Resumen

El problema objeto de estudio es la programación de piezas en un sistema flowshop híbrido flexible, con máquinas idénticas en paralelo en cada etapa y tiempos de preparación dependientes de la secuencia. El objetivo de este proyecto es implementar un procedimiento heurístico que permita resolver de modo eficiente la programación de las piezas en este entorno con el fin de minimizar el retraso medio en la entrega de las piezas.

El procedimiento implementado es una metaheurística del tipo *Iterated Local Search (ILS)*. Esta metaheurística parte de una solución inicial (que puede ser de mayor o menor calidad) y realiza una búsqueda local del vecindario, hasta encontrar un mínimo local. A continuación empieza la parte iterativa del algoritmo durante la cual se aplican tres operadores: la perturbación para modificar la solución en curso con el fin de escapar del mínimo local, la búsqueda local que tiene la finalidad de analizar el vecindario y encontrar un nuevo mínimo, y el criterio de aceptación que decide qué solución será perturbada en la iteración siguiente. Los parámetros del algoritmo han sido calibrados mediante un diseño de experimentos sobre 12 colecciones de 40 ejemplares, que resultan de la combinación de diferente número de trabajos, diferente número de etapas y del número de máquinas en cada etapa.

El análisis de la eficiencia del procedimiento propuesto ha sido realizado, por comparación con otras tres variantes del mismo, sobre un conjunto de 720 ejemplares. Las variantes implementadas resultan de cambios en la obtención de solución inicial, del vecindario explorado o de la forma en la que se obtiene la programación en las diferentes etapas.

Sumario

1. Glosario

- n : Número de piezas a procesar
- m : Número de etapas que componen el proceso de producción
- mpe_j : Número de máquinas en la etapa *j*
- p_{ij} : Tiempo de proceso de la pieza *i* en la etapa *j*
- S_{ijk} : Tiempo de preparación de la pieza *i* en la etapa *j* cuando es precedido por la pieza *k*
- d_i : Fecha de entrega de la pieza *i*
- w_i : Peso, índice de importancia de la pieza *i*
- T_i : Retraso de la pieza *i*
- C_i : Tiempo de salida de la pieza *i* de la última etapa del sistema
- $C_{\max}$: Instante de terminación de la última pieza
- F_i : Tiempo de permanencia de la pieza *i* en el sistema

2. Introducción

En esta sección se exponen brevemente las necesidades que impulsan la investigación en el campo de la programación de operaciones, y como caso particular, la realización de este proyecto. Sucesivamente se definen los objetivos que se desea lograr y el alcance del trabajo realizado.

2.1. Origen y motivación del proyecto

El éxito y la eficiencia de cualquier empresa de producción están relacionados no solo con los recursos de que esta dispone, sino sobre todo con el modo de utilizarlos. Cada empresa después de haber conseguido los medios de producción necesarios, tiene que resolver la cuestión de cómo explotar al máximo dichos recursos. La programación de operaciones nace para resolver estas cuestiones de forma eficaz.

La definición de un plan de producción optimizado es un problema tan importante como frecuente, ya que es necesario revisar las decisiones cada vez que se modifica la línea productiva o la cartera de productos.

En las empresas del pasado, el problema de la programación de operaciones ha sido resuelto típicamente de forma manual, utilizando criterios basados en la experiencia del personal. Este enfoque, aunque muy flexible, no presenta una base objetiva y tampoco garantiza la calidad de la solución escogida; por eso no se adapta a un escenario de gran competitividad como el actual. De aquí surge la necesidad de herramientas que proporcionen a estos problemas, si no una solución óptima, soluciones de calidad en un tiempo computacional adecuado.

Realizar un plan de producción quiere decir esencialmente contestar a la pregunta: *"¿Qué operación se realiza en cada máquina en un determinado momento?"*. Para hacerlo es necesario resolver tres sub-problemas: la *carga*, la *secuenciación* y la *temporización*, que se describen brevemente a continuación.

Carga: fase en la cual se asigna a cada operación el recurso en que va a ejecutarse

Secuenciación: fase en la cual se define el orden o secuencia en que se ejecutarán las operaciones asignadas al mismo recurso

Temporización: fase en la cual se establecen las fechas de la ejecución

Se pueden resolver estos sub-problemas simultáneamente o por separado (en realidad la temporización siempre se resuelve junta con la secuenciación ya que no es posible desintegrarlas), pero en ambos casos habrá que enfrentarse a un problema combinatorio. Existen procedimientos exactos para la resolución de estos problemas (*Programación entera y mixta o Branch & Bound*) los cuales consiguen proporcionar la solución óptima; pero la dimensión de los problemas que generalmente las empresas deben solucionar, son incompatibles con los tiempos de cálculo y los recursos informáticos necesarios en los métodos exactos. Como consecuencia muchas veces se ha optado por procedimientos heurísticos y metaheurísticos que encuentran soluciones no necesariamente óptimas, pero de buen nivel y en tiempos razonables.

El papel de la investigación en la programación de operaciones entonces es, entre otros, la de diseñar procedimientos heurísticos eficientes, flexibles y que se acerquen a los procesos productivos reales. En este contexto se enmarca, aunque humildemente, este proyecto.

2.2. Objetivo del proyecto

El objetivo del proyecto es implementar una herramienta informática que, introducidas las características de la línea de producción (número de etapas, número de máquina en cada etapa, etc.) y de los trabajos a realizar (tiempo de proceso, tiempos de preparación, fecha de entrega, etc.), realice su programación con el fin de minimizar los retrasos.

El procedimiento implementado se basa en una búsqueda local iterativa, más conocida como *Iterated Local Search* (ILS). Se han calibrado los parámetros principales con un diseño de experimentos sobre un primer juego de datos. El procedimiento se ha programado en *C#.*

La eficiencia del procedimiento, a falta de una base de comparación, se ha medido

analizando, por comparación de los resultados obtenidos, la eficiencia de tres variantes del programa original sobre los mismos juegos de datos.

2.3. Alcance del proyecto

El procedimiento de resolución propuesto es apto para resolver problemas que se puedan esquematizar como un flowshop híbrido flexible con tiempos de preparación dependientes de la secuencia, o como casos particulares de lo mismo. El índice de eficiencia utilizado es el retraso medio, pero una sencilla modificación permitiría poder utilizar el programa con otros índices de eficiencia como el instante máximo de finalización de los trabajos o makespan (C_{max}), ampliando así su campo de aplicación.

3. Programación de operaciones

Definimos la programación de operaciones como sigue [R. Companys 2003 p. 99-100]:

"La programación de operaciones tiene como objeto definir en forma concreta en qué recurso de los disponibles se ejecutara cada una de las operaciones necesarias para la realización de las ordenes de trabajo emitidas y los instantes (fechas) en que tendrá lugar dicha ejecución. Se asocian a la programación de operaciones diferentes subfunciones entre las que cabe señalar las de carga, secuenciación y temporización".

En este apartado se introducirán los conceptos fundamentales de la programación de operaciones, la clasificación de los problemas, las hipótesis generalmente aceptadas en la resolución de dichos problemas, las medidas usadas para calcular la eficiencia de un programa de producción y la nomenclatura.

3.2. El problema del taller mecánico

En este problema se consideran únicamente dos de las subfunciones de la programación de operaciones: la secuenciación y la temporización, suponiendo que la carga haya sido realizada anteriormente. El prototipo de problema de secuenciación suele denominarse *problema del taller mecánico* (*job-shop problem*) y el enunciado básico del problema es el siguiente [R. Companys, 2003, p. 99]:

"n piezas (lotes de piezas, pedidos u órdenes de trabajo) deben realizarse en m máquinas (recursos, secciones, puestos de trabajo). La realización de cada pieza implica la ejecución, en cierto orden establecido, de una serie de operaciones prefijadas donde cada operación está asignada a una de las m máquinas y tiene una duración (tiempo de proceso) determinada y conocida; debe establecerse un programa, es decir, la secuencia de operaciones en cada máquina y el intervalo temporal de ejecución de las operaciones, con el objetivo de optimizar un índice determinado que mide la eficiencia del programa."

Problemas como el descrito se originan en los más diversos sectores, aunque no tengan la menor relación con la mecánica ni con los talleres. De toda forma, para uniformar la nomenclatura, se utilizan sistemáticamente las palabras piezas y máquinas para indicar los conceptos equivalentes en dichos problemas. Es interesante notar como en cualquier problema del taller se puedan distinguir dos secuencias significativas:

la ruta: secuencia establecida *a priori*, en que cada una de las piezas pasa por la máquina

la secuencia: incógnita del problema, es la secuencia en que cada una de las máquinas ve pasar las piezas.

3.3. Clasificación: problemas estáticos y dinámicos

Es posible distinguir tres tipos de problema del taller: problemas estáticos, semidinámicos y dinámicos.

Las características de los problemas **estáticos** son las siguientes:

- Las piezas son en número finito y determinado y deben realizarse en un taller con un número finito de máquinas
- En el instante de realizar la programación es conocida la ruta de cada pieza, las operaciones que la componen, en qué máquina debe realizarse cada operación y la duración correspondiente.
- Todas las piezas están disponibles en el mismo instante que habitualmente se adopta como instante inicial o instante 0 en tiempo relativo.

En un programa **semidinamico**:

- Las piezas son en número finito y determinado y deben realizarse en un taller con un número finito de máquinas

- En el instante de realizar la programación es conocida la ruta de cada pieza, las operaciones que la componen, en qué máquina debe realizarse cada operación y la duración correspondiente.

- Los instantes de disponibilidad de las piezas y/o máquinas no son todos idénticos, pero sí conocidos en el instante de realizar la programación.

Los problemas semidinamicos entonces se diferencian de los estáticos solo en lo que hace referencia a los instantes de disponibilidad, sin embargo este aspecto acostumbra pesar en los procedimientos de resolución.

La finalidad en la resolución tanto de los problemas estáticos, como semidinamicos es la búsqueda de un programa que optimice un índice de eficiencia establecido, por ejemplo el tiempo de realización de la última pieza.

Al final los problemas **dinámicos** se caracterizan como sigue:

- El horizonte de funcionamiento del taller se considera ilimitado hacia el futuro, o sea el número de máquinas se queda limitado, mientras el número de piezas es ilimitado

- Todas las piezas que deberá tratar el taller en el futuro no estarán definidas en un momento determinado. La definición de las piezas se irá realizando a medida que va trascurriendo el tiempo

- Progresivamente algunas piezas terminan su elaboración en el taller y lo abandonan, siendo substituidas por otras que llegan para ser elaboradas.

Es evidente que los problemas dinámicos requieran un enfoque distinto respecto a los problemas ya analizados. En esta situación no es suficiente realizar un único programa, porque esto resultaría poco eficaz al trascurrir del tiempo, por lo tanto es necesario establecer un conjunto de programas sucesivos a lo largo del tiempo.

3.4. Hipótesis de Conway, Maxwell y Miller para problemas estáticos

En el 1967 Conway, Maxwell y Miller (1967, p. 5-6) establecieron una lista de hipótesis generalmente aceptadas en problemas estáticos del taller mecánico. Se reproducen a continuación:

1. Cada máquina está disponible continuamente desde un instante f ≥ 0, hasta T, con T arbitrariamente grande, no existen intervalos de indisponibilidad por averías o por mantenimiento programado.

2. En las rutas de las piezas no se producen convergencias (montajes) ni divergencias (partición en lotes). Para cada operación existe una sola operación precedente inmediata (exceptuada la primera operación de cada pieza) y una sola operación siguiente inmediata (exceptuada la última operación de cada pieza)

3. Cada operación puede ejecutarse en un solo tipo de máquina del taller

4. Solo hay una máquina de cada tipo en el taller

5. Cuando una operación ha comenzado debe terminarse antes de ejecutar otra en la misma máquina, no se admiten interrupciones

6. No pueden solaparse dos operaciones de la misma pieza (en la misma máquina o en máquinas distintas)

7. Cada máquina puede ejecutar una sola operación a la vez

8. La única restricción activa en el taller es la relativa a las máquinas; no existen restricciones relativas a la disponibilidad de mano de obra utensilios, materiales etc.

3.5. Medidas de eficiencia

Los índices de eficiencia permiten clasificar los programas según sus finalidad (minimizar los retrasos, o los tiempos de espera, etc.) y por otro lado cuantificar el valor del programa estableciendo si un programa es mejor que otro.

Una medida de eficiencia es *regular* si el índice puede formularse en función de los instantes de salida de las piezas:

$$\gamma = f\left(c_1, c_2, ..., c_n\right)$$

Por lo tanto una medida regular solo empeora (aumenta) si uno por lo menos de los valores c_i aumenta y solo mejora (disminuye) si alguno o todos los valores de c_i disminuyen.

Citamos como medidas de eficiencia regulares:

- F_{max}, el mayor de los tiempos de permanencia de las piezas
- F_{med}, el valor medio de los tiempos de permanencia de las piezas
- C_{max}, el instante de terminación de la última pieza
- C_{med}, el valor medio de los instantes de terminación de la piezas
- T_{max}, el mayor de los retrasos de las piezas
- T_{med}, el valor medio de los retrasos de las piezas

3.6. Clasificación del flujo de piezas en el taller

El los problemas de programación de operaciones es posible considerar distintos flujos de piezas en las máquinas. Los más comunes se detallan a continuación

Flujo regular (flow-shop): Todas las piezas tienen la misma ruta. Las *m* máquinas se pueden numerar *1, 2, ... , m* de tal forma que una pieza cualquiera será procesada por todas las máquinas en orden creciente. Ninguna pieza sufre más de una operación en cada máquina.

Flujo permutacional (permutacional flow-shop): Es un caso particular del flujo regular en el cual la secuencia de piezas es la misma en todas las máquinas. Por lo tanto el programa queda definido tan solo por una permutación de las piezas, de aquí el nombre.

Flujo aleatorio: Indica el caso en que la ruta de las piezas es casual o sea no responde a un esquema definido.

Flujo general (job-shop): Existen piezas en las que dos operaciones sucesivas se ejecutan la primera en la máquina *j* y la segunda en la máquina *j'* y otras piezas en que dos operaciones se ejecutan la primera en *j'* y la segunda en *j*. Además una pieza puede sufrir varias operaciones en la misma máquina.

3.7. Nomenclatura

3.7.1. Nomenclatura de Conway, Maxwell y Miller

Los problemas de secuenciación pueden describirse mediante una notación simple compuesta por cuatro símbolos, propuesta por Conway, Maxwell y Miller (1967, p. 6-8):

A / B / C / D

A , Se refiere a las piezas. En los problemas estáticos y semidinamicos es el número de las piezas (se usa el símbolo *n* si el número es arbitrario). En los problemas dinámicos puede indicar una ley de probabilidad de la llegada de las piezas al taller

B , Es el símbolo relativo a las máquina, indica su número. Para indicar el número arbitrario se utiliza el símbolo *m*.

C , Indica el tipo de flujo de las piezas en el taller:

- F: flujo regular
- P: flujo permutacional
- R: flujo aleatorio
- G: flujo general

D , Corresponde a la medida de eficiencia elegida para evaluar la calidad de los programas F_{max}, c_{max}, T_{med}, etc.

Este tipo de nomenclatura, aunque rápida y sencilla, a veces no es exhaustiva, dado el gran número de variantes del problema del taller mecánico estudiadas en la literatura.

3.7.2. Nomenclatura de Lawer

Con la nomenclatura de Lawer et al. (1982, p. 35-74) un problema del taller mecánico se describe mediante un grupo de tres campos:

α | β | γ

El campo α se refiere al entorno de las máquinas y contiene una sola inscripción. El campo β detalla características del proceso, y puede contener una, ninguna o varias inscripciones. Al final γ representa la medida de eficiencia. A continuación se indican unas posibilidades para los dos primeros símbolos.

α

- 1, una sola máquina
- P, máquinas idénticas en paralelo
- Q, máquinas en paralelo con velocidades diferentes
- F, Flow-shop, máquinas en serie
- HF, Flow-shop híbrido, generalización del caso anterior, existen una serie de etapas, cada una formada por unas máquinas en paralelo
- FF, Flow shop flexible, representa un flowshop en que no todas las piezas tienen que ser procesadas en todas las etapas.

β

- r_i, fechas de disponibilidad de las piezas diferente de 0
- $s_{h,i}$ tiempos de preparación dependientes de la secuencia

- bkdw indisponibilidad de máquina (breakdowns)

3.7.3. Nomenclatura de Vignier

En 1999, Vignier et al. (1999, p. 117-183) proponen una notación unificada que permita establecer la tipología de los problemas en estudio. Los autores proponen seguir la notación de Lawler et al. (1982), pero desglosando el campo α en cuatro parámetros según la siguiente forma:

$$\alpha = \alpha_1\alpha_2\left(\alpha_3\alpha_4^{(1)}, \alpha_3\alpha_4^{(2)}, ..., \alpha_3\alpha_4^{(\alpha_2)}\right)$$

Donde:

- α_1 : es el parámetro α de la nomenclatura de Lawer
- α_2 : indica el número de etapas en un flow-shop
- α_3 y α_4 se repiten para cada una de las α_2 etapas
    - $\alpha_3^{(k)}$ indica el tipo de máquinas en la etapa *k*. Valdrá *0* si hay únicamente una máquina en la etapa, *P* si hay máquinas idénticas en paralelo, *Q* si las máquinas son proporcionales o *R* si hay máquinas en paralelo no relacionadas
    - $\alpha_4^{(k)}$ indica el número de máquinas en la etapa *k*

La notación de Vignier es la más completa entre las analizadas y permite describir sistemas complejos y realísticos.

3.8. Clasificación: tipo de programa

El número de programas o soluciones factibles de un problema de programación de operaciones puede ser muy elevado, por lo tanto resulta útil filtrar estos programas,

eliminando los de bajo interés. Esta es la finalidad de la clasificacion en problemas semiactivos, activos y activos sin retraso.

