuencia de los trabajos y por ende se incluyen en los tiempos de procesamiento. Ambos algoritmos se basan en el método de dicotomía, el cual representa un esquema eficiente y económico para la optimización combinatoria, en particular, para la programación de la producción. La solución óptima se localiza en un intervalo, sobre el cual se realiza la búsqueda dicotómica.

La aplicación del método de dicotomía en optimización combinatoria resulta en un procedimiento integral que combina cualidades del método numérico de bisección y el método discreto de búsqueda binaria. Se analizan la convergencia y complejidad del método de dicotomía y se muestra que éste depende del intervalo inicial y de la complejidad del problema de decisión a resolver. A partir del análisis de complejidad del método de dicotomía y los algoritmos HA_{1+m} o HA_{M+m} se concluye que éstos poseen una alta eficiencia y son aplicables a la industria y a sistemas de tiempo real.

Con el propósito de abordar el problema del TFH con múltiples etapas, tiempos de ajuste dependientes de la secuencia, máquinas no relacionadas en cada etapa y elegibilidad de máquinas, se propone el algoritmo genético GA_{BC}. Se introducen tres nuevos operadores de cruzamiento. La asignación de los trabajos a las máquinas en cada etapa se realiza tomando en cuenta la máquina que permita terminar el trabajo en el menor tiempo posible, considerando las restricciones de elegibilidad de máquinas y tiempos de ajuste dependientes de la secuencia de los trabajos. Se innova el criterio de paro y reinicio para evitar que el algoritmo se cicle en óptimos locales, permitiendo la regeneración de la población después de un número de iteraciones sin mejora. Se conserva el 20% de los mejores individuos, el 50% del resto (80%) son reemplazados por mutaciones SHIFT del mejor individuo y el restante de los individuos son generados nuevamente de manera aleatoria. Este algoritmo se compara con el mejor AG conocido en la literatura que resuelve un problema similar, superándolo en un 6.27%.

Este AG se extiende a un TFH de múltiples etapas, con máquinas no relacionadas, tiempos de ajuste dependientes de la secuencia de los trabajos, elegibilidad de máquinas y búfer limitado en cada máquina. Los datos de entrada son tomados considerando el ambiente de producción y basados en las semillas del generador lineal congruente de números pseudoaleatorios propuesto por Taillard (1993) usado para este tipo de problemas. Así mismo, se usa un formato de entrada estándar complementado y adaptado por Ruiz y Maroto (2006) para un TFH. El algoritmo propuesto GA_{SBC} ha sido calibrado a través de experimentos computacionales aplicando 1800 algoritmos alternativos a un conjunto de 60 problemas, en total 108,000 evaluaciones, tomando en cuenta los factores: tipo de cruzamiento, tipo de mutación, probabilidad de cruzamiento, probabilidad de mutación y el tamaño de la población. Se toma como variable respuesta el porcentaje de incremento relativo sobre la mejor solución conocida. Los resultados de los experimentos fueron sometidos a un análisis de varianza multifactorial para probar la hipótesis nula: no existe un efecto significativo de los factores a la variable respuesta. La contribución de cada factor fue medida habiendo quitado los efectos de los otros factores. Los resultados de este análisis muestran que los cinco factores contemplados tienen efectos significativos a la variable respuesta a un 95% de nivel de confianza, por lo cual se rechaza la hipótesis nula, habiendo un efecto significativo de los factores a la variable respuesta.

Para evaluar la calidad del algoritmo calibrado, se realizan dos experimentos. Un experimento fue realizado para analizar el comportamiento del algoritmo bajo circunstancias de tiempo limitado. Los resultados muestran la superioridad del algoritmo propuesto. En otro experimento, donde el algoritmo se detiene de acuerdo al criterio de paro, los resultados (apartado 6.8) muestran una mejora entre el 99% y 378% respecto al algoritmo de referencia.

Otra contribución es el nuevo operador de cruzamiento para AGs, llamado ST2PX, enfocado a la programación de la producción. Este operador destaca ampliamente en los resultados de la evaluación de factores. En base al análisis de varianza multifactorial se muestra que este cruzamiento es el mejor entre los ocho considerados. Su éxito se explica por el hecho de que este operador de cruzamiento toma en cuenta los tiempos de ajuste de las máquinas de acuerdo a la secuencia de los trabajos. El tiempo de ajuste en este modelo es una importante circunstancia al momento de tomar las decisiones respecto a la asignación de los trabajos a las máquinas paralelas no relacionadas. Las condiciones de reinicio dentro del esquema generacional, juntamente con el criterio de paro, son otras innovaciones que se han realizado con éxito.

En resumen, la presente tesis aporta a la solución de un nuevo y complejo problema de TFH basado en un caso real modelado del entorno productivo de la sección de auto inserción para una industria electrónica de televisiones.

