Resumen

El caso que se trata es el del problema del taller mecánico con flujo regular y máquinas en paralelo.

En concreto, mediante la implementación de la heurística GRASP (*Greedy Randomized Adaptative Search Procedure*), se pretende obtener la secuencia de programación para preparar un conjunto de pedidos que dé lugar al menor retraso medio posible en la entrega de dichos pedidos.

Cada uno de estos pedidos está compuesto por un lote de un tamaño determinado de un mismo y único artículo (es decir, no se contempla el *picking*), existiendo un número limitado de artículos diferentes. Del mismo modo, estos pedidos pueden asignarse a una serie de máquinas diferentes situadas en paralelo, y en las cuales tiene lugar una única operación.

Existen limitaciones en la capacidad de fabricación de las máquinas: no todas las máquinas son capaces de fabricar todos los artículos. Asimismo, cada máquina lleva asociado un tiempo de preparación derivado de adaptar la máquina para fabricar un artículo a otro diferente, así como un tiempo de proceso o fabricación.

El retraso es la diferencia entre el momento en que se acaba de fabricar el pedido y la fecha de entrega del mismo.

El algoritmo empleado para la resolución del problema es la heurística GRASP, que se compone de dos fases consecutivas:

- La primera permite obtener una primera solución de partida mediante la aplicación de un algoritmo *greedy*.

- La segunda consiste en, partiendo de la solución inicial, valorar los vecinos de dicha solución y obtener como solución final la que dé lugar al menor retraso medio.

A fin de verificar el correcto funcionamiento y la eficiencia del procedimiento de resolución propuesto, se ha llevado a cabo una experiencia computacional sobre tres juegos de datos, diferenciándose en el número de máquinas disponibles y el número de artículos diferentes susceptibles de ser fabricados: caso 8 artículos y 3 máquinas, caso 12 artículos y 6 máquinas y caso 15 artículos y 9 máquinas.

Cada juego consta, a su vez, de 100 casos diferentes, conteniendo de entre 15 a 25 pedidos diferentes, con sus correspondientes datos.

Sumario

1. Glosario

i: índice de artículo

n: número máximo de artículos

j: índice de máquina

m: índice máximo de máquinas

k: índice de lote

z: número máximo de lotes

p_{ij}: tiempo de proceso unitario del artículo *i* en la máquina *j*

$tp_{(i \to i')j}$ o $ST_{(i \to i')j}$: tiempo de preparación o *setup – time* asociado a pasar de fabricar el artículo *i* al artículo *i'* en la máquina *j*

c_k: tiempo de salida del lote *k*

d_k: fecha de entrega del lote *k*

q_k: tamaño del lote *k*

T_k: retraso del lote *k*

n_k: artículo de que se compone el lote *k*

Reloj: momento temporal en que sucede un acontecimiento en el sistema (salida de un lote de una máquina, por ejemplo)

2. **Prefacio**

2.1. Origen del proyecto

La programación de operaciones es una herramienta ampliamente utilizada en las industrias. Existen numerosos procedimientos para establecer una secuencia de fabricación que otorgue buenos resultados, cuyo potencial depende en gran medida de su capacidad de adaptación a cada caso concreto.

Resulta útil y recomendable conocer dichos métodos (normalmente agrupados en heurísticos, metaheurísticos y exactos), así como saber implementarlos adecuándolos a las exigencias de cada problema.

En este caso, el método objeto de estudio es el algoritmo GRASP, que ha sido desarrollado y aplicado sobre un conjunto de datos asimilables a casos de la realidad.

Finalmente, los resultados obtenidos se compararán con los propuestos por otro método, el algoritmo genético, a fin de determinar su eficiencia en la resolución de los mismos casos propuestos.

2.2. Motivación

Uno de los principales problemas que se presentan en el área operativa de las empresas dedicadas a fabricación es la programación de las operaciones a realizar a fin de optimizar la producción, es decir, organizar los recursos disponibles de la mejor manera posible para servir a los clientes los pedidos recibidos: cómo asignar cada pedido a las máquinas disponibles.

Normalmente, en empresas pequeñas y poco informatizadas, el orden de fabricación se establece siguiendo criterios creados a base de la experiencia previa, de una manera arbitraria, poco objetiva, y sin poder garantizar que la solución presentada sea la mejor posible.

La utilización de heurísticas y metaheurísticas para el establecimiento de los programas de fabricación garantiza obtener soluciones no necesariamente óptimas, pero sí con un índice de aceptabilidad elevado. Además, el hecho de seguir algoritmos concretos permite que las soluciones obtenidas sean objetivas, permitiéndose la comparación entre procedimientos.

La motivación del proyecto responde a dos fines claramente delimitados.

Por un lado, el estudio del algoritmo GRASP como herramienta empleada en la programación de operaciones, y más concretamente, en el caso expuesto: programar la producción de *z* pedidos, compuestos cada uno por un número determinado de unidades de *n* artículos diferentes, en *m* máquinas diferentes en paralelo.

Por otro lado, responde a fines estrictamente comparativos: conocer cuál de entre dos heurísticas, un algoritmo Genético (GA) o un algoritmo GRASP, proporciona, en términos generales, los mejores resultados y presenta una mayor eficiencia.