3.8.1. Programas semiactivos

Imaginamos un programa cualquiera en el diagrama de Gantt, los desplazamientos de las operaciones hacia la izquierda tanto como sea posible, sin alterar el orden de las operaciones en ninguna máquina, cogen el nombre de *desplazamientos limitados a la izquierda*. Los desplazamientos quedan limitados porque se llega a un punto en que o la máquina, o la pieza no están libre. Un Programa en el cual no son posibles ulteriores desplazamientos limitados a la izquierda se llama *programa semiactivo*. Es interesante notar como hay distintos programas que lleven al mismo programa semiactivo, o sea los programas semiactivos son un subconjunto de las soluciones del problema.

3.8.2. Programas activos

Ulteriores retrasos pueden ser recuperados aplicando desplazamientos hacia la izquierda en que la secuencia de las piezas en alguna máquina pueda cambiar. Podemos definir los *desplazamientos generales a la izquierda* como el conjunto de estos desplazamientos y de los desplazamientos limitados, y entonces definir *programa activo* un programa en que no sean posibles ulteriores desplazamientos generales a la izquierda. En este caso hay distintos programas semiactivos que llevan al mismo programa activo, o sea los programas semiactivos son un subconjunto de los programas semiactivos.

3.8.3. Programas activos sin retrasos

Se define programa *activo sin retraso* un programa en que las máquinas disponibles no permanecen inactivas manteniendo en cola operaciones, en espera de poder realizar una operación sobre una pieza con mayor prioridad.

3.8.4. Programa óptimo

Se denomina programa óptimo un programa que minimice la medida de eficiencia. Si la medida de eficiencia considerara es regular, la búsqueda de la solución óptima se efectúa en el conjunto de los programas activos. De hecho existirá siempre un programa óptimo activo de calidad igual o superior a un programa cualquiera, porque la trasformación a programa activo no puede empeorar la medida de eficiencia.

4. Configuraciones Productivas

En este capítulo se detallan las configuraciones productivas más comunes, empezando del caso más sencillo de una sola máquina y una etapa, hasta llegar a las configuraciones más completas, como la del flowshop híbrido, objeto del proyecto.

La finalidad es dar un cuadro completo de los problemas de programación de operaciones.

4.2. Una máquina

La configuración productiva más sencilla que se pueda encontrar es constituida por una única máquina en la cual se realiza una sóla operación por cada pieza. Todas las piezas entonces serán procesadas por esta máquina, o sea la subfuncion de carga se realiza automáticamente, y solo quedan la secuenciación y la temporización. Dado un ejemplar con *n* trabajos a realizar, las soluciones posibles serán *n!*.

Figura 4.1 – Configuraccion poductiva con una sola máquina. Fuente: propia.

Según la nomenclatura de Conway et al. (1967) esta configuración productiva se definiría como:

n / 1 / / D

A continuación se detallan algunas reglas de secuenciación utilizadas para realizar la secuenciación en ejemplares con una sola máquina

SPT (Shortest Processing Time): Secuencia conforme a la duración. Corresponde a la regla heurística *"ejecutar primero la operación más corta"* y permite minimizar el índice

de eficiencia F_{med} y el stock medio. Las piezas se procesan en orden creciente de tiempos de proceso p_i:

$$p_1 \leq p_2 \leq ... \leq p_n$$

EDD (Earliest Due Date): Secuencia conforme a la fecha de vencimiento. Corresponde a la regla heurística *"ejecutar primero lo más urgente"* y permite minimizar T_{max} . Las piezas se procesan en orden creciente de fecha de vencimiento d_i.

$$d_1 \leq d_2 \leq ... \leq d_n$$

ECT (Earliest Completion Time): Secuencia conforme al tiempo de terminación. Corresponde a la regla heurística *"ejecutar primero lo que acaba antes"*. Corresponde al SPT si no hay tiempos de preparación. Las piezas se procesan en orden creciente de fecha de terminación C_i

$$C_1 \leq C_2 \leq ... \leq C_n$$

4.3. Máquinas en paralelo

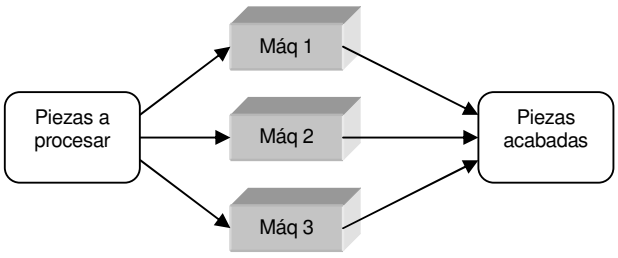


Figura 4.2 – Configuacion productiva caracterizada por máquinas en paralelo. Fuente: propia.

Una generalización de la configuración productiva precedente es la que prevé más máquinas en paralelo. Según la nomenclatura de Lawer et al. (1982) este caso se indica de la siguiente forma (se considera el caso de máquinas idénticas):

Pm / β / γ

En este caso el proceso productivo se queda formado por una sola operación, pero existen varios centros de trabajo alternativos que la pueden realizar. En referencia a la eficiencia relativa de las máquinas podemos detallar tres casos distintos:

- Máquinas idénticas: Todas las máquinas tardan el mismo tiempo en realizar la misma tarea

- Máquinas uniformes: Cada máquina lleva asociado un *factor de velocidad*, por lo tanto puede realizar las tareas con una velocidad diferente respecto a las otras. Existirán máquinas mas rápidas, que realizaran las tareas en menos tiempo y máquinas más lentas.

- Máquinas diferentes: el tiempo de procesado puede ser completamente arbitrario; una misma máquina puede ser rápida en una tarea pero lenta en otra en comparación con otras máquinas.

En problemas con máquinas en paralelo es necesario desarrollar también la subfuncion de carga, diferentemente de lo que ocurre en configuraciones compuestas por una sola máquina por etapa de trabajo. Un desarrollo interrelacionado de carga y secuenciación permite lograr generalmente programas mejores, pero por las dificultades que ello entraña, en algunas ocasiones se resuelven independientemente las dos subfunciones.

4.4. Flowshop

Otra configuración productiva fundamental es el flowshop, según la nomenclatura de Conway et al. (1967):

n / m / F / D

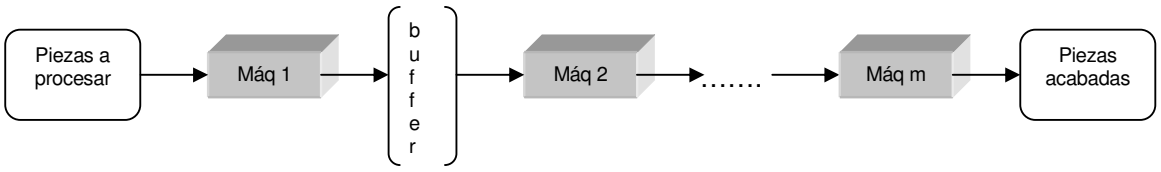


Figura 4.3 - Representación grafica de un flowshop. Fuente: propia

En el campo de la programación de operaciones el sistema flow-shop es una de la problemáticas más estudiada. En este tipo de problemas hay que procesar un conjunto de *n* piezas en *m* etapas de trabajo dispuestas en serie, en cada etapa hay una sola máquina. Todas las piezas siguen la misma ruta de producción o sea son procesadas por la máquina 1, después por la máquina 2 y así hasta la máquina *m.* No es posible para ninguna pieza, seguir una ruta de producción distinta de esta, ni volver dos o más veces a la misma máquina. Por lo tanto cada pieza será procesada en cada máquina una sola vez y cada operación tendrá una duración establecida llamada tiempo de proceso indicada con p_{ij} donde el pedice *i* indica el numero de la pieza y el pedice *j* indica la máquina en que se procesa.

p_{ij}	Máquina 1	Máquina 2	…..	Máquina m
Pieza 1	p_{11}	p_{12}	…	p_{1n}
Pieza 2	p_{21}	p_{22}	…	p_{2n}
. . .	. . .			
Pieza n	p_{n1}			p_{nm}

Figura 4.4 – Tiempos de preparacion. Fuente: propia.

5. Planteamiento del problema

En esta sección se describe en detalle el problema del flow shop híbrido flexible con tiempos de preparación dependientes de la secuencia, su dificultad de resolución y su importancia en la programación de operaciones.

5.1. Una variante realística del flowshop

Los estudios acerca de los flow-shop representan las bases de la programación de operaciones, pero últimamente han sido criticados, debido a la excesiva simplificación del proceso de producción. De hecho casi nunca una línea de producción está formada por una máquina de cada tipo, en cambio muchas veces tendremos un número distinto de máquinas en paralelo en cada etapa. Por ejemplo, es común poner más máquinas iguales en las etapas *cuello de botella* para limitar o eliminar el fenómeno. De la misma manera está claro que no todas las piezas siguen la misma ruta de producción, ni requieren el mismo número de operaciones, sino que habrá operaciones necesarias para algunas piezas, y no para otras. Otro problema del flowshop tradicional es la falta de consideración de los tiempos de preparación o *Setup*, o sea todas aquellas operaciones no-productivas que se deben realizar sobre una máquina antes de empezar el proceso de la pieza (limpieza de la pieza, cambio de molde, etc.). Estas operaciones a veces son muy importantes, llegando incluso a tardar tanto como el tiempo de trabajo, por lo tanto si éstos no son considerados se puede generar errores importantes.

Por lo tanto, un sistema flowshop puro no es suficiente para simular la mayoría de las líneas de producción. Por esto, en nuestro estudio, se han introducido unas variantes, con la finalidad de obtener un sistema más real. Estas variantes se detallan a continuación, y nos permiten llegar a la definición exacta del problema tratado de *Flowshop Hibrido Flexible con tiempos de preparación dependientes de la secuencia*.

5.1.1. Máquinas idénticas en paralelo

La primera variante introducida es la posibilidad de tener más máquinas iguales en cada etapa, que realicen la misma tarea. El número de máquinas puede ser distinto en cada etapa, y es un dato del problema. Es fácil entender como ésto generaliza mucho más el esquema, y incluye muchos más casos reales. Un problema de flow-shop, con máquinas iguales en paralelo es conocido como *Flowshop Hibrido.*

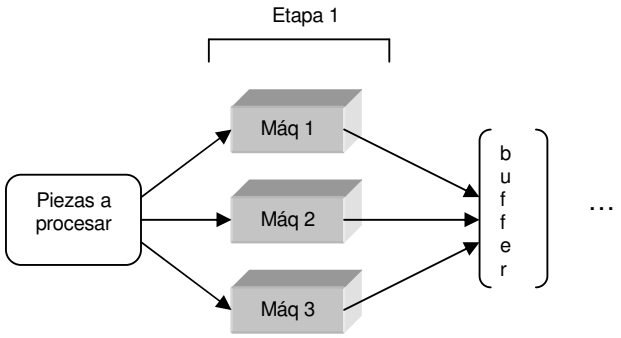


Figura 5.1 – Ejemplo de máquinas en paralelo en una etapa. Fuente: propia.

5.1.2. Stage skipping

La segunda variante introduce la posibilidad de que las piezas puedan saltar algunas etapas de la línea de producción, lo que la convierte en una línea de producción flexible.

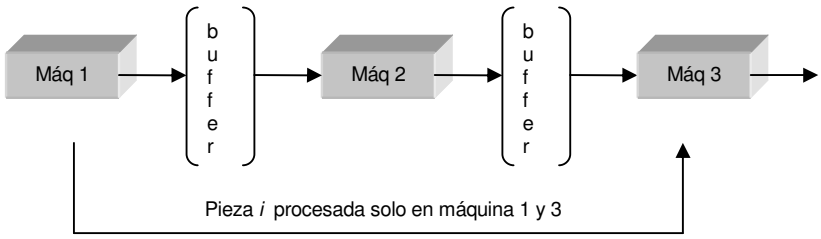


Figura 5.2 – Ejemplo de stage-skipping. Fuente: propia.

A menudo existen trabajos que solo se hacen sobre un tipo de piezas, o piezas que requieren operaciones especiales, por ejemplo de limpieza o tratamientos de la superficie, el *stage skipping* permite considerar este hecho. Un problema de flowshop con posibilidad de stage skipping es conocido como *Flowshop Flexible.*

5.1.3. Tiempos de preparación dependientes de la secuencia

La tercera variante tiene como objetivo considerar los tiempos de preparación necesarios para preparar las máquinas para el trabajo siguiente. En la mayoría de los entornos industriales, la duración de estas operaciones depende tanto de la pieza que se acaba de procesar como de la pieza que se va a procesar, es decir, depende de la secuencia de producción establecida. Por esta razón, en este estudio se han considerado tiempos de preparación dependientes de la secuencia.

5.2. Ejemplos industriales

En esta sección se describen dos ejemplos de sistemas productivos que se pueden modelizar como sistemas flowshop híbrido. Una explicación más detallada de los mismos se puede encontrar en Ribas (2007). La finalidad es aclarar los conceptos que se han introducido y dar una idea de las posibles aplicaciones del proyecto.

5.2.1. Fabricación de etiquetas de papel

El primer ejemplo lo podemos encontrar en el sistema productivo de una empresa que produce etiquetas adhesivas

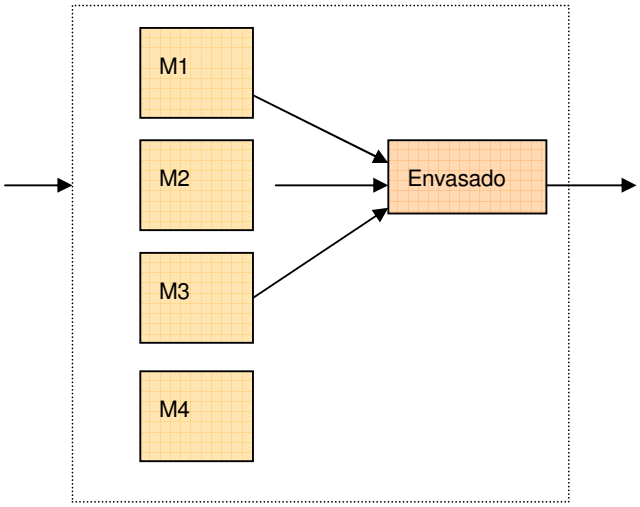


Figura 5.4 – Sistema productivo para fabricar etiquetas en hojas DIN 4. Fuente: Ribas (2007)

El sistema está formado por 2 etapas. En la primera etapa hay 4 máquinas idénticas que pueden fabricar etiquetas de diferentes medidas en función del troquel que se coloque, en hojas DIN4. En la segunda etapa hay una única máquina de envasado (Fig. 5.4). Este sistema se puede esquematizar como un flowshop híbrido de dos etapas y tiempos de preparación dependientes de la secuencia.

5.2.2. Fabricación de cacao en polvo

Otro ejemplo lo encontramos en una empresa del sector de la alimentación que produce cacao en polvo. El proceso productivo empieza mezclando el cacao con harina y azucar, se efectúa entonces a una fase de tamizado y enfriado. El cacao en polvo obtenido se almacena en unos silos intermedios que se comunican con la zona de envasado. La zona de envasado está formada por diferentes líneas. Una para sobres, una para botes estrechos, una para bolsas, una para botes grandes y una línea rápida (formado estándar). Es posible también saltar la tapa de envasado, en el caso en que se venda el producto a marcas blancas (Fig. 5.5). Este sistema productivo se puede modelizar como un flowshop híbrido flexible, formado por 3 etapas con 2, 2 y 5 máquinas respectivamente y tiempos de preparación independientes de la secuencia.

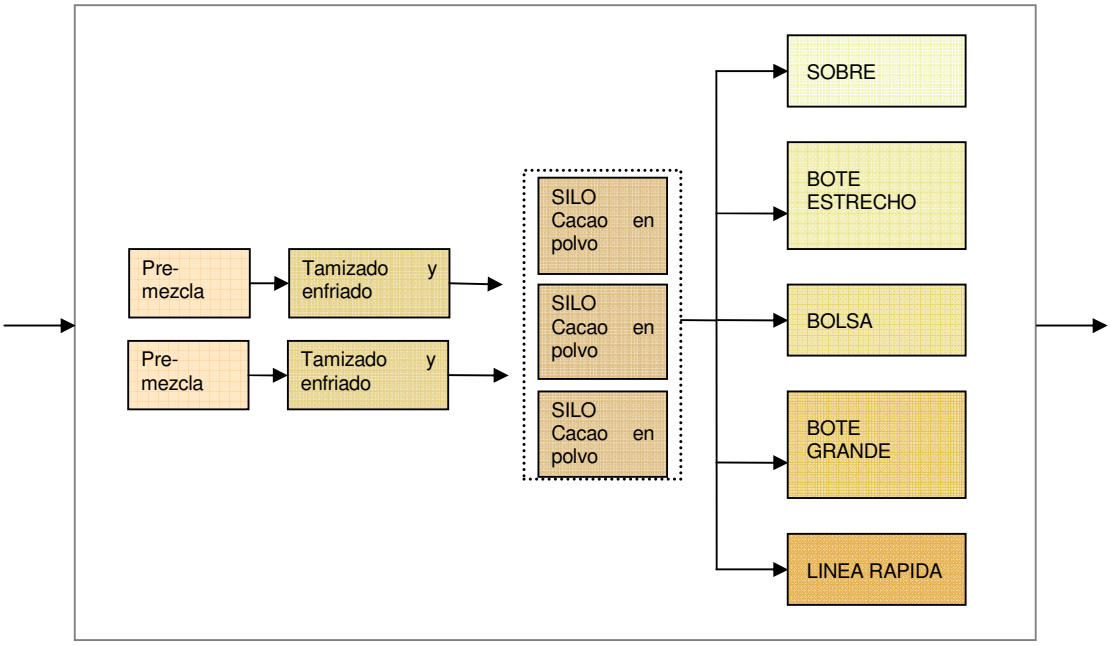


Figura 5.5 – Sistema productivo para fabricar Cacao en polvo. Fuente: Ribas (2007)

5.3. El índice de eficiencia: el retraso medio

Ilustrado el problema que vamos a tratar, nos dedicamos ahora a la descripción del índice de eficiencia elegido, a su definición matemática, sus propiedades y su significado e importancia para una firma.