7.2 Futuras líneas de trabajo

El análisis de las publicaciones confirma que el modelo bajo estudio es nuevo, por lo tanto, se abre un abanico de investigaciones en el futuro considerando casos particulares, otros criterios de optimización y restricciones adicionales.

La primera línea de investigación futura tiene que ver con casos particulares del modelo. Dentro de las características de un entorno productivo, se tienen restricciones que aumentan la complejidad del problema. Bajo esta perspectiva es interesante conocer el impacto que tienen las restricciones al modelo en general. Un caso particular, donde los tiempos de procesamiento son iguales a uno y el número de máquinas es el mismo en todas las etapas, permite hacer un análisis experimental para determinar qué restricción tiene

mayor efecto en el modelo. Por ejemplo, si en el entorno productivo bajo estudio se encuentra que la restricción de búfer limitado tiene el mayor impacto en el modelo, se tomarían decisiones para ampliar el búfer en cada máquina. Conocer esta información es importante porque permite hacer valoraciones como: o sacrificar espacio físico, o sacrificar tiempo de producción.

Otra línea de investigación futura corresponde al criterio de optimización. En la presente tesis se busca minimizar el tiempo de terminar los trabajos; este es el criterio más utilizado en la literatura. Sin embargo, distintos casos prácticos requieren la consideración de otros criterios, como la minimización del tiempo promedio de finalización, minimización del retraso máximo, minimización del tiempo medio de flujo o criterios múltiples. En investigaciones futuras se pretende abordar el criterio de minimizar el tiempo promedio de finalización para analizar su comportamiento en comparación con el criterio de minimizar el tiempo máximo de terminar los trabajos y escoger el más práctico para la industria. Minimizar el tiempo promedio de finalización implica establecer un tiempo de terminación esperado para cada trabajo; esto es, definir el tiempo límite (*due date*) en que el trabajo debe terminar. Este tiempo generalmente se fija de tal manera que se cumplan las fechas de compromiso en la entrega de los productos. Usualmente se usan funciones de penalización en caso que no se cumplan con los tiempos establecidos. Algunas estrategias que se abordan en estos casos, son: procesar primero el trabajo cuya fecha límite es más próxima, procesar primero el trabajo que tarda menos, procesar el trabajo que tarda más, procesar el trabajo cuyo peso de urgencia (*priority*) es mayor.

Por último, una interesante línea de investigación tiene que ver los períodos de mantenimiento preventivo en las máquinas. El mantenimiento de las máquinas de manera programada disminuye la ocurrencia de no disponibilidad de las máquinas por descompostura (*breakdowns*). Implica el estudio de la probabilidad de fallo, modos de fallo, fiabilidad de los equipos, disponibilidad requerida y la consideración de equipos críticos. El mantenimiento debe programarse de manera que preferentemente no interrumpa la producción. La principal estrategia es programar el mantenimiento preventivo en los periodos de tiempo (huecos), en los cuales las máquinas se encuentran en espera del siguiente trabajo a procesar.

De acuerdo a la revisión de la literatura y observando la tendencia hacia un enfoque real de la producción, se espera que este tipo de modelos motive investigaciones futuras.

REFERENCIAS

Acero-Domínguez, M.J. & Paternina-Arboleda, C.D. (2004). Scheduling Jobs on a K-Stage Flexible Flow Shop using a Toc-Based (Bottleneck) Procedure. *Proceedings of the 2004 Systems and Information Engineering Design Symposium* (*Matthew H. Jones, Stephen D. Patek, and Barbara E. Tawney, eds.*), 295-298.

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science*, 34, 391-401.

Adler, L., Fraiman, N., Kobacker, E., Pinedo, M., Plotnicoff, J. C., & Wu, T. P. (1993). BPSS: A Scheduling Support System for the Packaging Industry. *Operations Research*, 41(4), 641–648.

Aghezzaf, E. H., Artiba, A., Moursli, O., & Tahon C. (1995). Hybrid flowshop problems, a decomposition based heuristic approach. *En Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'95* (pp. 43-56) Marrakech. FUCAM - INRIA.

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

Alcaraz, J., Maroto, C., & Ruiz, R. (2003). Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with Genetic Algorithms. *Journal ofthe Operational Research Society*, 54, 614-626.

Aldowaisan, T. & Allahvedi, A. (2003). New heuristics for no-wait flowshops to minimize makespan. *Computers & Operations Research*, 30, 1219-1231.

Alisantoso, D., Khoo L.P. & Jiang, P.Y. (2003). An immune algorithm approach to the scheduling of a flexible PCB flow shop. *The International Journal of Advanced Manufacturing Technology*, 22, 819-827.

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *OMEGA, The international Journal of Management Science*, 27, 219–239.

Allaoui, H., & Artiba, A. (2004). Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints. *Computers & Industrial Engineering*. 47, 431-450.