Nuestro índice de eficiencia será el retraso medio, o sea la suma de los retrasos de cada pieza, divididos por el número total de piezas. Esta definición corresponde a la siguiente expresión matemática.

$$T_{med} = \frac{1}{n} \cdot \sum_{i=1}^{n} T_i \qquad \text{(Ec. 5.1)}$$

Para mayor precisión, vale la pena recordar la definición de retraso de una pieza, definido como el máximo entre la diferencia entre la fecha de entrega de una pieza C_i y su fecha de vencimiento d_i, y cero. A continuación su definición matemática.

$$T_i = \max\left[(C_i - d_i), 0\right] \qquad \text{(Ec. 5.2)}$$

5.4. Definición del problema

El proyecto se centra en la programación de operaciones en un flowshop hibrido flexible con tiempos de preparación dependientes de la secuencia, con el objetivo de minimizar el retraso medio. Utilizando la notación de Vignier et al.(1999) el problema se define:

$$HFFm\left(P\alpha_4^1, ..., P\alpha_4^m\right) \ / \ S_{ijk} \ / \ T_{med}$$

Donde

- *HFF* , indica la configuración flowshop hibrido flexible
- *m* , es el número de etapas
- *P* , indica el hecho que las máquinas en paralelo en cada etapa son idénticas

- α_4 , es el número de máquinas en cada etapa. Los ejemplares considerados se dividen en dos grupos:

  - Simétricos: hay el mismo número de máquinas en cada etapa

  - Asimétricos: el número de máquinas en cada etapa es distinto

- S_{ijk}, indica que se consideran los tiempos de preparación dependientes de la secuencia, distintos en cada etapa de trabajo.

$$S_{ijk} \quad \text{donde} \quad \begin{cases} i & \text{es el numero de pieza que se va a procesar} \\ j & \text{es el numero de etapa} \\ k & \text{es el numero de la ultima pieza que se ha procesado} \end{cases}$$

- T_{med}, es la medida de eficiencia, el retraso medio.

El problema considerado es estático y respecta todas las hipótesis de Conway, Maxwell y Miller, menos la 4.

De momento no se han considerado penalizar la fabricación anticipada de piezas, es decir, que la fabricación concluya antes del momento de la entrega. Esto puede suponer unos costes de posesión, como la creación y mantenimiento de la capacidad e almacenaje, la entrada y salida de artículos en el stock, costes de seguridad etc. Esto podría ser uno de los desarrollos futuros del programa.

6. Métodos de resolución

El problema al cual nos enfrentamos es de tipo combinatorio, como muchos que se presentan en la disciplina de la organización industrial.

En estos problemas existe un conjunto finito de soluciones, por lo tanto, en teoría, sería suficiente evaluar cada una de ellas con respecto al objetivo marcado para encontrar la solución óptima. Sin embargo, en la realidad este procedimiento no es viable, ya que la complejidad de la mayoría de los problemas de las empresas requeriría tiempos de elaboración excesivamente largos.

Hacemos un ejemplo aclaratorio: consideramos un caso muy sencillo de programación de operaciones el que hay que procesar *20* piezas en una sola máquina (Conway et al.: 20/1/-/D). Como ya explicado, se trata de un caso de simple secuenciación, porque no hay que realizar la carga. Las soluciones posibles entonces serán las permutaciones de las piezas $20!=2,43*10^{18}$. Suponiendo que se dispone de un ordenador muy rápido, capaz de evaluar una combinación cada nanosegundo tardaríamos más que *77* años en analizar todas las combinaciones. Se entiende entonces la importancia del desarrollo de métodos de optimización que no requieran una evaluación exhaustiva de todos los candidatos del campo de soluciones.

En este capítulo se revisan los métodos más utilizados en la resolución de los problemas de programación, y se identifican las ventajas y desventajas de cada uno. Puede resultar útil hacer una clasificación de las metodologías empleadas. La primera división posible es entre métodos exactos y heurísticos, después entre estos últimos es posible destacar otra subcategoría: los metaheurísticos.

6.2. Los métodos exactos

Los métodos exactos son los que permiten obtener la solución óptima de un problema. Una solución es óptima, si es la mejor de todas las soluciones posibles al problema. En nuestro

caso se resuelve un problema de minimización, ya que la finalidad es minimizar los retrasos, por lo tanto la solución óptima se define así:

$$f(x') = f(x_{opt}) \Leftrightarrow f(x') < f(x) \qquad \forall x \in A \qquad \text{(Ec. 6.1)}$$

Donde *A* representa el espacio de las soluciones, es decir todas las soluciones posibles.

Se podría pensar, por lo tanto, que los métodos exactos representan la mejor opción para la resolución de problemas flowshop, pero en la realidad tanto los métodos B&B como la programación matemática sólo son aplicables a ejemplares de tamaño muy reducido ya que el tiempo de cómputo requerido aumenta exponencialmente con la dimensión del ejemplar.

6.2.1. Branch and Bound

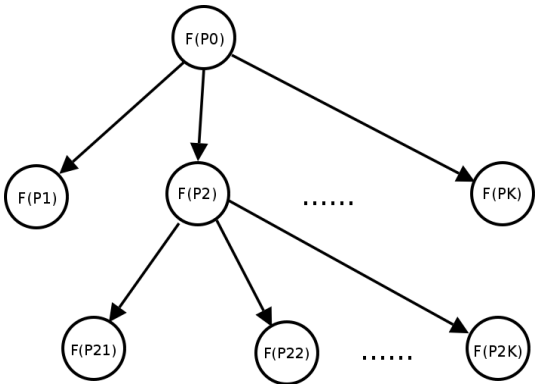


Figura 6.1 – Ejemplo de resolucion con método B&B.
Fuente: Wikipedia

La mayoría de los métodos exactos están basados en procedimientos *branch and bound* (B&B). El método B&B ha sido propuesto por *Land y Doig* en el 1960, y forma parte de los algoritmos de enumeración implícita, ya que "enumera" todas las soluciones posibles hasta llegar a la óptima, aunque por el camino se van descartando soluciones, demostrando *a priori* su no-optimalidad. En este tipo de algoritmo el espacio de las soluciones se divide en unas ramas (fase de ramificación), cada rama se evalúa, y se sigue investigando sólo sobre las ramas susceptibles de llegar a soluciones óptimas (fase de poda).

6.2.2. Programación entera y mixta

Casi la totalidad de los problemas combinatorios admiten una formulación como programas lineales enteros o binarios, y especialmente como mixtos. El inconveniente de este método reside en la gran cantidad de variables que surgen en la mayoría de los problemas.

6.3. Los métodos heurísticos

Debido a la imposibilidad de utilizar algoritmos exactos para la resolución de problemas combinatorios comunes, la investigación se ha dirigido hacia el desarrollo de metodologías heurísticas y metaheurísticas, que sacrifican la optimalidad de la solución a cambio de tiempos de cálculo aceptables.

Podemos definir un algoritmo heurístico, o sencillamente heurística, como [Zanakis y Evan, 1981]:

> *"Un procedimiento sencillo, solvente basado en el sentido común, que proporciona una solución de buen nivel (no necesariamente optima) a problemas difíciles, de forma fácil y rápida."*

Los algoritmos heurísticos suelen ser métodos de resolución de problemas muy útiles cuando se cumplen una o más de las siguientes circunstancias:

- Cuando no puede aplicarse un método exacto para la resolución del problema, por no existir o porque requiera demasiados recursos (tiempo de cálculo, memoria, presupuesto, etc.)

- Cuando el problema o la situación planteada no requieren una solución exacta o precisa, sino sólo aproximada

- Cuando los datos disponibles son poco fiables

- Como un paso intermedio para la aplicación de otros algoritmos. Muchas veces la solución aportada por un método heurístico es tomada como punto de partida de otros procedimientos (especialmente metaheurísticos).

- Cuando las limitaciones de tiempo, espacio (para el almacenaje de datos), etc. conducen a la utilización de métodos de respuesta rápida aunque sea a costa de la precisión.

Existen varios algoritmos heurísticos, a continuación se detalla una de las posibles clasificaciones.

6.3.1. Métodos constructivos

Son aquellos métodos que añaden componentes individuales a una solución parcial hasta que se obtiene una solución factible. El más popular de estos métodos lo constituye un algoritmo goloso o voraz, "greedy", el cual construye la solución buscando el máximo beneficio en cada paso.

6.3.2. Métodos de descomposición

Estos métodos consisten en dividir el problema en subproblemas más pequeños, siendo la salida de uno la entrada de otro, de forma que al resolver ambos subproblemas obtengamos una solución para el problema global.

6.3.3. Métodos de reducción

Los métodos de reducción identifican alguna característica que deba poseer la solución óptima y de este modo simplifican el problema. Por ejemplo la detección de alguna variable con ciertos valores o correlación.

6.3.4. Métodos de manipulación del modelo

Modifican las estructuras del modelo con el fin de hacerlo más sencillo de resolver, deduciendo, a partir de la solución del problema modificado, la solución del problema

original. Como por ejemplo, se puede reducir el espacio de soluciones o eliminando restricciones del problema.

6.3.5. Métodos de búsqueda por entornos

Parten de una solución factible inicial (probablemente obtenida de otra heurística) y mediante alteraciones de la solución, van iterando a otras factibles de su entorno, almacenado la mejor solución encontrada hasta que se cumpla un determinado criterio de parada. Pertenecen a esta categoría dos de las heurísticas más utilizadas: los algoritmos de descenso exhaustivo y no exhaustivo.

En la heurística de descenso exhaustivo (AED) a partir de la solución en curso se generan y evalúan todos los vecinos, si el mejor de ellos es mejor que la solución en curso se toma como nueva solución en curso y se reitera el procedimiento; en caso contrario si el mejor vecino es peor o igual a la solución en curso el procedimiento se da por terminado.

En cambio en el método de descenso no-exhaustivo (ANED) a partir de la solución en curso se generan y evalúan en cierto orden sus vecinos, si uno de ellos es mejor que la solución en curso se toma como nueva solución en curso (sin terminar la generación de los vecinos de la solución primitiva) y se prosigue aplicando el procedimiento a los vecinos de la nueva solución en curso; cuando se han generado todos los vecinos de una determinada solución en curso sin que ninguno sea mejor el procedimiento se da por terminado.

6.4. Los métodos metaheuristicos

Las metaheuristicas son una tipología de heurísticas apta para solucionar una clase más general de problemas combinatorios, combinando distintos procedimientos (generalmente heurísticos) de forma eficiente.

Usando la definición de Osman y Kelly (1996):

"Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son ni efectivos ni eficientes."

A continuación se exponen los procedimientos metaheurísticos mas empleados.

6.4.1. Recocido Simulado (SA, *Simulated Annealing*)

El nombre de esta metodología procede de la analogía con el tratamiento térmico de recocido que consiste en calentar y luego enfriar controladamente un material para eliminar sus defectos.

SA parte de una solución inicial determinada por una heurística, y mediante la exploración de su entorno, trata de encontrar la solución óptima. En cada iteración, el procedimiento considera algunos vecinos del estado actual *s*, y probabilísticamente decide entre cambiar el sistema al estado *s'* o quedarse en el estado *s*. Las probabilidades se escogen para que el sistema tienda finalmente a soluciones mejores. Típicamente este paso se repite hasta que se alcanza un estado suficientemente bueno para la aplicación o hasta que se cumpla cierto tiempo computacional dado.

La probabilidad de hacer la transición al nuevo estado *s'* es una función *P(δ E, T)* de la diferencia de energía *δE=E(s')-E(s)* entre los dos estados, y de la variable *T*, llamada temperatura. Generalmente sigue la distribución de Boltzmann.

$$P = e^{-\frac{\partial E}{kT}} \qquad \text{(Ec. 6.2)}$$

Una cualidad importante del método SA es que la probabilidad de transición P es siempre distinta de cero, aún cuando *δE* sea positivo, es decir, el sistema puede pasar a un estado de mayor energía (peor solución) que el estado actual. Esta cualidad impide que el sistema se quede atrapado en un óptimo local. Cuando la temperatura tiende al mínimo, la probabilidad tiende a cero asintóticamente. Así, cada vez el algoritmo acepta menos movimientos que aumenten la energía. Si *δE* es negativo, es decir, la transición disminuye la energía, el movimiento es aceptado con probabilidad *P=1*.

6.4.2. Búsqueda Tabú (TS, *Tabú Search*)

La búsqueda tabú, técnica propuesta en la forma actual por F. Glover (1989, p. 190-206), es una evolución del clásico método de descenso (AED o ANED). El método clásico consiste en empezar de una solución inicial y analizar las soluciones vecinas buscando una solución en que la función objetivo tenga un valor inferior respecto a la solución actual. Si en el vecindario no hay ninguna solución mejor, el procedimiento se acaba, pero la solución encontrada es un mínimo local que puede estar muy lejos de la solución optima. El TS en cambio permite moverse a soluciones vecinas peores, para que la solución no se quede entrapada en un mínimo local, y marca las últimas soluciones analizadas como prohibidas o *tabú* para evitar de volver a caer en el mismo mínimo local en la búsqueda de vecinos siguiente. La característica fundamental de la TS entonces es el uso de la memoria, para tener un rastro del recorrido de la búsqueda.

6.4.3. Algoritmo Genético (GA, Genetic Algorithm)

Los GA, introducidos por J. Holland (1975), son algoritmos basados en las ideas de la selección natural, específicamente aquellos que siguen el principio de la supervivencia en función de la adaptabilidad.

En lugar de una solución en curso se arrastran varias (población). Se seleccionan los padres, se cruzan y/o mutan y se obtienen los hijos regenerando la población inicial. Más en detalle, la información de un problema se codifica en una serie de cadenas (cromosomas) que forman una población. Por medio del intercambio de información entre ellas (cruce) y/o la variación aleatoria de esta información (mutación), se evalúa la solución definida por las cadenas, relacionada con el problema en cuestión, midiendo de esta forma su adaptabilidad al entorno (evaluación de la función objetivo). Los individuos se eligen en función de su mejor adaptación (selección). Se genera una nueva población a partir de estos cromosomas de élite, sobre la que se vuelve a aplicar este proceso iterativo una y otra vez. De este modo, la calidad media de los individuos aumenta progresivamente.

6.4.4. GRASP (Greedy Randomized Adaptive Search Procedure)

El GRASP es una metaheurística de tipo iterativo, desarrollada originalmente por Feo y Resende (1989). En este algoritmo se repiten iterativamente dos fases. En la primera se construye una lista de soluciones factibles utilizando la heurística Greedy aleatorizada. El método Greedy es de tipo constructivo, es decir añade a la solución un elemento a la vez, y se caracteriza por elegir de forma miope la decisión que lleva el máximo beneficio en cada paso. En el Greedy aleatorizado en cambio se elige la decisión en cada paso de forma casual entre una lista de los mejores candidatos. En la segunda fase del algoritmo, las soluciones factibles generadas pasan por un proceso de mejora local, y la mejor solución obtenida se queda guardada como resultado del procedimiento.

6.4.5. Búsqueda Local Iterativa (ILS, Iterated Local Search)

La búsqueda local iterativa es la metaheuristica elegida para la resolución de nuestro problema. Los métodos ILS se detallaran en el capítulo siguiente, pero definimos a continuación el procedimiento de base. Se empieza con una solución inicial, que puede ser generada de forma aleatoria o por medio de una heurística. Entonces se realizan iterativamente dos fases. En la primera se efectúa un análisis de los vecinos, con la finalidad de encontrar un mínimo local; en la segunda se aplica una perturbación a la solución en curso, para salir del mínimo local y explorar otras regiones del espacio de las soluciones a la búsqueda de soluciones mejores.

7. Procedimiento de resolución propuesto

7.1. Introducción a los métodos ILS

El método resolutivo que ha sido implementado es una metaheurística del tipo Iterated Local Search (ILS). El ILS es una metaheurística que aplica iterativamente un procedimiento de búsqueda local que es capaz de generar una sucesión de soluciones que converge a un resultado mucho mejor de lo que se conseguiría aplicando repetidamente el mismo método de búsqueda local de forma aleatoria.

En la búsqueda local iterativa se empieza con una solución inicial (que puede ser de mayor o menor calidad) y se realiza una búsqueda local del vecindario, hasta encontrar un mínimo local. Empieza entonces la parte iterativa del algoritmo durante la cual se aplican tres operadores: la perturbación en que se modifica la solución en curso para alejarla del mínimo local, la búsqueda local que tiene la finalidad de analizar el vecindario y encontrar un nuevo mínimo, y el criterio de aceptación que decide qué solución será perturbada en la iteración siguiente. La idea básica, por lo tanto, es aplicar la búsqueda local a una modificación de la solución escogida por el criterio de aceptación. Estudios de Hoos y Stützle (2005) afirman que el ILS es una de las metodologías más sencilla y eficaz para evitar el estancamiento de la solución alrededor de óptimos locales. A continuación se describe el pseudocódigo elemental del procedimiento.

```
s = Solución_Inicial
s* = s
do
      s' = Búsqueda_Local(s)
      s* = Criterio_Aceptación(s*,s')
      s = Perturbación(s*)
while (condición de salida)
```

Esta idea tiene una historia larga y ha sido usada muchas veces en el pasado, pero los primeros que la aplicaron en la forma moderna fueron Lin y Kernigham (1973, p. 498-516), para la resolución del problema del viajante de comercio (TSP, *Traveling Salesman Problem*).

Desde entonces este método ha sido utilizado a menudo en distintos problemas de programación de operaciones, produciendo resultados *estado del arte*, como por ejemplo en problemas de flowshop regulares, para la minimización del C_{Max} [Ruiz y Maroto, 2005, p.479-494].

El ILS es una metodología modular, sus prestaciones dependen tanto de la calidad, como de la interacción entre sus cuatro módulos:

- Generación de una solución inicial
- Búsqueda local
- Criterio de aceptabilidad
- Perturbación de la solución

El primer paso del algoritmo es la generación de la solución inicial que representa el punto de partida de la búsqueda. Esta se puede generar aleatoriamente o por medio de una heurística. Generalmente en el campo de la programación de operaciones se prefiere la segunda opción ya que la mayoría de las veces soluciones iniciales aleatorias se traducen en métodos caracterizados por convergencia lenta. Un estudio de Stützle (2003) confirma esta teoría, demostrando pero, por otro lado, que para simulaciones largas la elección de la solución inicial no es crítica (Fig 7.1).

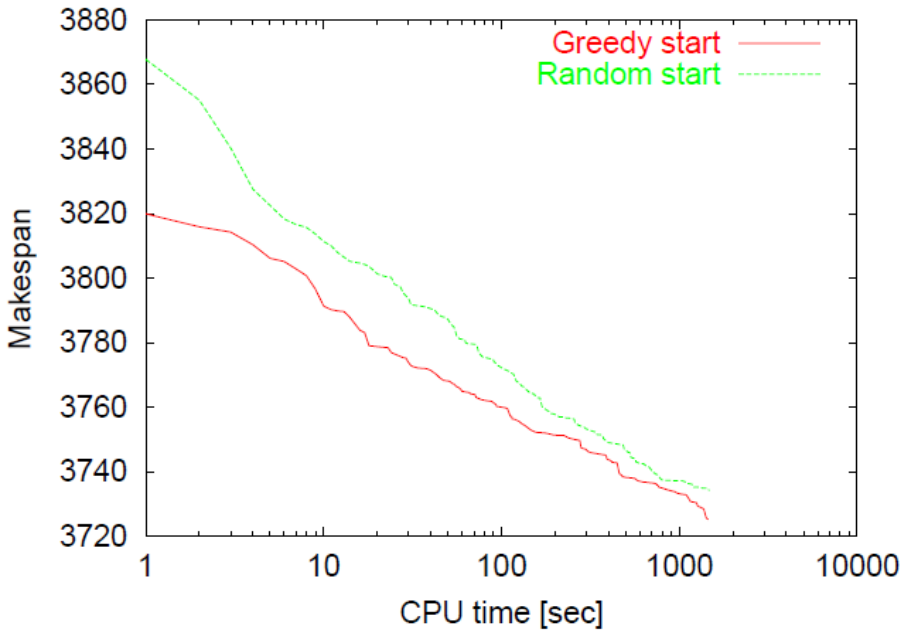


Figura 7.1 – Efecto de la calidad de la solucion inicial con referencia a la convergencia y a la solucion asintotica. Fuente: [Stützle, 2003]

El segundo módulo es la búsqueda local lo cual permite encontrar el mínimo local de la región del espacio de las soluciones que se está analizando. Existen distintos métodos de búsqueda local, dependiendo del número de las piezas desplazadas y del tipo de desplazamiento realizado, y su eficacia no es absoluta sino que está relacionada con el tipo de problema tratado. Los parámetros importantes de esta fase son su eficacia y sobretodo su rapidez, porque es la que más veces se repite en el programa.

El criterio de aceptación es una fase muy importante del programa ya que es la que permite mejorar la solución. Se pueden escoger distintas estrategias, pero la elección principal es entre la intensificación y la diversificación de la solución. Se puede dirigir la búsqueda hacia la máxima intensificación, eligiendo una nueva solución sólo si esta es mejor que la anterior, o por otro lado, favorecer la diversificación aceptando también soluciones peores, como en el Recocido Simulado. Otra posibilidad es la de utilizar la memoria a largo plazo para evitar el análisis de soluciones ya consideradas, como se hace en la Busqueda Tabù.

Finalmente el módulo de perturbación de la solución. Ésta tiene como finalidad sacar la solución del entorno del mínimo local en el cual se encuentra y explorar otras regiones del espacio de las soluciones, con el fin de encontrar una región con un nuevo mínimo local, menor que el anterior. La entidad de la perturbación tiene que ser definida con mucho cuidado. De hecho una perturbación excesiva puede transformar el ILS en un algoritmo de búsqueda aleatoria y entonces poco eficiente, mientras una demasiado débil puede ser no suficiente para sacar la solución del mínimo local.

En la imagen a continuación (Fig 7.2) se ilustra la función de los dos módulos de búsqueda local y perturbación de la solución.

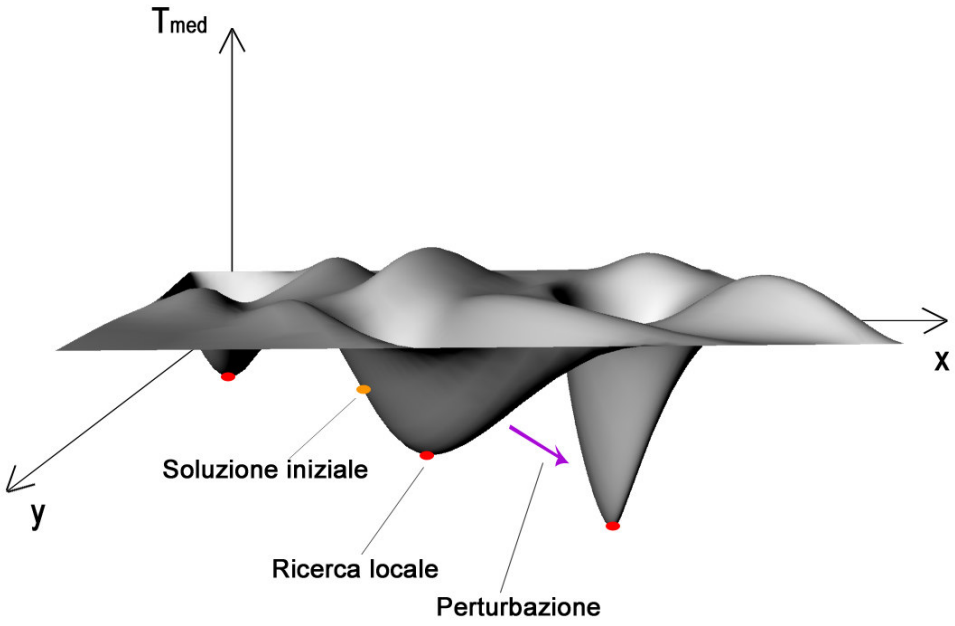


Figura 7.2 – Funcion de la busqueda local y de la perturbacion de la solucion.
Fuente: propia.

La fuerza de las metaheuristicas ILS reside en el binomio eficacia-sencillez. De hecho el método es competitivo con otros más complejos, pero está basado en una idea sencilla, lo que permite una implementación rápida en las versiones básicas. Además su estructura modular permite introducir las mejoras (y por lo tanto las dificultades) paso a paso, optimizando cada fase. En fin, el esquema del ILS es muy flexible con lo que el programador tiene suficiente libertad de acción y puede elegir que táctica aplicar para lograr los mejores resultados. Todo esto convierte este modelo en una potente herramienta

para la resolución de problemas complejos y explica su éxito en el campo de la programación de operaciones.

Módulo	Estrategias posibles
Solución inicial	- Solución aleatoria - Solución heurística
Búsqueda local	- Enfoque a la eficacia - Enfoque a la rapidez
Perturbación	- pequeña entidad - gran entidad
Criterio de aceptación	- enfoque a la intensificación - enfoque a la diversificación

Tabla 7.1 – Resumen de las posibles estrategia en cada modulo del ILS

7.2. Aplicación del ILS al problema

En esta sección se detalla el procedimiento de búsqueda local iterativa implementado para resolver nuestro problema de flowshop hibrido flexible con tiempos de preparación dependientes de las secuencias.

El procedimiento implementado es capaz de resolver problemas estáticos, generando programas activos sin retrasos. Además las subfunciones de carga y secuenciación se realizan simultáneamente, lo que generalmente produce mejores resultados.

7.2.1. Generación de la solución inicial

Con la finalidad de reducir los tiempos de convergencia se ha optado por una solución inicial de calidad. Esta se genera utilizando una versión modificada de la heurística ATCS (*Apparent Tardiness Cost with Setup*) para máquinas en paralelo [Lee y Pinedo, 1997 p. 464-474]. Estas modificaciones se vuelven necesarias para adaptar el método ATCS, que nace para la programación en configuraciones productivas de una sola etapa, a nuestro problema, caracterizado por más etapas en serie.

El ATCS es una heurística de secuenciación basada en el cálculo de un índice de prioridad (como por ejemplo son el SPT u el EDD); los trabajos a procesar se eligen en orden decreciente de índice. En el ATCS clásico para una sola etapa, el índice de la pieza genérica *i* tiene la siguiente expresión:

$$I(i,t,k) = \frac{w_i}{p_i} \cdot e^{\left[-\frac{\max[(d_i - p_i - t),0]}{k_1 \cdot \bar{p}}\right]} \cdot e^{\left[-\frac{S_{ik}}{k_2 \cdot \bar{s}}\right]} \qquad \text{(Ec. 7.1)}$$

El índice queda compuesto por el producto de tres términos, el primero fraccionario y los otros dos exponenciales.

El primero depende exclusivamente de los datos de la pieza y se conoce como término WSPT (Weighted Shortest Processing Time) $\frac{w_i}{p_i}$ donde:

w_i es llamado peso de la pieza y se utiliza para diferenciar las piezas por nivel de importancia. El peso aparece de forma linear y se encuentra en el numerador, así que el índice será mayor si el peso es mayor.

p_i es el tiempo de proceso. Aparece en forma linear y en el denominador, por lo tanto el método da mayor prioridad a las piezas que tienen un menor tiempo de proceso.

El primero exponencial es conocido como término de margen, $e^{\left[-\frac{\max[(d_i - p_i - t),0]}{k_1 \cdot \bar{p}}\right]}$ donde:

d_i es la fecha de vencimiento de la pieza

p_i es el tiempo de proceso de la pieza

t es el instante en que se evalua el índice

k_1 es el primer factor de escala, su expresion, como la del segundo factor de escala, será detallada más adelante

$\overline{p}$ es el tiempo medio de proceso, calculado como la media de los tiempos de proceso de todas las piezas.

Si la pieza considerada ya lleva retraso, o llevará retraso después de la tarea, el numerador del exponente será nulo, por lo tanto el exponencial cogerá su valor máximo, igual a uno. En caso contrario el exponencial será tanto más próximo a cero, cuanto más margen tenga la pieza, es decir la prioridad de la pieza disminuye proporcionalmente a su margen.

El factor k_1 tiene un efecto de amplificación sobre el término, cuanto más próximo a cero sea su valor, tanto mayor será la importancia del término en la expresión.

Finalmente, el último exponencial es el término de setup $e^{\left[-\frac{s_{ik}}{k_2 \cdot \overline{s}}\right]}$ donde:

s_{ik} es el tiempo de preparación de la pieza *i* cuando es precedido por la pieza *k.*

k_2 es el segundo factor de escala

$\overline{s}$ es el tiempo medio de preparación. Se calcula como la media de los tiempos de preparación de todas las piezas.

Esto tiene la función de desanimar la elección de trabajos que requieren tiempo de preparación largos con respecto a la media. El factor de escala k_2 tiene un efecto análogo a k_1 en el término anterior, o sea confiere más importancia al término, y entonces a los tiempos de preparación, a medida que vaya disminuyendo.

En nuestra versión modificada del ATCS se sustituyen unos conceptos de la configuración con una sola etapa, con los análogos en la configuración de flowshop híbrido, conservando

la estructura y los principios del método. Es evidente además que se ha introducido la dependencia del índice de la etapa de trabajo:

$$I(j,i,t,k) = \frac{w_i}{p_{rem,i}} \cdot e^{\left[-\frac{\max[(d_i - p_{rem,i} - t),0]}{k_{1,j} \cdot \overline{p_j}}\right]} \cdot e^{\left[-\frac{s_{ijk}}{k_{2,j} \cdot \overline{s_j}}\right]}$$ (Ec. 7.2)

Detallamos los cambios:

$p_i \rightarrow p_{rem,i}$: Se ha sustituido el tiempo de proceso de una pieza, con la suma de los tiempos de proceso que faltan para acabar su proceso. El cambio afecta el término WSPT y el término de margen.

En el WSPT el criterio de selección resultante se queda lo mismo, se confiere mayor prioridad a las piezas que saldrán antes del proceso de producción, y que, por lo tanto, tendrán ocupadas las máquinas el menor tiempo posible. Otra posibilidad habría sido dejar en este término el tiempo de proceso de la etapa corriente, se habría conferido así mayor prioridad a la tarea con menor duración en la etapa. Tal principio de selección pero, aparece miope ya que no considera la duración de las tareas en las etapas siguiente y por esto ha sido descartado.

En el término de margen la sustitución permite evaluar el margen que tendría la pieza si fuera procesada en todas las etapas siguientes, sin esperas. La idea entonces es la misma del ATCS tradicional.

$s_{ik} \rightarrow s_{ijk}$: Los tiempos de preparación son función no solo de la secuencia, sino también de la etapa. Hemos sustituido entonces el concepto de tiempo de preparación anterior con lo de tiempo de preparación de la pieza *i* cuando es precedido por la pieza *k* en la etapa *j*.

$k_1 \rightarrow k_{1,j}$, $k_2 \rightarrow k_{2,j}$: En una configuración flowshop los factores de escala serán distintos en cada etapa.

$\overline{p} \rightarrow \overline{p_j}$, $\overline{s} \rightarrow \overline{s_j}$: Lo mismo vale para los tiempos medios de proceso y de preparación.

La potencia de la heurística ATCS reside principalmente en sus dos parámetros de escala:

$$k_{1,j} = 1,2 \cdot \ln(\mu_j) - R \qquad k_{2,j} = \frac{\tau}{A_2\sqrt{\eta_j}} \qquad \text{(Ec. 7.3)}$$

Debido a estos factores, el índice no se calcula sólo sobre los datos de cada pieza, sino también teniendo en cuenta las características generales y estadísticas del problema que se está tratando cada vez. Es decir, los factores de escala tendrán un valor y no otro (y entonces los términos del ATCS tendrán mayor o menor importancia) dependiendo por ejemplo del retraso medio de las piezas, o del número medio de piezas que procesará cada máquina en una etapa. Esto permite un enfoque distinto con respecto a otras heurísticas que resuelven las instancias independientemente de sus peculiaridades.

Los factores de escala dependen de cuatro parámetros estadísticos:

- τ (comprendido entre cero y uno) indica lo cerca que están las fechas de vencimiento, cuanto más alto es el valor del coeficiente, más cerca están. Es una medida del retraso medio de las piezas.

$$\tau = 1 - \left(\frac{\overline{d}}{C_{\max}} \right) \qquad \text{(Ec. 7.4)}$$

- *R* es una medida de como las fechas de vencimiento están distribuidas en el tiempo. Si el valor de R es bajo, estas estan concentradas alrededor de la fecha media de vencimiento, en caso contrario estan distribuidas en un intervalo de tiempo más amplio.

$$R = \frac{(d_{\max} - d_{\min})}{C_{\max}} \qquad \text{(Ec. 7.5)}$$

- μ nos dice cuantas piezas en media procesa cada máquina en cada etapa.

$$\mu_j = \frac{n}{m_j} \qquad \text{(Ec. 7.6)}$$

- η es una medida de qué importancia tienen los tiempos de preparación con respecto a los tiempos de proceso.

$$\eta_j = \frac{\overline{s_j}}{\overline{p_j}} \qquad \text{(Ec. 7.7)}$$

Mientras τ y R son características de todo el ejemplar, μ y η cambian de etapa a etapa, esto comporta que también los factores de escala varíen en cada etapa.

Se puede observar también que en la expresión de τ y R aparece el instante de terminación de la última pieza: $C_{\max}$. Como estamos en fase de construcción de la primera solución, este valor queda desconocido, es necesario entonces hacer una estimación. A tal fin utilizaremos la formula siguiente obtenida modificando la fórmula original para una sola etapa de Lee y Pinedo (1997).

$$\widehat{C}_{Max} = \frac{\sum_{j=1}^{m}\left(\overline{s_j} + \overline{p_j}\right)\cdot \mu_j}{\overline{mpe}} \qquad \text{(Ec. 7.8)}$$

$\overline{mpe}$ es el número medio de máquinas por etapas.

Describimos ahora el procedimiento aplicado para generar la solución inicial: en cada etapa el procedimiento calcula el ATCS modificado para todas las combinaciones máquina-pieza disponibles en el instante considerado y elige la pareja con el máximo valor del índice. Entonces se actualiza la solución y se vuelve a aplicar el procedimiento. Es necesario volver a calcular el ATCS de las piezas cada vez que se actualiza la solución, porque habrán cambiado el instante de disponibilidad y la secuencia en una de las máquinas.

Si ninguna máquina o ninguna pieza queda disponible en el instante considerado, el programa adelanta el tiempo al momento en que habrá disponibilidad de la primera pareja máquina-pieza. Podemos decir que el tiempo se adelanta de forma discreta, o sea el reloj no avanza de segundo en segundo, en la manera clásica, sino que se desplaza directamente a los instantes significativos para la programación.