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

Azizoglu, M., Cakmak, E., & Kondakci, S. (2001). A flexible flowshop problem with total flow time minimization. *European Journal of Operational Research*, 132, 528-538.

Andrés, C. (2001). *Programación de la Producción en Talleres de Flujo Híbridos con Tiempos de Cambio de Partida Dependientes de la Secuencia. Modelos, Métodos y Algoritmos de Resolución. Aplicación a Empresas del Sector Cerámico*. Tesis Doctoral, Departamento de Organización de Empresas. Universidad Politécnica de Valencia.

Andrés, C., Albarracín, J.M., Tormo, G., Vicens, E., & García-Sabater, J.P. (2005). Group technology in a hybrid flowshop environment: A case study. E*uropean Journal of Operational Research*, 167, 272-281.

Arthanary, L.S., & Ramaswamy, K.G. (1971). An Extension of Two Machine Sequencing Problem. *OPSEARCH, The Journal of the Operational Research Society of India*, 8(4), 10-22.

Babayan, A., & He, D. (2004). Solving the n-job 3-stage flexible flowshop scheduling problem using an agent-based approach. *International Journal of Production Research*, 42(4), 777-799.

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

Bautista, J., Companys, R., Corominas, A., & Mateo, M. (1999). GRWASP: Una generalización de los algoritmos GRASP. Aplicación al ORVP, *III Jornadas de Ingeniería de Organización*, CPDA - ETSEIB, 475-482.

Bellman R. (1957). *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 366p.

Bertel, S., & Billaut, J.-C. (2004). A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*, 159, 651-662.

Bierwirth, C., Mattfeld, D., & Kopfer, H. (1996). On permutation representations for scheduling problems. En: H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (editors), *Parallel Problem Solving from Nature - PPSN IV*, Vol. 1141 (pp. 310-318). Berlin, Germany: LNCS, Springer.

Błazewicz, J., Lenstra, J. K., & Rinnooy-Kan, A. H. G. (1983). Scheduling Subject to Constraints: Classification and Complexity. *Discrete Applied Mathematics*, 5, 11-24.

Brah, S.A., & Hunsucker, J.L. (1991). Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research,* 51, 88-99.

Brah, S. A., & Loo, L. L. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, 113, 113-122.

Botta-Genoulaz, V. (2000). Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *International Journal of Production Economics*, 64, 101-111.

Burden R. L., & Faires, J. D. (1985). *Análisis Numérico*. Grupo Editorial Iberoamérica, 856p.

Burtseva L. P., Panishev, A. V. (1989). Creación de schedulings en un problema de secuenciación secuencial-paralela de las tareas. "Sistemas automáticos de control y dispositivos de automática", *Editorial de la Universidad Estatal de Jarkov*, Num.91, (pp. 94-100), Jarkov, Ucrania.

Burtseva, L., & Yaurima, V. H. (2004). Algoritmo aproximado para creación de scheduling de tareas de dos etapas empleando el método de bisección. - *Memoria del XXVI Congreso Internacional de Ingeniería Electrónica ELECTRO 2004*, Vol. XXVI, 2004, IT de Chihuahua, Chih., México, pp. 299-304.

Burtseva L., Yaurima V. H., Espinoza, D., & Arce, O. (2005). 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, pp. 65-70.

Campbell, H. G., Dudek, R.A., & Smith, M. L. (1970). An heuristic algorithm for the *n* jobs *m* machine sequencing problem. *Management Science*, 16(10), 630-637.

Carlier, J., & Rebaï, I. (1996). Two banch and bound algorithms for the permutation flow shop problem. *European Journal of Operational Research*, 90, 238-251.

Cerny, V. (1985). Themodynamical approach to the travelling salesman problem: An efficient simulation algorithm, *Journal of Optimization Theory and Applications*, 45, 41-51.

Chandrasekharan, R., & Chaudhuri, D. (1992a). A multi-stage parallel-processor flowshop problem with minimum flowtime. *European Journal of Operational Research*, 57, 111-122.

Chandrasekharan, R., & Chaudhuri, D. (1992b). Scheduling in *n*-jobs, *m*-stage flowshops with parallel processors to minimize makespan. *International Journal of Production Economics*, 27, 137–143.

Chang, J., Yan, W., & Shao, H. (2004). Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times, *Proceeding of the 2004 American Control Conference* (pp. 1412-1416). Boston, Massachusetts June 30.

Chen, C.-L., Vempati, V. S., & Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80, 389-396.

Cheng, T. C. E., Gupta, J. N. D., & Wang, G. (2000). A Review of Flowshop Scheduling Research with Setup Times. *Production and Operations Management*, 9(3), 262-282.

Cleveland, G., & Smith, S. (1989). Using genetic algorithms to schedule flow shop releases, *Proceedings of the Thirt International Conference on Genetic Algorithms* (pp. 160-169). USA: Morgan Kaufmann Publishers.

Coffman, E. G. Jr., & Sethi, R. (1976). Algorithm minimizing mean flow time: Schedule-length properties, *Acta Informática*, 6, 1-14.

Colorni, A., Dorigo, M. & Maniezzo, V. (1992). An Investigation of Some Properties of an Ant Algorithm, *Proceeding of the PPSN-II, Second International Conference on Parallel Problem Solving from Nature*, Manner, R. & Manderick (Eds.), Elsevier, Amsterdam, The Netherlands, 509-520.

Companys-Pascual, R., & Corominas-Subias, A. (1996). Organización de la producción II. Dirección de operaciones 4. Barcelona, España: Ediciones UPC, 236p.

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). *Theory of Scheduling*. Reading, Massachusetts: Addison-Wesley, 304p.

Cormen, T.H., Leiseron, Ch.E., & Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, Massachussets: The MIT Press, 930p.

Djellab, H., & Djellab, K. (2002). Preemptive Hybrid Flowshop Scheduling problem of interval orders. *European Journal of Operational Research*, 137, 37-49.

Dréo, J., Pétrowski, P., Siarry, A., & Taillard, E. (2007). *Metaheuristics for Hard Optimization*. Springer-Verlag, Berlin Heidelberg, Vol.66, Num.3.

Dudek, R. A., Panwalkar, S. S., & Smith, M. L. (1992). The Lessons of Flowshop Scheduling Research. *Operations Research*, 40(1), 7-13.

El-Zahar, M. H., & Rival, I. (1985). Greedy linear extensions to minimize jumps. *Discrete Applied Mathematics*, 11, 143-156.

Engin, O., & Döyen, A. (2004). A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20, 1083-1095.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for computationally difficult set covering problem. *Operations Research Letters*, 8, 67-71.

Feo, T. A., Bard, J., & Holland, S. (1995). Facility-Wide Planning and Scheduling of Printed Wiring Board Assembly. *Operations Research*, 43, 219-230.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2), 117-129.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company, 340p.

Gen, M., & Cheng, R. (1997). *Genetic algorithms & engineering optimization*. New York: John Wiley & Sons, 512p.

Giffer, B., & Thompson, G. L. (1960). Algorithms for Solving Production Scheduling Problems. *Operations Research*, 8, 487-503.

Glover, F. (1990). Tabu Search: A Tutorial. *Interfaces*, 20, 74-94.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 432p.

Gourgand, M., Grangeon, N., & Norre, S. (1999). Metaheuristics for the deterministic hybrid flow shop problem. *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'99* (pp. 136–145). Glasgow: FUCAM - INRIA.

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

Guinet, A. G. (1991). Textile Production Systems: A Succession of Non identical Parallel Processor Shops. *Journal of the Operational Research Society*, 42(8), 655-671.

Guinet, A. G., & Solomon, M. M. (1996). Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. *International Journal of Production Research*, 34(6), 1643-1654.

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

Gupta, J. N. D. (1972). Heuristic algorithms for multistage flowshop scheduling problem. *AIIE Transactions*, 4 (1), 11-18.

Gupta, J. N. D. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem. *Journal of the Operational Research Society*, 39(4), 359-364.

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

Gupta, J. N. D., Hariri, A. M. A., & Potts, C. N. (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, 69, 171-191.

Gupta, J. N. D., & Tunc, E. A. (1998). Minimizing tardy jobs in a two-stage hybrid flowshop. *International Journal of Production Research*, 36(9), 2397-2417.

Gupta, J. N. D., Kruger, K., Lauff, V., Werner, F., & Sotskov, Y. N. (2002). Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers & Operations Research*, 29, 1417-1439.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 228p.

Hong, T.P., & Wang, T.T. (2000). Fuzzy Flexible Flow shops at two machine centers for continuous fuzzy domains. *Information Sciences*, 129, 227-237.

Hong, T.-P., Wang, C.-L., & Wang, S.-L. (2000). A heuristic Gupta-based flexible flow-shop scheduling algorithm. *Systems, Man, and Cybernetics, 2000 IEEE International Conference*, 1, 319-322.

Hong, T.P., & Wang, T. T. (2004). A New Fuzzy Flexible Flow-Shop Algorithm for Continuous Fuzzy Domain. *Information Sciences*, 129, 227-237.

Huang, W., & Li, S. A. (1998). Two-Stage Hybrid Flowshop with Uniform Machines and Setup Times. *Mathl. Comput. Modelling*, 27(2), 27-45.

Huang, P.-Y., Hong, T.-P., Horng, G., & Kao, C.-Y. (2004). Extending the Palmer Algorithm to Solve Group Flexible Flow-shop Problems. *Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, 2, 1902-1907.

Hunsucker, J. L., & Shah, J. R. (1992). Performance of Priority Rules in a Due Date Flow Shop. *OMEGA, The international Journal of Management Science*, 20(1), 73-89.