Cuando ya no quedan trabajos para realizar en la etapa actual, se pasa a la sucesiva, y el reloj vuelve al instante en que está disponible la primera pareja máquina-pieza. Así hasta llegar a la última etapa. Para mayor claridad el procedimiento está ilustrado a continuación (Fig 7.3)

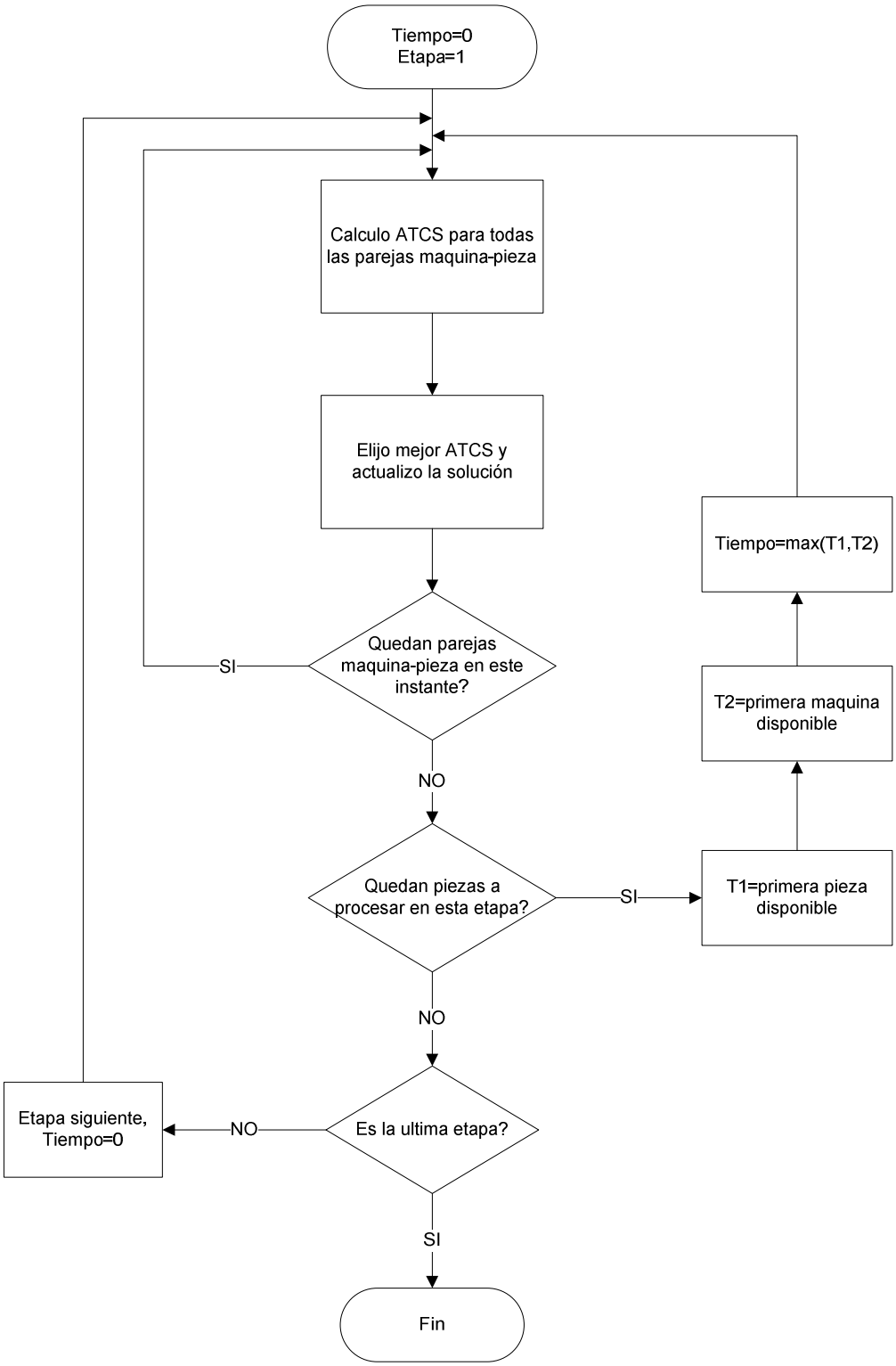


Figura 7.3 – Diagrama de flujo de la generacion de la solucion inicial con ATCS

7.2.2. Búsqueda local

Antes de empezar la descripción detallada del módulo de búsqueda local, es oportuno clarificar un concepto.

Tanto en la fase de búsqueda local, como en la de perturbación, la nueva solución se determina efectuando una modificación en la programación de la primera etapa y aplicando sucesivamente una regla heurística para la determinación de carga y secuencia en las etapas siguientes.

Se empieza con una solución, llamada solución de partida, e indicada más adelante como S, esta puede ser la solución inicial, o derivar del proceso de perturbación.

Se aplica a dicha solución una modificación en la secuenciación de una de las máquinas de la primera etapa, de tipo intercambio de piezas. Entonces se completa la nueva solución (de la segunda etapa hasta la última) utilizando el criterio heurístico ATCS modificado, que hemos descrito en el apartado anterior, llamaremos la nueva solución S'.

Sucesivamente se evalúa el retraso total de la nueva programación. Si este es superior a la de la solución que se acaba de modificar (retraso (S') > retraso(S)), se descarta la nueva solución y se incrementa un contador, llamado α. Se efectua entonces un primer control: si el contador ha alcanzado su valor umbral, el programa abandona la fase de búsqueda local, y pasa a la perturbación de la solución; en caso contrario se repite del principio la búsqueda local, aplicando una nueva modificación a la solución de partida.

Si el retraso de la nueva programación es menor que el de la solución de partida (retraso(S') < retraso(S)), el contador α se pone a cero, y la nueva solución pasa a ser la solución de partida ($S = S'$). Además se efectúa una segunda verificación: si la nueva solución de partida es mejor que la más provechosa encontrada hasta el momento, indicada con $S*$, entonces se sustituye a esta la nueva solución encontrada ($S* = S$).

Se añade un diagrama de flujo para aclarar el procedimiento (Fig 7.4).

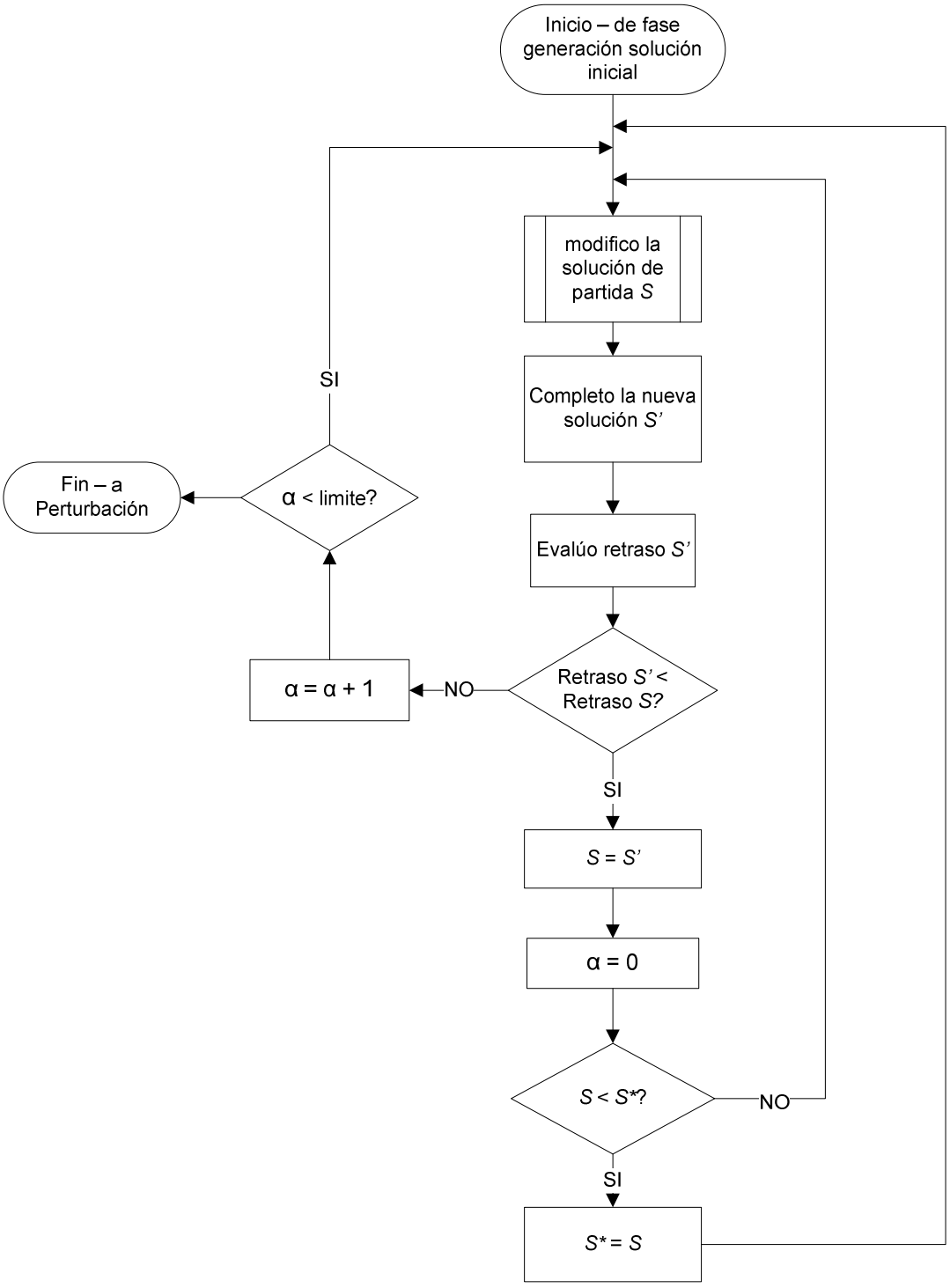


Figura 7.4 – Diagrama de flujo de la fase de búsqueda local. Fuente: propia.

Analizamos más en detalle la fase de modificación de la solución. Esta consiste en el *intercambio de la posición de dos piezas* dentro de la secuenciación de una de las máquinas de la primera etapa de trabajo: se elige una máquina al azar y se seleccionan dos piezas de la máquina elegida intercambiando su posición (Fig. 7.5).

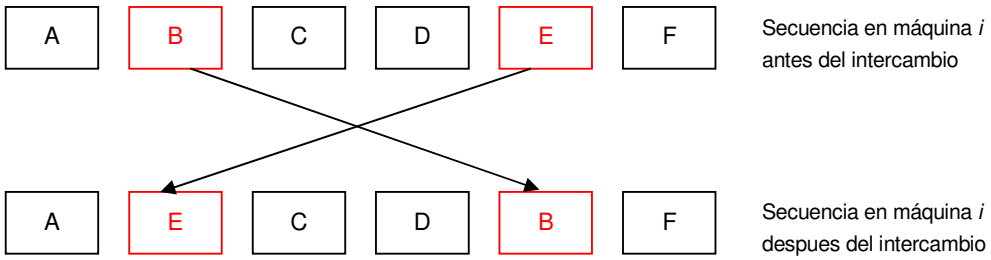


Figura 7.5 – Representación gráfica del intercambio de piezas durante la fase de búsqueda local. Fuente: propia.

Se completa la solución en las etapas siguientes mediante la heurística ATCS y se calcula el retraso de la nueva programación.

El programa dispone de un control para evitar eventos no deseados, por ejemplo la selección de una máquina que procesa una sola pieza, o la selección de la misma piezas para el intercambio.

La elección entre rapidez y eficacia de de la fase de búsqueda local no ha sido fijada *a priori* sino que es uno de los parámetros que se van a ajustar. De hecho, se permite escoger el valor umbral del contador α, es decir el número de veces que se repite la modificación de la solución en curso. A medida que el valor umbral suba, aumenta la calidad de la solución pero también aumenta el tiempo de cómputo necesario.

7.2.3. Criterio de aceptación

El criterio de aceptación se encuentra después de la búsqueda local y define la solución a la cual se aplicará la perturbación. Ha sido elegido un criterio que fomente la máxima diversificación: la solución que sale de la fase de búsqueda local es aceptada siempre

como nueva solución en curso del algoritmo y a ésta se aplicará la perturbación aunque no sea la mejor solución encontrada hasta entonces por el programa (se elige la solución S, no la $S*$).

La finalidad de la elección es la de explorar una cadena de mínimos locales que no tienen que tener obligatoriamente valores decrecientes en cada paso, evitando así que el algoritmo se quede atrapado alrededor de un mínimo local.

7.2.4. Perturbación de la solución

El diseño de la perturbación debe fijar el grado de diversificación deseado. Si la perturbación es fuerte la diversificación será alta permitiendo escapar de mínimos locales pero con el riesgo de saltar a zonas del vecindario donde la solución se aleja del óptimo. El grado se perturbación se ha fijado a través de un parámetro que ha sido calibrado mediante el diseño de experimentos.

Después de la búsqueda local y del criterio de aceptación la solución en curso entra en la fase de perturbación. Puesto que la perturbación aplica unos cambios aleatorios a la solución de partida, la solución resultante puede ser peor. Para limitar este fenómeno se ha realizado un proceso de perturbación más complejo: en vez de generar una sola solución perturbada, se genera una muestra y se escoge la mejor. De esta forma se limita el desplazamiento hacia regiones poco interesantes del espacio de las soluciones.

Se actualiza entonces un contador, llamado β, y se efectúa un control: si β es inferior a un valor umbral, el programa vuelve a la fase de búsqueda local, en caso contrario el programa se cierra y el output será la mejor de las soluciones encontradas hasta entonces.

Vamos a mirar más en detalle cómo se generan las soluciones perturbadas. Como se ha explicado en el apartado relativo a la búsqueda local, los cambios en la programación se introducen en la primera etapa y después se completan las soluciones utilizando un criterio heurístico, que en nuestro caso es otra vez el ATCS modificado.

Las soluciones modificadas se generan intercambiando la posición de algunas piezas que pueden pertenecer a la misma máquina o a máquinas diferentes. La cuestión de fijar el número de piezas a intercambiar, con la finalidad de proporcionar mejores resultados, es

objeto de estudio del próximo capítulo, de momento indicaremos este número genérico con la letra d.

El programa entonces elige dos máquinas de la primera etapa por medio de la generación de dos números aleatorios. Para cada máquina se escogen, siempre de forma aleatoria, d piezas y se generan tres intercambios:

1. Se intercambian las piezas de una máquina con las piezas análogas de la otra máquina. Se intercambia la posición de la *pieza 1* de la *máquina 1* con la posición de la *pieza 1* de la *máquina 2*. La posición de la *pieza 2* de la *máquina 1* con la posición de la *pieza 2* de la *máquina 2*. Y así hasta las piezas d.

2. Se intercambia el orden de las piezas pero éstas se quedan en las mismas máquinas. Se intercambia la *pieza 1* de la *máquina 1* con la *pieza* d de la *máquina 1*. Y la *pieza 2* de la *máquina 1* con la *pieza* (d *-1)* de la *máquina 1*. Y así para ambas máquinas.

3. Se intercambian las piezas de una máquina con las piezas opuestas de la otra máquina. Se intercambia la posición de la *pieza 1* de la *máquina 1* con la posición de la *pieza* d de la *máquina 2*. La posición de la *pieza 2* de la *máquina 1* con la posición de la *pieza* (d *-1)* de la *máquina 2*. Y así para todas las piezas.

Hacemos un ejemplo clarificador con $d = 2$. Consideramos una solución con la siguiente programación escrita en forma de matriz donde cada fila de la matriz corresponde a la secuenciación de las piezas asignadas a una máquina. Si se haya elegido para la perturbación las máquinas 1 y 3, y respectivamente las piezas 1, 4 y 8, 10 (Fig 7.6):

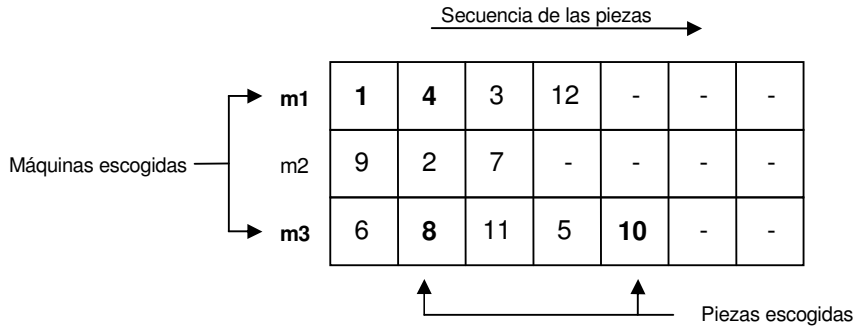
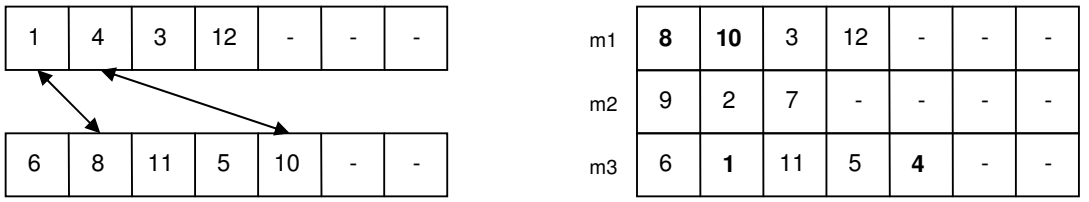


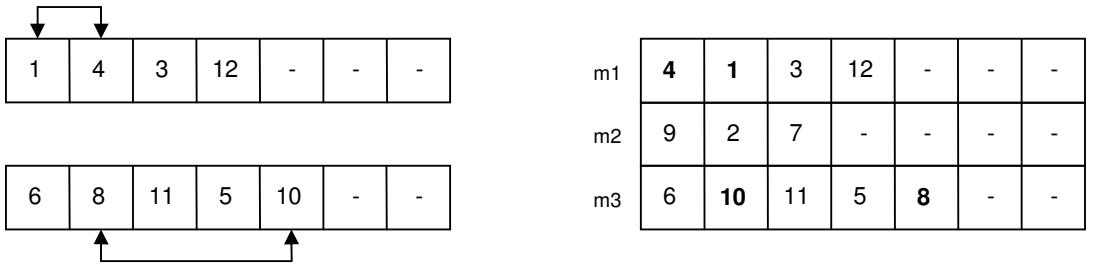
Figura 7.6 – matriz *1_stage_sequence* de una solución en curso entrada en la fase de perturbación.