Hunsucker, J. L., & Shah, J. R. (1994). Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. *European Journal of Operational Research*, 72, 102-114.

Jain, N., & Bagchi, T. P. (2000). Hybridized GAs: Some new results in flowshop scheduling. Pittsburg. *IASTED International Conference on Modelling and Simulation* (MS'2000).

Jain, N., Bagchi, T. P., & Wagneur, E. (2000). Flowshop scheduling by hybridized GA: Some new results. *International Journal of Industrial Engineering*, 7(3), 213-223.

Janiak, A., Kozan, E., Lichtenstein, M., & Oguz, C. (2005). Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion. *International Journal of Production Economics*, 105(2), 407-424.

Jedrzejowicz, J., & Jedrzejowicz, P. (2003). Population-Based Approach to Multiprocessor Task Scheduling in Multistage Hybrid Flowshops. *7th International Conference, "Knowledge-Based & Intelligent Information & Engineering Systems, 2003" Oxford, UK, September 3-5, 2003, Lecture Notes in Computer Science (Springer Berlin / Heidelberg)*, 2773: 279-286.

Jeffcoat, D. E., & Bulfin, R. L. (1993). Simulated annealing for resource constrained scheduling. *European Journal of Operational Research*, 70, 43-51.

Jin, Z.H., Ohno, K. Ito, T. y Elmaghraby, S.E. (2002). Scheduling Hybrid Flowshops in Printed Circuit Board Assembly Lines. *Production and Operations Management Society*, 11:216-230.

Jin, Z., Yang, Z., & Ito, T. (2006). Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100, 322-334.

Johnson, L. A., & Montgomery, D. C. (1974). *Operations Research in Production Planning, Scheduling and Inventory Control.* New York: John Wiley & Sons, 544p.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, 61-68.

Kelley, D. (1995). *Teoría de Autómatas y lenguajes formales*. Madrid, España: Prentice Hall, 302p.

Rosen, K. H. (2004). *Matemática Discreta y sus aplicaciones*. España: McGraw Hill, 888p.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220, 671-680.

Kochhar, S., & Morris, R. J. T. (1987). Heuristic Methods for Flexible Flow Line Scheduling. *Journal of Manufacturing Systems*, 6(4), 299-314.

Kontoravdis, G., & Bard, J. F. (1995). A GRASP for the Vehicle Routing Problem with time windows. *ORSA Journal of Computing*, 7, 10-23.

Koulamas, C., & Kyparisis, G. J. (2000). Asymptotically Optimal Linear Time Algorithms for Two-Stage and Three-Stage Flexible Flow Shops. *Naval Research Logistics*, 47(3), 259-268.

Kyparisis, G.J., & Koulamas, C. (2001). A note on weighted completion time minimization in a flexible flow shop. *Operations Research Letters*, 29:5-11.

Kyparisis, G.J., & Koulamas, C. (2005). A note on makespan minimization in two-stage flexible flow shops with uniform machines. *European Journal of Operational Research*, In Press.

Kyparisis, G.J., & Koulamas, C. (2006). Flexible flow shop scheduling with uniform parallel machines. *European Journal of Operational Research*, 168, 985-997.

Laguna, M., & González-Velarde, J. L. (1991). A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2, 253-260.

Lee, C-Y., & Vairaktarakis, G.L. (1994). Minimizing makespan in hybrid flowshops. *Operational Research Letters*, 16, 149-58.

Lee, G.C., & Kim, Y. D. (2004). A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness. *International Journal of Production Research*, 42(22), 4731-4743.

Lee, G. C., Kim, Y. D., & Choi, S.-W. (2004). Bottleneck-focused scheduling for a hybrid flowshop. *International Journal of Production Research*, 42(1), 165-181.

Lee, C-Y. (2004). Machine scheduling with availability constraints. En: Leung J. Y-T (Ed.), *Handbook of Scheduling*. CRC Press. pp. 22.1-22.13.

Leung, J. Y-T. (2004). *Handbook of Scheduling. Algorithms. Models and Performance Analysis*. Joseph Y-T Leung (Ed). Boca Raton, Florida: Chapman & Hall / CRC Press, 1120p.

Leyva, E., Burtseva, L., & Yaurima, V. (2007). El Concepto de Dicotomía y su uso en Matemáticas. *IV Foro Nacional "La problemática en el aprendizaje de las ciencias básicas"*. Instituto Tecnológico de Mexicali, SEP.

Linn, R., & Zhang, W. (1999). Hybrid Flow Shop Scheduling: A Survey. *Computers and Industrial Engineering*, 37, 57-61.

Lin, H., & Liao, C. (2003). A case study in a two-stage hybrid flow shop with setup time and dedicated machines. *International Journal of Production Economics*, 86, 133-143.

Ling-Huey, S. (2003). A hybrid two-stage flowshop with limited waiting time constraints. C*omputers & Industrial Engineering*, 44, 409-424.

Logendran, R., deSzoeke, P., & Barnard, F. (2005). Sequence-dependent group scheduling problems in flexible flow shops. *International Journal of Production Economics*, In Press.

Luger, G. F. (2002). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. London: Addison Wesley, 928p.

MacCarthy, B. L., & Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1), 59-79.

Martin, B., & Sanz, A. (1997). *Redes neuronales y Sistemas borrosos*. Marid, España: Edit. RAMA, 442p.

McKay, K. N., Pinedo, M., & Webster, S. (2002). Practice-Focused Research Issues for Scheduling Systems. *Production and Operations Management*, 11(2), 249-258.

Metropolis, N., Rosenblund, A. W., Rosenblund, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21, 1087-1092.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag, tercera edición, 387p.

Montgomery, D. C., & Runger, G. C. (1994). *Applied Statistics and Probability for Engineers*. New York: John Wiley & Sons, 1024p.

Montgomery, D. C. (1996). *Introduction to Statistical Quality Control*. New York: John Wiley & Sons, 3rd edition (1st edition, 1985, 2nd edition, 1991), 734p.

Morita, H., & Shio, N. (2005). Hybrid Branch and Bound Method with Genetic Algorithm for Flexible Flowshop Scheduling Problem. *JSME International Journal*, 48(1), 46-52.

Morton, T. E., & Pentico, D. W. (1993). *Heuristic Scheduling Systems*. Chichester, Inglaterra: John Wiley & Sons, Inc, 720p.

Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Multi-Objective Genetic Algorithm and its Applications to Flowshop Scheduling. *Computers and Industrial Engineering*, 30(4), 957-968.

Nawaz,M., Enscore, Jr, E. E., & Ham, I. (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem. *OMEGA, The International Journal of Management Science*, 11(1), 91-95.

Norman, B. A. (1999). Scheduling flowshops with finite buffers and sequence dependent setup times. *Computers and Industrial Engineering*, 36, 163-177.

Nowicki, E., & Smutnicki, C. (1998). The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research*, 106, 226-253.

Nowicki, E., & Smutnicki, C. (1999). Flow Line Scheduling by Tabu Search. En Voß, S., Martello, S., Osman, I. H., y Roucairol, C., editores, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Dordrecht. Kluwer Academic Publishers, pp. 175-189.

Oguz, C. O., Ercan, M. F., Cheng, T. C. E., & Fung, Y. F. (2003). Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. *European Journal of Operational Research*, 149, 390-403.

Oguz, C., Zinder, Y., Do, V. H., Janiak A., & Lichtenstein, M. (2004). Hybrid flow-shop scheduling problems with multiprocessor task systems. *European Journal of Operational Research*, 152, 115-131.

Onwubolu, G. C., & Mutingi, M. (1999). Genetic algorithm for minimizing tardiness in flow-shop scheduling. *Production Planning and Control*, 10(5), 462-471.

Osman, I. H., & Potts, C. N. (1989). Simulated Annealing for permutation flowshop sequencing. *OMEGA International Journal of Management Science*, 17, 551-557.

Osman,I., H. (1995). An introduction to Meta-Heuristics. Lawrence, M., & Wilson, C. Editores. *Operational Research, Tutorial Papers*, 92-122.

Palmer, D. S. (1965). Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum. *Operational Research Quarterly*, 16, 101-107.

Panishev, A., Danilchenko, O., & Skachkov, V. (2004). Introducción a la teoría de complejidad de problemas discretos. Zgitomir, Ucrania: *Editorial de la Universidad Estatal de Zgitomir*, 326 p. (en ucraniano).

Papadimitriou, C. H. (1994). *Computational Complexity*. Reading, Massachusetts: Addison-Wesley, 500p.

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

Ponnambalam, S. G., Aravindan, P., & Chandrasekaran, S. (2001). Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. *Production Planning and Control*, 12(4), 335-344.

Portmann, M. C. (1997). Scheduling methodology: optimization and compusearch approaches I. En Artiba, A., & Elmaghraby, S. E., editores, *The Planning and Scheduling of Production Systems. Methodologies and Applications*, London: Chapman & Hall, pp. 271-300.

Portmann, M. C., Vignier, A., Dardilhac, D., & Dezalay, D. (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107, 389-400.

Rajendran, C., & Chaudhuri, D. (1992). A multi-stage parallelprocessor flowshop problem with minimum flowtime. *European Journal Operation Research*, 57, 111-122.

Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal Operational Research*, 155, 426-438.

Reeves, C. R. (1995). A Genetic Algorithm for Flowshop Sequencing. Computers and Operations Research, 22(1), 5-13.