Las tres perturbaciones resultantes serían:

1.



2.



3.

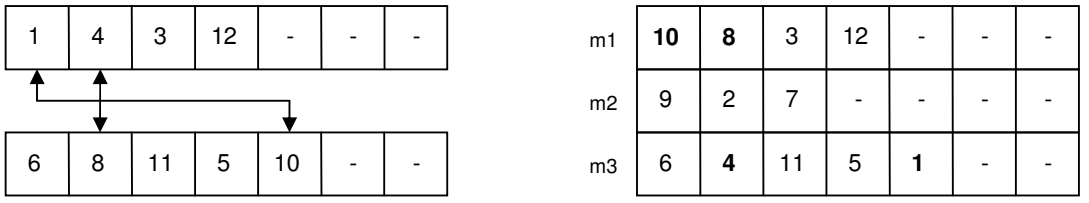


Figura 7.7 – Representación gráfica de las tres perturbaciónes evaluadas y relativa matriz *1_stage_sequence.*

7.3. Variantes del programa principal

Debido a la imposibilidad de obtener soluciones óptimas ya que no se conoce una colección de ejemplares con soluciones óptimas o, al menos conocidas, se ha decidido analizar la eficiencia de los procedimientos propuestos por comparación entre variantes del procedimiento propuesto. Los cambios que caracterizan cada una de estas variantes (Tab. 7.2) se detallan en los apartados a continuación.

	Solución inicial	Vecindario de búsqueda local	Heurística para completar la solución
Original	ATCS	Intercambio de piezas en la misma máquina	ATCS
Variante 1	*Aleatoria*	Intercambio de piezas en la misma máquina	ATCS
Variante 2	ATCS	*Inserción de piezas de una máquina a otra*	ATCS
Variante 3	ATCS	Intercambio de piezas en la misma máquina	*ECT*

Tabla 7.2 – Características de los cuatro programas analizados.

7.3.1. Variante 1

La primera variante del programa se centra en la generación de la solución inicial y tiene la finalidad de evaluar la importancia de empezar con una solución de buen nivel.

En vez de utilizar la heurística ATCS modificada, este programa determina la solución inicial de forma aleatoria: cada pieza es asignada a una máquina que se elige con la generación

de un número aleatorio. El procedimiento se repite para todas las etapas de la configuración productiva.

```
for(j=1,2,…,última etapa)
    for(i=1,2,…,última pieza)
      m=número_casual
      secuencia pieza i en máquina m
    endfor
endfor
```

Este procedimiento aleatorio sólo se utiliza en la solución inicial, es decir, sólo cambia el punto de partida del programa, mientras al momento de completar la programación de la primera etapa que se produce en los módulos de búsqueda local y perturbación se sigue utilizando el ATCS.

7.3.2. Variante 2

La segunda variante del programa modifica el vecindario utilizado en la búsqueda local. El procedimiento de mejora empleado se queda igual al original (Fig. 7.4).

En el programa original se utiliza un vecindario de intercambio de piezas en la misma máquina, en esta variante se utiliza la *inserción de piezas de una máquina a otra.* Detallamos el proceso: en cada iteración se eligen aleatoriamente dos máquinas. De la misma manera se elige una pieza de la primera máquina y una posición de la segunda, que puede estar ya ocupada por otra pieza o también estar al fin de la secuencia. Se desplaza entonces la pieza de la primera máquina a la posición elegida en la segunda (Fig 7.8).

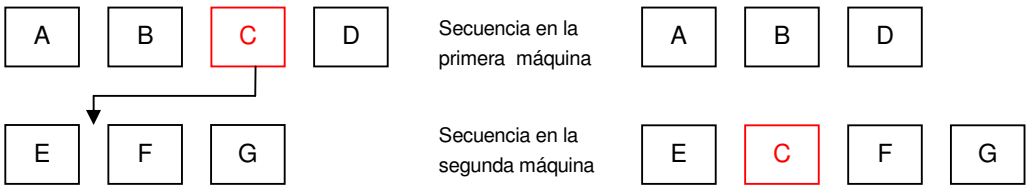


Figura 7.8 – Rapresentación gráfica de la secuencia antes y después de la modificación *inserción de piezas de una máquina a otra*. La pieza elegida en la primera máquina es la *C*, la posición elegida como destino de la segunda máquina es la dos.

Finalmente se completan las etapas siguientes de la programación con la heurística ATCS.

7.3.3. Variante 3

La tercera y última variante del programa cambia el método utilizado para completar las soluciones que se generan durante las fases de búsqueda local y perturbación. Como se ha explicado anteriormente, estas fases producen nuevas soluciones modificando la programación de la primera etapa y aplicando luego una regla heurística. En esta variante la regla usada no será el ATCS sino el ECT (pag. 23).

El procedimiento (muy parecido a lo que se ha explicado para el ATCS (Fig. 7.3)), empieza en la segunda etapa de trabajo y en el instante en que se encuentra disponible la primera pareja pieza-máquina. Se evalúa el índice ECT de todas las parejas disponibles y se elige el más bajo, o sea la combinación pieza-máquina que permite que se finalice el proceso de la pieza lo antes posible. Se sigue entonces con la programación de las parejas restantes. Es necesario volver a calcular el índice porque la secuencia como el tiempo de disponibilidad de una de las máquinas habrá cambiado.

Cuando no quedan parejas disponibles en el instante considerado el tiempo se adelanta. El procedimiento se repite hasta la última pieza, y la última etapa de trabajo.

8. Diseño de experimentos

En el presente capítulo se analiza la eficiencia de distintas opciones de diseño y parámetros del algoritmo ILS desarrollado. La calibración ha sido realizada sobre el programa original, pero breves experimentos realizados, han demostrado que los resultados obtenidos son también válidos para las tres variantes implementadas.

8.1. Parámetros de calibración

Son tres los parámetros que han sido calibrados:

- d : 2 niveles { 2, 4 }
- α : 2 niveles { n, 2n }
- β : 2 niveles { n, 2n }

El parámetro d es el número de intercambios que se realizan en la fase de perturbación de la solución. La elección de los niveles está basada en las alternativas que han proporcionado mejores resultados en la literatura para la resolución de problemas parecidos [Naderi, Ruiz y Zandieh, 2008, p. 236-246].

El segundo parámetro, α, es el número máximo de iteraciones consecutivas sin aportar mejora que se realizan en la fase de búsqueda local, antes de perturbar la solución. El valor de este parámetro ha sido relacionado con el número de piezas del ejemplar analizado.

Finalmente, β, es el número de veces que se repite el proceso, o sea el número de veces que se perturba la solución en curso antes de salir del programa. También en este caso, como en el anterior, se ha relacionado los niveles del parámetro con el número de las piezas.

Considerando todas las combinaciones de los parámetros de calibración se obtienen ocho posibles configuraciones del programa implementado. Estas serán ensayadas sobre un juego de ejemplares y se elegirá la configuración que proporcione los mejores resultados.

8.2. Instancias de calibración

Las colecciones de ejemplares utilizadas para la calibración están compuestas por 480 ejemplares: 10 ejemplares para cada una de las 48 combinaciones de las siguientes variables.

- *n* { 20, 50, 80 }
- *m* { 2, 4 }
- *mpe* { 2, U[1, 4] }
- *dd* { HH, HL, LH, LL }

n es el número de piezas a procesar. Han sido considerado ejemplares de veinte, cincuenta y ochenta piezas.

m es el número de etapas de trabajo. Ha sido fijado igual a dos o a cuatro.

mpe es el número de máquinas en paralelo en cada etapa. Los ejemplares se dividen en simetricos, caracterizados por tener dos máquinas en todas las etapas de trabajo, y asimetricos, para los cuales el número de máquinas en cada etapa se genera aleatoriamente según una distribución uniforme entre uno y cuatro.

Por fin *dd* son las fechas de vencimiento de las piezas. Estas se generan de forma aleatoria según una distribución uniforme entre dos valores que quedan definidos por las características del ejemplar considerado y por los coeficientes *T* y *R.* Se utiliza la formula siguiente [Potts y Van Wassenhove, 1982], adaptada al caso de flowshop híbrido:

$$dd = U\left[P\cdot\left(1-T-\frac{R}{2}\right), P\cdot\left(1-T+\frac{R}{2}\right)\right] \qquad \text{(Ec. 8.1)}$$

Analizamos la formula más en detalle:

P es una estimación de cuánto podría tardar en media la línea de producción en procesar todas las piezas. Como es conocido, el tiempo de un proceso de producción queda definido por su etapa más lenta, llamada *cuello de botella*. Por esto la *P* se elige como el máximo entre los tiempos medios de la producción de cada etapa: $\overline{p}_j$.

$$P = \max[\overline{p}_j] \qquad con \quad j = 1,...,m \qquad \text{(Ec. 8.2)}$$

Los tiempos medios de producción de cada etapa, p_j , se definen como la suma de los tiempos de proceso de todas las piezas en la etapa, partido por el número de máquinas en paralelo de la etapa, más la suma de los tiempos medios de preparación de cada pieza en la etapa, siempre partidos por el número de máquinas en la etapa:

$$\overline{p}_j = \frac{\sum_{i=1}^{n} p_{ij}}{\alpha_j} + \frac{\sum_{i=1}^{n}\sum_{k=0,k\neq i}^{n} S_{ijk}}{n \cdot \alpha_j} \qquad con \quad j = 1,...,m \qquad \text{(Ec. 8.3)}$$

T (Tardiness) representa el factor de retraso de las fechas de entrega. Mas alto es este valor, más cerca en estarán las fechas de entrega de las piezas, esto es la media de las fechas de entrega será menor del tiempo medio de proceso *P*. En los ejemplares analizados se han cogido los siguientes valores: *T=0,1* y *T=0,5.*

R (Range factor) es el factor de distribución en el tiempo de las fechas de entrega. Una vez que el factor *T* haya definido lo cerca que, en media, están estas fechas, *R* nos dice si estas están más concentradas cerca de la media (*R* alto), o distribuidas en un intervalo de tiempo más amplio (*R* bajo). En los ejemplares hemos utilizados los valores: *R=0,8* y *R=1,8*.

Combinando los valores de los parámetros *T* y *R* se obtienen las cuatro fechas de vencimiento distintas que se han indicado anteriormente:

- HH: T=0,5 y R=1,8
- HL: T=0,5 y R=0,8
- LH: T=0,1 y R=1,8

- LL: T=0,1 y R=0,8

Nuestra elección de los parámetros no impide que se generen fechas de entrega negativas, o sea anteriores al momento de la generación del programa de producción. El método de resolución implementado por lo tanto es capaz de resolver también casos en que hayan sido desatendidas las fechas de entrega.

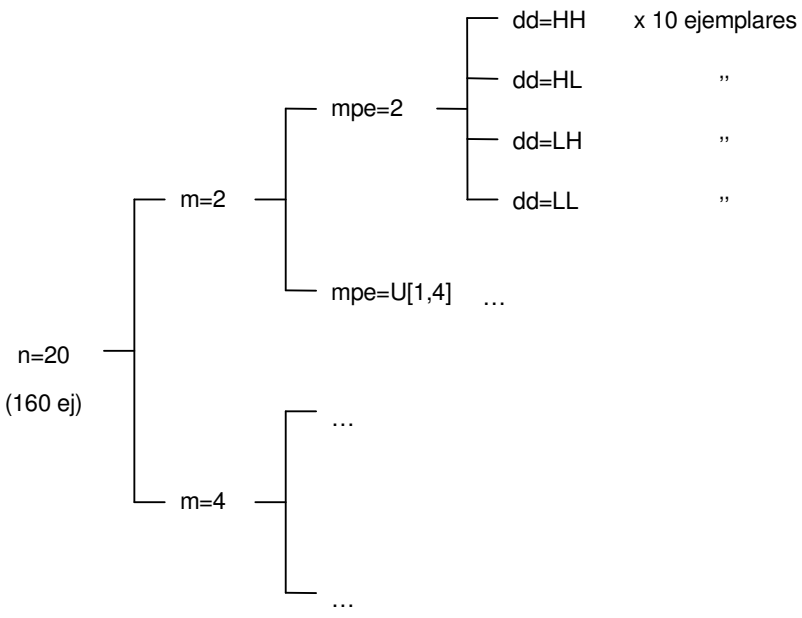


Figura 8.1 – Rapresentación gráfica del arbol de las instancias. Fuente: propia.

Nos queda por definir de qué forma se han escogido los otros datos del problema. El peso ha sido considerado igual para todas las piezas, y de valor unitario. Los tiempos de proceso y preparación se han generado según una distribución uniforme, entre 0 y 99.

$$p_{ij} = U[0,99] \qquad S_{ijk} = U[0,99] \qquad \text{(Ec. 8.4)}$$

Finalmente la probabilidad de saltar una etapa de trabajo para las piezas se ha fijado con una distribución uniforme, entre el 10% y el 40%.

8.3. Resultados

Como hemos comentado anteriormente, debido a la imposibilidad de tener soluciones óptimas o conocidas de buen nivel, los resultados de las distintas configuraciones del ILS han sido analizados comparándolos entre ellos. El índice usado para efectuar la comparación se denomina *error relativo normalizado* [Lee, Kim y Choi, 2004].

$$error = \frac{solucion_{hallada} - solucion_{mejor}}{solucion_{peor} - solucion_{mejor}} \qquad \text{(Ec. 8.5)}$$

El error relativo normalizado es un valor comprendido entre cero (si la solución coincide con la mejor solución encontrada) y uno (caso en que la solución coincide con la peor solución encontrada).

Otros índices de error, como por ejemplo el error relativo (Ec. 8.6), generalmente usados en problemas de minimización del $C_{\max}$ no proporcionan en nuestro caso una descripción objetiva de los resultados porque para resultados óptimos que tienden a cero, también pequeñas desviaciones del óptimo generan errores muy grandes.

$$error = \frac{solucion_{hallada} - solucion_{mejor}}{solucion_{mejor}} \qquad \text{(Ec. 8.6)}$$

Los resultados han sido analizados mediante el *análisis de la varianza*, conocido como ANOVA (Analysis of Variance). El análisis de varianza sirve para comparar si los valores de un conjunto de datos numéricos son significativamente distintos a los valores de otro o más conjuntos de datos. El procedimiento para comparar estos valores está basado en la varianza global observada en los grupos de datos numéricos a comparar. Típicamente, el análisis de varianza se utiliza para asociar una probabilidad a la conclusión de que la media de un grupo de puntuaciones es distinta de la media de otro grupo de puntuaciones. En nuestro caso la usaremos para definir qué opciones de diseño generan una media de error significativamente inferior respecto a las otras.

El ANOVA parte de tres supuestos previos, los cuales han sido verificados con éxito en nuestro caso.

1. Independencia de los experimentos realizados

2. Distribución normal de los resultados

3. Homogeneidad de la varianza

Los resultados del análisis se enseñan a continuación. La primera comparación se efectúa entre configuraciones del programa con distintos valores de d. Como se enseña en la gráfica (Fig. 8.2), la diferencia entre los valores medios no sólo es estadísticamente significativa (los intervalos de confianza no se solapan), sino también relevante. La opción de diseño $d = 2$ logra errores mucho más bajos y por esto es la solución elegida.

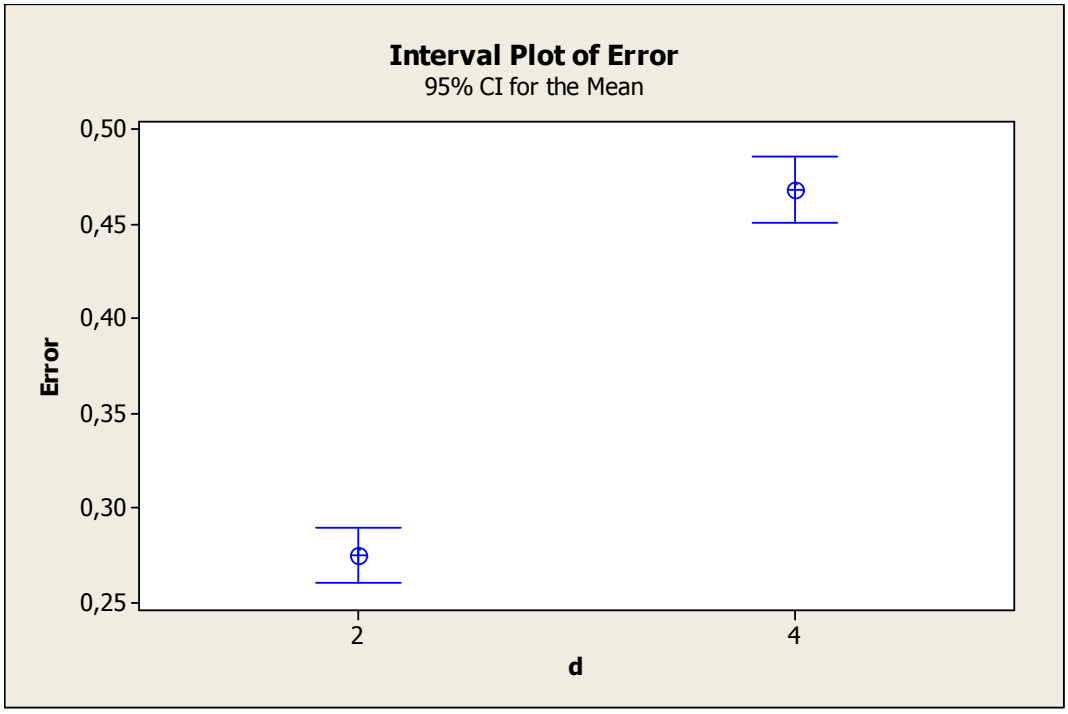


Figura 8.2 – Media e intervalo de confianza al 95% para las configuraciones con d distinta.