Reeves, C. R., & Höhn, C. (1996). Integrating Local Search into Genetic Algorithms. En Rayward-Smith, V. J., Osman, I. H., Reeves, C. R., & Smith, G. D., editores, *Modern Heuristic Search Methods*, New York: John Wiley & Sons, pp. 99-115.

Reeves, C. R., & Yamada, T. (1998). Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem. *Evolutionary Computation*, 6(1), 45-60.

Riane, F., Artiba, A., & Elmaghraby, S. E. (1998). A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. *European Journal of Operational Research*, 109, 321-329.

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

Romero, R. (2007). *Sistema Computacional para la evaluación de algoritmos de Planificación de Trabajos en un Taller de Flujo Híbrido*. Tesis de Maestría, Facultad de Ingeniería. Universidad Autónoma de Baja California. México.

Rosen, K. H. (1999). *Discrete Mathematics and Its Applications*. Boston: WCB/McGraw-Hill, 678p.

Rosen K. H. (2004). *Handbook of Discrete and Combinatorial Mathematics*. CRC Press, 1248p.

Ruiz, R. (2003). *Técnicas Metaheurísticas para la Programación Flexible de la Producción*. Tesis doctoral. Departamento de Estadística e Investigación Operativa Aplicadas y Calidad. Universidad Politécnica de Valencia.

Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169, 781-800.

Ruiz, R., Şerifoğlu, F. S., & Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computer Operations Research,* 35(4), 1151-1175.

Salvador, M. S. (1973). A solution to a special case of flow shop scheduling problems. En Elmaghraby, S. E., editor, *Symposium of the Theory of Scheduling and Applications*, New York: Springer-Verlag, pp. 83-91.

Santos, D. L., Hunsucker, J. L., & Deal, D. E. (1995). Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research*, 80, 112-120.

Santos, D. L., Hunsucker, J. L., & Deal, D. E. (1996). An Evaluation of Sequencing Heuristics in Flow Shops With Multiple Processors. *Computers and Industrial Engineering*, 30(4), 681-692.

Sawik, T. (2000). Mixed Integer Programming for Scheduling Flexible Flow Lines with Limited Intermediate Buffers. *Mathematical and Computer Modelling*, 31(13), 39-52.

Sawik, T. (2002). An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modelling*, 36(4-5), September 2002, 461-471.

Shapiro, J. F. (1979). A Survey of Lagrangian Techniques for Discrete Optimization. *Annals of Discrete Mathematics*, 5, 113-138.

Sherali, H. D., Sarin, S. C., & Kodialam, M. S. (1990). Models and algorithms for a two-stage production process. *Production Planning and Control*, 1(1), 27-39.

Silver, E. A., Vidal, R. V., & De-Werra, D. (1980). A Tutorial on Heuristics Methods. *European Journal of Operational Research*, 5(3), 153-162.

Shmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121, 1-15.

Soewandi, H., & Elmaghraby, S. E. (2001). Sequencing three-stage flexible flowshops with identical machines to minimize makespan. *IIE Transactions,* 31, 985-993.

Soewandi, H., & Elmaghraby, S.E. (2003). Sequencing on two-stage hybrid flowshop with uniform machines to minimize makespan. *IIE Transactions*, 35, 467-477.

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

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64, 278-285.

Tang, L., & Liu, J. (2002). A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. *Journal of Intelligent Manufacturing*, 13, 61-67.

Tang, L., Luh, P.B., Liu J., & Fang, L. (2002). Steel-making process scheduling using Lagrangian relaxation. *International Journal of Production Research*, 40(1), 55-70.

Tang, L., Liu, W., & Liu J. (2005). A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. *Journal of Intelligent Manufacturing*, 16, 361-370.

Tang, L., Xuan, H., & Liu, J. (2005). A new Lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. *Computers & Operations Research*, in Press.

Tang, L., & Zhang. Y. (2005). Heuristic Combined Artificial Neural Networks to Schedule Hybrid Flow Shop with Sequence Dependent Setup Times. *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Vol. 3496/2005, Advances in Neural Networks – ISNN 2005, 788-793.

Thomas, J. L., & McClain, J. O. (1993). An overview of production planning. En Graves, S. C., Rinnooy Kan, A. H. G., y Zipkin, P. H., editores, *Logistics of Production and Inventory*, volumen 4 de *Handbooks in Operations Research and Management Science*, Amsterdam: Elsevier Science Publishers, B. V., pp. 333–370.

Thornton, H. W., & Hunsucker, J. L. (2004). A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. *European Journal of Operational Research*, 152, 96-114.

Townsend, W. (1977). Sequencing n jobs on m machines to minimize maximum tardiness: a branch and bound solution. *Management Science*, 23, 1016-1019.

Vancza, J., & Markus, A. (1991). Genetic Algorithms in Process Planning. *Computers in Industry*, 17(2-3), 181-194.

Vazquez, J. A., & Salhi, A. (2005). Performance of Single Stage Representation Genetic Algorithms in Scheduling Flexible Flow Shops. *IEEE 0-7803-9363-5/05/2005*.

Vignier, A., Billaut, J. C., Proust, C., & T'Kindt, V. (1996). Resolution of some 2-stage hybrid flowshop scheduling problems. *IEEE International Conference On Systems Man And Cybernetics*, 4, 2934-2941.

Vignier, A., Billaut, J. C., & Proust, C. (1999). Les Problèmes D'Ordonnancement de Type Flow-Shop Hybride: État de L'Art. *RAIRO Recherche opérationnelle*, 33(2), 117-183.

VoB, S., & Witt, A. (2005). Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. *International Journal Production Economics*. In Press.

Wang, L., & Li, D. (2002). A Scheduling Algorithm for Flexible Flow Shop Problem. *Proceedings of the 4th World Congress on Intelligent Control and Automation*. Vol. 4, 2002, pp. 3106-3108.

Wang, H., Jacob V., & Rolland E. (2003). Design of efficient hybrid neural networks for flexible flow shop scheduling. *Expert Systems*, 20(4), 208-231.

Wang, Z., Xing, W., & Bai, F. (2005). No-wait flexible flowshop scheduling with no-idle machines. *Operations Research Letters*, 33, 609-614.

Wang, L., Zhang, L., & Zheng, D. (2006). An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers and Operations Research archive*, 33(10), 2960-2971.

Weng, M. X. (2000). Scheduling Flow-Shops With Limited Buffer Spaces. En: J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick, eds., *Proceedings of the 2000 Winter Simulation Conference* (pp. 1359- 1363).

Widmer, M., & Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41, 186-193.

Wirth, N. (1976). *Algorithms + data structures = Programs*. Englewood Cliffs, N. J.: Prentice-Hall, 366p.

Witt, A., & Voí, S. (2007). Simple heuristics for scheduling with limited intermediate storage. *Computers and Operations Research*, 34(8), August 2007, 2293-2309.

Wittrock, R. J. (1985). Scheduling algorithms for flexible flow lines. *IBM Journal of Research and Development*, 29(4), 401-412.

Wittrock, R. J. (1988). An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, 36(3), 445-453.

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

Xuan, H., & Tang, L. (2005). Scheduling a hybrid flowshop with batch production at the last stage. *Computers & Operations Research.* In Press.

Yamada, T., & Reeves, C. R. (1997). Permutation Flowshop Scheduling by Genetic Local Search. *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications* (pp. 232-238), Glasgow: GALESIA ’97 IEE.

Yang, W.-H., & Liao, C.-J. (1999). Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2), 143-155.

Yaurima V., Romero R., Burtseva L., & Valle Y. (2006). Aplicación del Método de Dicotomía para Optimización Combinatoria. *XXVIII Congreso Internacional de Ingeniería Electrónica - ELECTRO 2006* (pp. 283-288). IT Chihuahua, Creel, Chih., México, ISSN 1405-2172.

Yaurima, V., Burtseva, L., & Tchernykh, A. (2007). Hybrid Flowshop with Unrelated Machines, Sequence Dependent Setup Time and Availability Constraints: An Enhanced Crossover Operator for a Genetic Algorithm. In: Wyrzykowski et al. (Eds.), *Parallel Processing and Applied Mathematics*, LNCS 4967, Springer-Verlag, (in press).

Zanakis, S. H., & Evans, J. R. (1981). Heuristic Optimization: Why, When and How to Use it. *Interfaces*, 11(5), 84-90.

Zandieh, M., Fatemi Ghomi, S., & Moattar Husseini, S. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, 180, 111-127.

Zhang, W., Yin, C., Liu, J., & Linn, R. J. (2005). Multi-job lot streaming to minimize the mean completion time in m-1 hybrid flowshops. *International Journal Production Economics*, 96, 189-200.

Zhou, D. N., Cherkassy, V., Baldwin, T. R., & Olson, T. R. (1991). A Neural Network Approach to job shop scheduling. *IIEEE transactions on Neural Networks*, 2, 175-179.

ANEXOS

Anexo A Capacidades de las máquinas
CD Tesis: A-Capacidad_Maquinas.pdf

Anexo B Familias de televisiones, modelos y tipos de tableros
CD Tesis: B-Familias_Televisiones.pdf

Anexo C Ejemplo de archivo de instancia
CD Tesis: C-Archivo_Instancia.pdf

Anexo D Resultados del experimento ta_100_3_1
CD Tesis: D-Experimento_ta_100_3_1.pdf

Anexo E Gráficas de idoneidad
CD Tesis: E-Idoneidad.pdf

Anexo F Tabla ANOVA para 50 trabajos
CD Tesis: F-Anova_50_trabajos.pdf

Anexo G Gráficas de ANOVA para 50 trabajos
CD Tesis: G-Graficas_Anova_50_trabajos.pdf