En previsión de desarrollos futuros es interesante notar como al aumentar del número de piezas del ejemplar, aumente el valor medio de los errores de las configuraciones con $d = 2$, mientras disminuye lo de las configuraciones con $d = 4$ (Fig 8.3). Esto sugiere que

el número óptimo de intercambio a efectuar en la fase de perturbación sea proporcional al número de piezas.

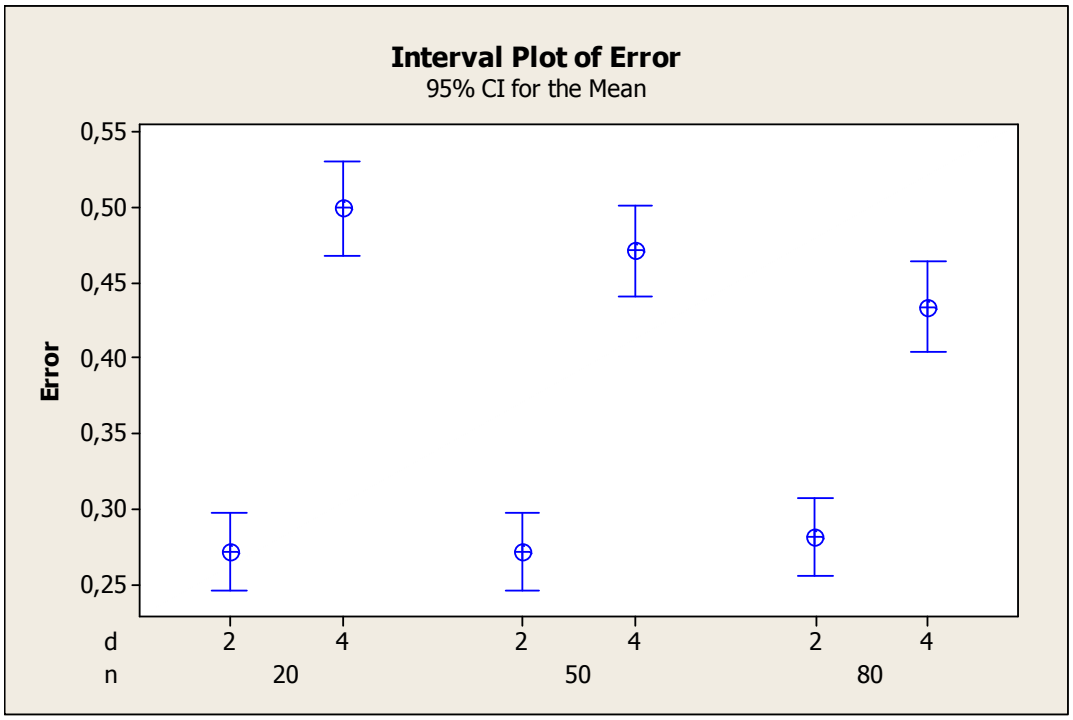


Figura 8.3 – Media e intervalo de confianza al 95% para las configuraciones con d distinta, divididas por número de piezas del ejemplar.

La segunda comparación se realiza para determinar los parámetros α y β, o sea el número de iteraciones respectivamente de las fases de búsqueda local y perturbación. También en este caso la situación aparece bien definida: los errores decrecen al aumentar de las iteraciones, como se podía imaginar (Fig 8.4). Además la dependencia aparece lineal. Se podría pensar entonces aumentar aún más el valor de los parámetros, alcanzando soluciones cada vez mejores; sin embargo esto comportaría un aumento consistente de los tiempos del procedimiento, contrario a nuestro objetivo de proporcionar soluciones de calidad, pero en tiempos aceptables. Se elige por lo tanto, entre las consideradas, la solución de minimiza los errores: $\alpha = 2n$ y $\beta = 2n$.

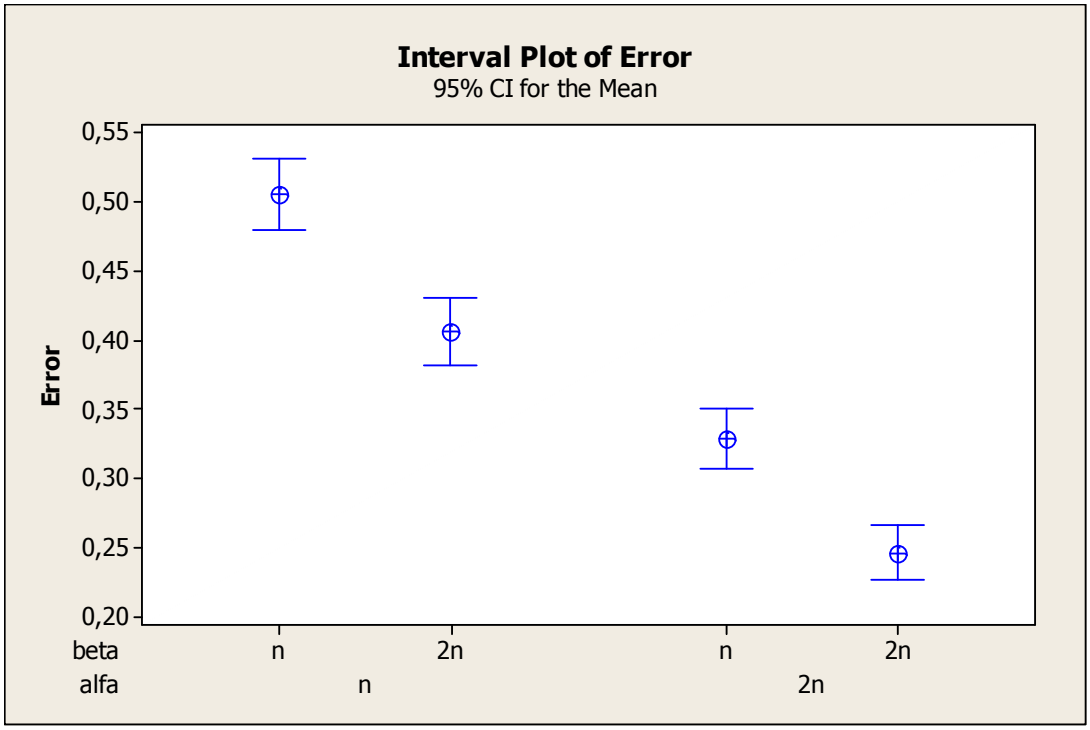


Figura 8.4 – Media e intervalo de confianza al 95% para las configuraciones con α y β distintas. Se nota la disminución lineal del error al aumentar de las iteraciones.

9. Experiencia computacional

En este capítulo se compara la eficiencia de las cuatro versiones propuestas del programa. Como no se disponía de un problema concreto al que aplicar los algoritmos, la comparación se ha realizado ensayando los programas sobre otro juego de 720 ejemplares distintos: 15 para cada combinación de las variables siguientes:

- Número de piezas: *n* { 20, 50, 80 }
- Número de etapas: *m* { 2, 4 }
- Número de máquinas por etapa: *mpe* { 2, U[1, 4] }
- Fechas de vencimiento: *dd* { HH, HL, LH, LL }

Más detalles acerca del modo en que se generaron las instancias se pueden encontrar en el apartado 8.2, mientras el índice utilizado para la comparación está descrito en el apartado 8.3.

El ordenador utilizado para la experiencia presenta las características siguientes:

- Microprocesador: Intel Centrino Duo 2400 MHz
- Memoria RAM: 2 Gb

Al ejecutar el programa, aparece la interface de usuario (Fig. 9.1) en la que se debe introducir el fichero con los ejemplares que se quiere procesar y el fichero de salida. Los parámetros del programa aparecen marcados de acuerdo al ajuste realizado, pero se deja al usuario la posibilidad de modificarlos según necesidad.

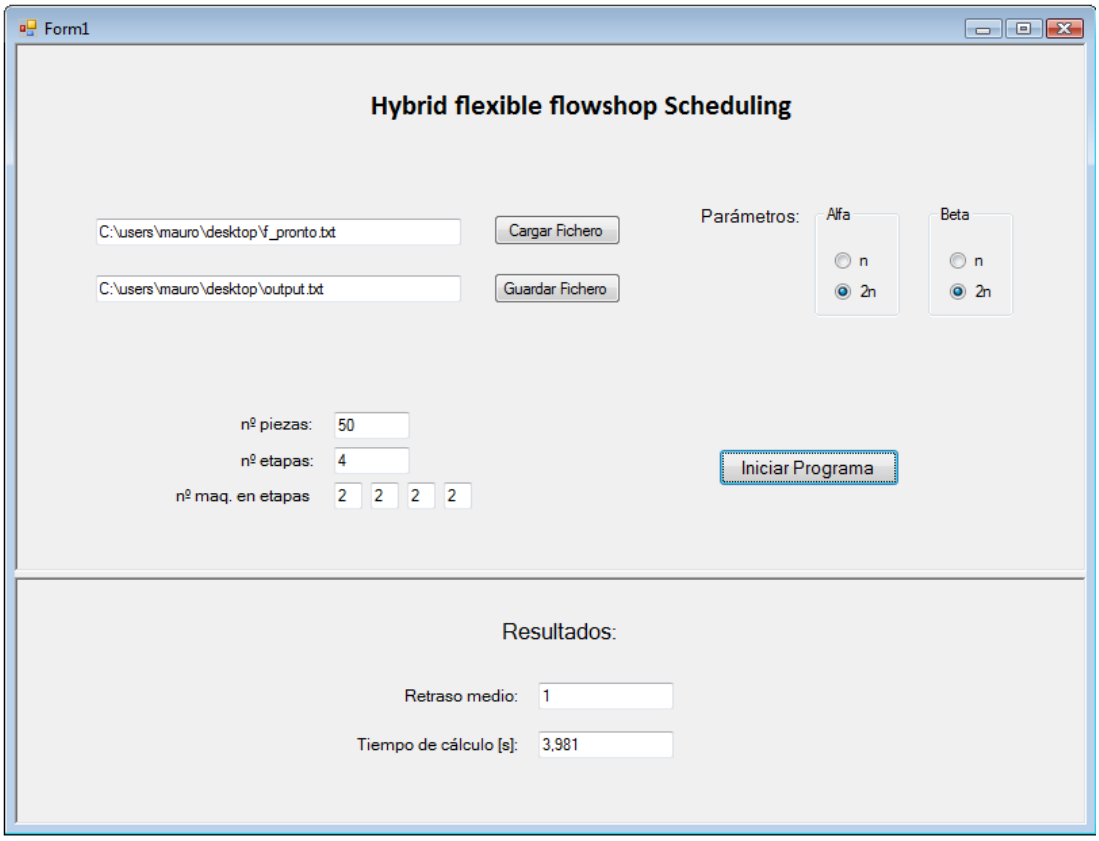


Figura 9.1 – Ventana principal del programa.

9.1. Análisis de los resultados

El análisis ANOVA de los errores de los cuatro programas indica claramente como el original y la variante 2 proporcionan los resultados mejores, mientras la variante 1 genera resultados de nivel muy inferior a los otros programas (Fig 9.2). La razón de esta ineficiencia reside en la importancia de la solución de partida. La variante 1 es el único programa que empieza con una programación aleatoria. La velocidad de convergencia es proporcional a la solución inicial, por lo tanto, fijado el número de iteraciones que se efectúan, el programa con solución de partida random alcanzará un resultado mucho peor con respecto a los que empiezan ya de una solución de calidad.

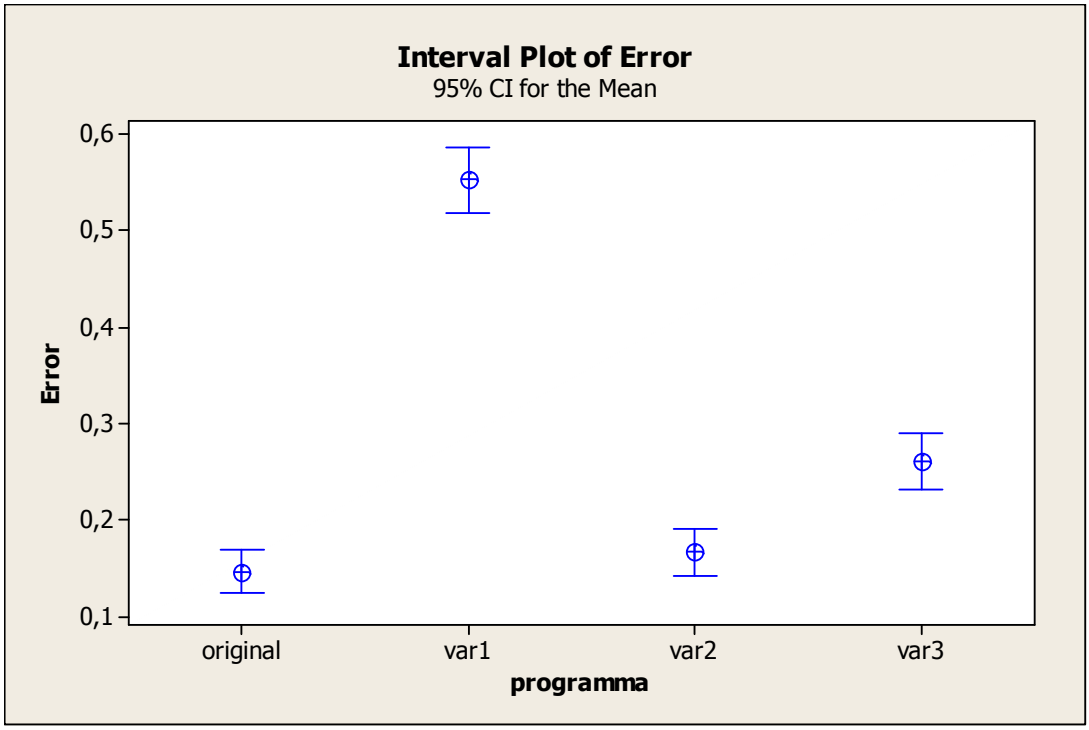


Figura 9.2 – Media e intervalo de confianza al 95% para los cuatro programas. El programa original y la variante 2 permiten lograr los mejores resultados.

Se deduce que utilizar la heurística ATCS modificada para determinar la solución inicial ha sido una elección correcta.

La diferencia entre los resultados del programa original y de la variante 2 es mínima y no estadísticamente significativa: los intervalos se solapan así que no es cierto si uno de los dos programas genere soluciones mejores que el otro. Los dos programas difieren por un aspecto fundamental: en el original las fases de búsqueda local y perturbación son complementarias, en la variante 2 esto no ocurre. De hecho en el primero la búsqueda local desplaza las piezas dentro de la misma máquina, mientras la perturbación intercambia piezas de una máquina a otra. Por lo tanto en la fase de búsqueda local no se pueden anular los cambios realizados por la perturbación, sino solo buscar la mejor solución posible modificando la secuenciación en cada máquina. Es decir la búsqueda local sólo modifica la secuenciación y la perturbación rediseña la carga (lo que implica cambios en la secuenciación). En cambio en el segundo esta división no existe, y ambos módulos

modifican carga y secuenciación. Los resultados demuestran que ambos enfoque son validos.

La variante 3, que utilizaba la heurística ECT para completar las soluciones en la primera etapa, proporciona resultados aceptables, pero lejos de los alcanzados por el programa original. Esto valora todavía más la hipótesis que el índice ATCS modificado sea apropiado para la resolución de problemas en este entorno.

El análisis de los resultados divididos por número de piezas de la instancia ratifica estas observaciones (Fig. 9.3). Se puede notar además como todos los programa excepto la variante 1 alcancen resultados mejores con el aumentar de la complejidad de la instancia.

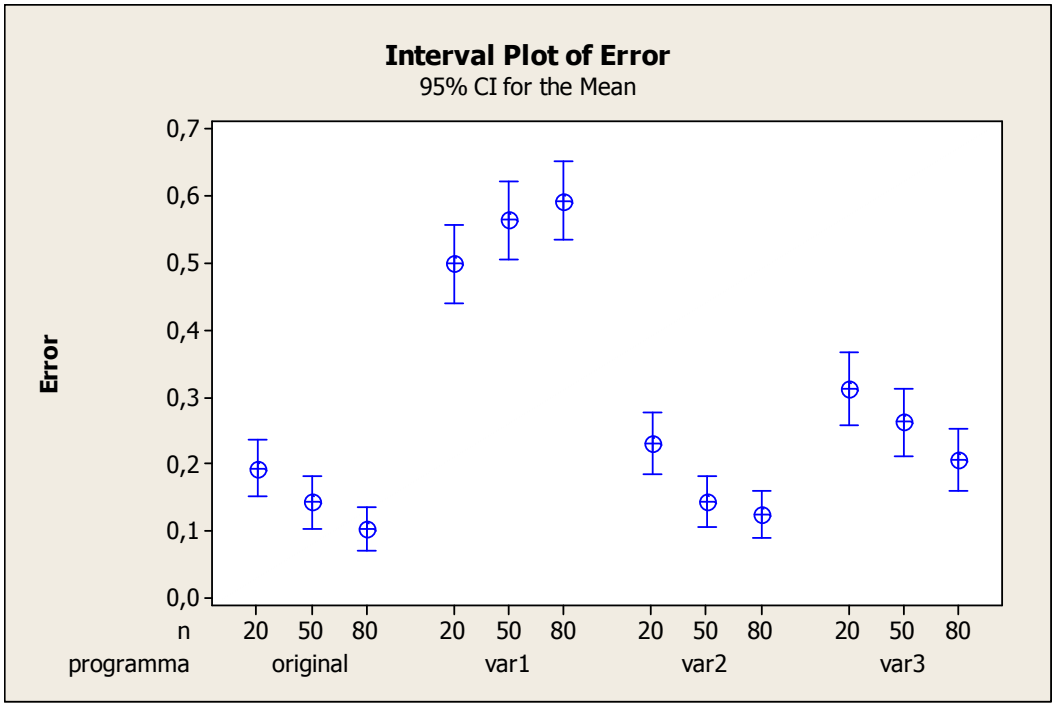


Figura 9.3 – Media e intervalo de confianza al 95% para las configuraciones de los programas, divididos por número de piezas de las instancias. El programa original y la varante 2 se confirman los más performantes

Otro aspecto interesante para comparar los programas es el número de soluciones con retraso nulo encontradas (Fig. 9.4). Los resultados confirman en sustancia los del análisis

ANOVA: El programa original es el más eficiente y consigue encontrar soluciones con retraso nulo aproximadamente en un caso sobre tres. Siguen la variante 3 y la dos, con resultados parecidos, mientras la variante 1 obtiene los peores resultados.

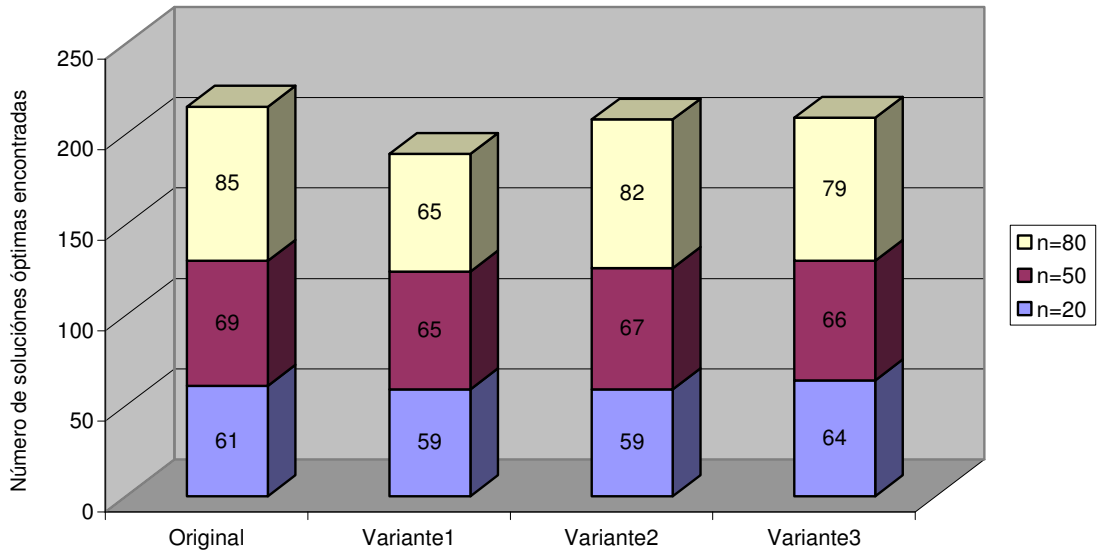


Figura 9.4 – Istograma del número de soluciónes con retraso nulo encontradas por los programas.

Finalmente, vamos a analizar la evolución de los tiempos de cálculo. Todas las versiones del programa presentan un aumento importante del tiempo de proceso con la complejidad del ejemplar, en particular, los tiempos parecen seguir aproximadamente una ley cuadrática, si el número de piezas se dobla, los tiempos de cálculo se multiplican por cuatro. Esta tendencia era previsible, ya que el número de iteraciones ha sido fijado proporcional al cuadrado del número de las piezas ($\alpha \cdot \beta$).

También en este caso, contrariamente a lo que se podría pensar, la variante 1 obtiene los resultados peores. Esto es debido probablemente al hecho que empezando con una solución de muy baja calidad, esta se mejorará muy a menudo, y esto requiere continuas actualizaciones. Además, como se ha explicado anteriormente, cada vez que se consigue una mejora en la fase de búsqueda local, el contador de las iteraciones efectuadas se pone a cero, con lo que se acaba haciendo más iteraciones.

El programa más rápido, en cambio, es la variante 3 y la motivación es la mayor sencillez de la heurística ECT respecto a la ATCS modificada. Esta variante necesita en media un 34% de tiempo menos para proporcionar la solución, pero, por otro lado, sus soluciones presentan un error medio mayor del 44% respecto al programa original. Por lo tanto se considera preferible, en una programación enfocada a la eficiencia, utilizar la versión original respecto a la variante 3, por lo menos en ejemplares con un nivel de complejidad comparable a los analizados en la experiencia computacional, ya que el incremento absoluto de los tiempos de cálculo en estos casos dificilmente supera algunos segundos.

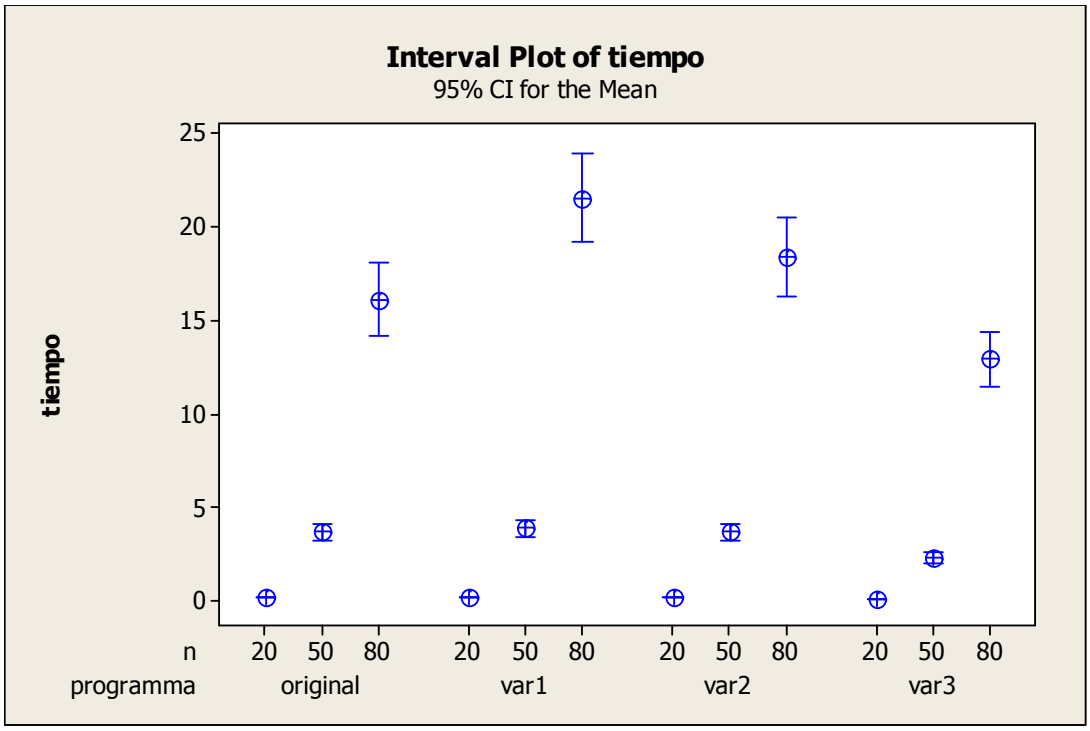


Figura 9.5 – Media e intervalo de confianza al 95% de los tiempos de cálculo para de los cuatro programas.

10. Presupuesto

En este apartado se realiza una valoración económica de la globalidad del proyecto, analizando las fases que lo componen. La realización del proyecto requeriría dos profesionales: un ingeniero industrial, que tendría que estudiar el problema y elaborar un proceso de resolución y un informático cuya tarea sería básicamente traducir dicho proceso en código de programa.

Se supone que el ingeniero posea sólidos conocimientos de dirección de operaciones y que el informático sea experto del lenguaje de programación utilizado y por lo tanto solo tendrían que adaptarse al caso particular.

Enumeramos las fases de que sería compuesto el proyecto, añadendo una breve descripción de las tareas que se lleva a cabo en cada una:

Estudio del caso particular: Búsqueda y estudio de publicaciones que traten problemas de flowshop híbrido flexible, con tiempos de preparación dependientes de la secuencia y índice de eficiencia de minimización de retraso medio. Determinación del *estado del arte.*

Estudio de los métodos de resolución: Estudio de los métodos más utilizados y provechosos para resolver el problema. Individuación de ventajas y desventajas.

Diseño del algoritmo: Elección del método que se ajuste más a las necesidades del problema y determinación de la estrategia a seguir para obtener los mejores resultados. Definición de los pasos que componen el algoritmo.

Programación software en C#: Traducción de las operaciones que componen al algoritmo, en código del lenguaje escogido.

Diseño de interface: Diseño de la interface gráfica necesaria para una versión comercial del programa.

Pruebas y depuración: Evaluación sistemática del programa sobre unos juegos de prueba. Determinación y corrección de las situaciones que pueden llevar a fallo.

Análisis resultados y propuestas de mejora: Analizando los resultados obtenidos en los juegos de prueba, se determinan los parámetros que proporcionan resultados mejores. Además se buscan eventuales puntos débiles del algoritmo y se introducen los cambios que se creen necesarios.

Las fases han sido divididas por área de competencia entre los dos profesionales y para cada una se ha estimado una duración aproximada. El salario horario de las dos categorías, compreso de IVA, ha sido calculado cogiendo como punto de partida el sueldo medio anual de profesionales con experiencia. Los resultados se detallan en la tabla a continuación (Tab. 10.1).

Categoría Profesional	*Concepto*	*Tiempo [h]*	*Salario [€/h]*	*Coste total [€]*
Ingeniero industrial	Estudio caso particular	60	25	1 500
	Estudio métodos de resolución	35	25	875
	Diseño del algoritmo	70	25	1 750
	Análisis resultados y mejoras	35	25	875
Informático	Programación software en C#	70	20	1 400
	Pruevas y depuración	30	20	600
	Diseño interface	20	20	400
		Total (IVA incluido)		**7 400**

Tabla 10.1 – Cálculo del presupuesto del proyecto

11. Análisis del impacto ambiental

En el entorno industrial actual, caracterizado por el desafío de la sostenibilidad, es necesario considerar, ya en la fase de diseño, beneficios y riesgos medioambientales derivados de los proyectos que se quere desarrollar.

11.1. Beneficios medioambientales de la implantación

En nuestro análisis no se han detectado riesgos derivados de la hipotética implantación de nuestra herramienta informática en una empresa. Es posible en cambio individuar varios beneficios posibles.

La búsqueda de un programa de fabricación que mejore el retraso medio comporta generalmente la reducción de la duración total del proceso, lo que supone una reducción en el gasto de energía y por lo tanto de combustibles fósiles. Todo esto se traduce en última instancia en una disminución de la contaminación del aire y del efecto invernadero. Además será suficiente una refrigeración menor, entonces menos liquido de refrigeración, lo que implica menor contaminación térmica del agua.

Tiempos de proceso menores significan también menor desgaste de las máquinarias, que tendrán entonces una vida útil más larga. Por consiguiente se habrá un ahorro de materias primeras y menores costes de eliminación.

Otro teme importante es el transporte de producto acabado. Utilizando nuestra herramienta una empresa podría ajustar más sus previsiones y optimizar la logística, explotando el tamaño de los contenedores de transporte y eliminando viajes para la entraga de productos fabricados con retraso.

Finalmente, el uso de una herramienta informática para gestionar la programación puede reducir la necesidad de suporte físico para datos y cálculos con la consiguiente reducción en el uso de papel.

Conclusiones

El problema tratado en este trabajo es la programación de operaciones en un sistema flowshop híbrido flexible, con tiempos de preparación dependientes de la secuencia. La medida de eficiencia considerada es el retraso medio de las piezas.

Se ha propuesto un procedimiento de programación basado en un algoritmo de tipo ILS.

Un primer diseño de experimentos ha permitido fijar los parámetros del algoritmo: número de iteraciones, grado de la perturbación.

A continuación se han evaluado las eficiencia del procedimiento implementado, por comparación con tres variantes del mismo, que difieren o en el procedimiento de obtención de la solución inicial o en el vecindario utilizado en la búsqueda local o en la heurística usada para completar las soluciones en la primera etapa de trabajo.

Los resultados han evidenciado que la calidad de la solución obtenida depende en gran medida de la calidad de la solución inicial: soluciones iniciales de alto nivel resultan en una convergencia más rápida, y entonces, a paridad de tiempo disponible, en resultados mejores.

Los dos enfoques considerados en la fase de búsqueda local no han llevado a diferencias estadísticamente relevantes ni de los resultados, ni de los tiempos de cálculo.

Por último, el índice heurístico de prioridad, ECT, utilizado para completar la programación de la primera etapa, tiene fuerte influencia tanto sobre los resultados como sobre los tiempos de cálculo: un criterio más complejo como el ATCS genera soluciones mejores, pero comporta también tiempos de cálculo más largos.

En definitiva el algoritmo implementado, caracterizado por una solución inicial de calidad, una búsqueda local basada en el intercambio de piezas en la misma máquina, y un índice heurístico ATCS para programar los trabajos en cada etapa es el que resulta más competitivo

El programa admite desarrollos y mejoras futuras, sugerimos algunas:

- Los factores de escala de la heurística ATCS han sido calculado con fórmulas usadas en el caso de configuración productiva compuesta por una sola etapa. Sería interesante estudiar la influencia de estos factores sobre los resultados, y comprobar si las fórmulas siguen siendo válidas para configuraciones con más etapas, o si se pueden introducir mejoras.

- Estudiar el efecto de un criterio de aceptación de empates como el utilizado en la metaheurística *Simulated annealing.*

- Introducir la penalización de entregas anticipadas (criterio JIT).

- Estudiar el impacto, en la calidad de la solución, que tiene el aumentar el número de muestras generadas en la fase de perturbación de la solución.

Bibliografia

Referéncias bibliográficas

[1] COMPANYS, R. *Secuenciacion.* Barcelona, CPDA, 2003, pp. 99-100

[2] CONWAY, R.W.; MAXWELL, W.L.; MILLER, L.W. *Theory of scheduling*, Addison-Wesley, 1967, p. 5-6

[3] LAWLER, E. L.; LENSTRA, J. K.; RINNOOY KAN, A. H. G., y SHMOYS, D. B. *Recent developments in deterministic sequencing and scheduling: A survey*. In Lenstra Dempster & Kan Rinnooy (Eds.), *Deterministic and Stochastic Scheduling*, Dordrecht, Netherlands: D. Reidel Publishing Company, 1982, p. 35-74

[4] VIGNIER, A.; BILLAUT, J.-C.; PROUST, C. *Scheduling problems of hybrid flowshop type: state of the art*, RAIRO Operations Research, 1999, Vol. 33 (2), p. 117–183

[5] RIBAS, I. *Programación multicriterio de un sistema productivo con flujo regular sin esperas y estaciones en paralelo. Aplicación a una fábrica de helados.* Tesis Doctoral, Univerdad Politécnica de Valencia, 2007

[6] ZANAKIS, S.H.; EVANS, J.R. *Heuristic 'Optimization': Why, When, and How to Use It*, Interfaces, 1981 Vol. 11(5)

[7] OSMAN, I.H.; KELLY, J.P. Metaheuristics: Theory & Applications, Kluwer Academic Publishers, Boston, USA, 1996

[8] GLOVER, F. *Tabu Search — Part I*, ORSA Journal on Computing, 1989, Vol. 3, p. 190-206.

[9] HOLLAND, J.H. , *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[10] FEO, T.A.; RESENDE, M.G.C.; *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters, 1989, Vol. 8, p. 67–71.

[11] HOOS, H.H.; STÜTZLE, T. *Stochastic local search: foundations and applications*, Morgan Kaufmann Publishers, San Francisco, CA, 2005

[12] LIN, S.; KERNIGHAN, B.W.. *An Effective Heuristic Algorithm for the Traveling-Salesman Problem*, Operations Research, 1973, Vol. 21, p. 498-516.

[13] RUIZ, R.; MAROTO, C. *A comprehensive review and evaluation of permutation flowshop heuristics*, European Journal of Operational Research, 2005, Vol. 165 (2), p.479-494

[14] STÜTZLE, T. *Iterated Local Search & Variable Neighborhood Search*, M.N. Summerschool, Tenerife, 2003

[15] LEE, Y.H.; PINEDO, M. *Scheduling jobs on parallel machines with sequence-dependent setup times.* European Journal of Operational Research Vol. 100, 1997, p. 464-474

[16] NADERI, B.; RUIZ, R.; ZANDIEH, M *Algorithms for a realistic variant of flowshop scheduling*, Computers & Operations Research, 2009, Vol. 37 (2), p. 236-246

[17] POTTS, C.N. ; VAN WASSENHOVE, L.N. *A decomposition algorithm for the single machine total tardiness problem*, Operations Research Letters, 1982, Vol. 1 (5), p. 177-181

[18] LEE, G.C.; KIM, Y.D.; CHOI, S.W.; *Bottleneck focused scheduling for a hybrid flowshop.* International Journal of production research, 2004, Vol. 42 (1), p. 165-181.

Bibliografia complementária

En este apartado se detallan las fuentes que se han utilizado, pero no han sido citadas a lo largo del texto.

[19] COMPANYS, R. *Secuenciacion.* Barcelona, CPDA, 2003, pp. 99-100

[20] PINEDO, M. *Scheduling: Theory, Algorithms and Systems*, New York, Springer, 2008

[21] MATEO, M; VALHONDO, J.B; COSTA, C.T.; IBAÑEZ, J.G.; COMPANYS, R. *Direcció d'operacions*, Transparencias, Barcelona, CPDA

[22] VALHONDO, J.B. *Transparencies d'organització Industrial*, Transparencias, Barcelona, CPDA