2.3. Requerimientos previos

Como requisitos previos, se requieren los archivos de texto con los datos del problema (según el formato pfc_n_m.text, donde n es el número de artículos y m el número de máquinas de que constan los datos). Del mismo modo, es imprescindible conocer los resultados obtenidos mediante el Algoritmo Genético para el conjunto de datos de partida: fundamentalmente, el retraso medio y el tiempo que necesita el programa informático para proporcionar los resultados.

Partiendo de esta información, es posible obtener resultados y llevar a cabo comparaciones entre ambas metodologías.

3. Introducción

3.1. Objetivos del proyecto

El objetivo de este trabajo es el desarrollo de un procedimiento que, mediante el establecimiento de unos datos de entrada o *inputs* como puedan ser el número de máquinas disponibles en el taller, el número de artículos a fabricar, los tiempos de procesado, los tiempos de preparación, el tamaño de los lotes, etc. ofrezca como respuesta u *output* la asignación, secuenciación y temporización de los pedidos a fabricar en las máquinas disponibles para optimizar los tiempos muertos y obtener retrasos medios aceptables.

El caso concreto corresponde a un problema tipo taller con las máquinas en paralelo y flujo regular.

Los pasos de fabricación serían los siguientes:

1. Llega un pedido del cliente/es, que estará constituido por *k* lotes diferentes, conteniendo cada uno de ellos una cantidad determinada de un único tipo de artículo, y debiendo ser entregados en una fecha concreta.

2. Debe asignarse a cada máquina cada uno de los pedidos, sabiendo que las máquinas son diferentes, esto es, no están preparadas para fabricar todos los artículos.

3. Cada máquina consta de unos tiempos de preparación y de unos tiempos de fabricación para cada artículo.

4. Se lleva a cabo una única operación en cada máquina. Según los artículos salen de la máquina, se consideran acabados y listos para su expedición.

5. La finalidad del programa de producción es minimizar los tiempos muertos y por ende, el retraso total.

Así pues, debemos saber:

- Qué pedido debe ir en cada máquina, esto es, qué máquinas fabricarán cada pedido
- En una misma máquina (a la que ya se le han fijado los pedidos que fabricará) establecer el orden de cada pedido.

Finalmente, los resultados obtenidos sirven de comparación entre métodos, permitiendo conocer cuál es el comportamiento de los algoritmos GRASP y Genético frente al tipo de problemas expuestos y a los ejemplares tratados, y permitiendo discernir cuál es el que proporciona mejores resultados en términos generales.

3.2. Alcance del proyecto

El procedimiento de resolución propuesto ofrece resultados para todos aquellos casos en los que exista un número finito de máquinas y un número determinado de artículos distintos (o gama). Y aquellos problemas en los que deba fabricarse un número determinado de pedidos, formados cada uno de ellos por un único tipo de los artículos disponibles.

El programa informático desarrollado para automatizar la aplicación del procedimiento propuesto está preparado para elaborar programas de hasta 100 pedidos diferentes, con una gama de 100 artículos diferentes y disponiendo de unos recursos de hasta 100 máquinas.

Por este hecho, el método es aplicable a la mayoría de industrias que cumplan con las condiciones productivas especificadas.

4. Problema del taller mecánico

4.1. Notación de las variables

Para el caso taller, la notación es la siguiente [1]: *n/m/A/B* → $n/m/1/T_{med}$

- *n*: número de artículos diferentes
- *m*: número de máquinas en paralelo
- *A*: tipo de flujo; en este caso, cuando un producto sale una máquina se considera acabado. Por tanto, no existe flujo entre máquinas.
- *B*: índice de eficiencia; en este caso T_{med} o retraso medio

En el caso que cada lote tuviera un peso específico diferente debido, por ejemplo, a contratos de exclusividad, clientes preferentes, nivel de pérdidas por lote no servido, etc., se atribuirá una ponderación a cada pedido ω_k. Así pues, en este caso: $n/m/1/\omega_k T_{med}$

4.2. Clasificación del problema del taller mecánico

El problema del taller mecánico se puede clasificar de tres maneras diferentes [1]: problema estático, semidinámico y dinámico, cada uno con sus correspondientes características diferenciales.

4.2.1. Problema estático

- Las piezas en número finito y determinado deben realizarse en un taller con un número finito de máquinas.
- En el instante de realizar la programación, se conoce:
    - La ruta de cada pieza.
    - Las operaciones de que se compone cada pieza.
    - En qué máquina debe realizarse cada operación.
    - La duración correspondientes de cada operación.

- Todas las piezas y máquinas están disponibles en el mismo instante que habitualmente se adoptará como inicial o tiempo 0.

- FINALIDAD: buscar un programa que optimice un índice de eficiencia establecido.

    - F_{max}: tiempo de permanencia máximo
    - c_{max}: instante máximo de salida de la pieza
    - T_{med}: retraso medio
    - T_{max}: retraso máximo
    - L: huelgo
        - Si $L<0$ la entrega se realiza con antelación
        - Si $L>0$ la entrega se realiza con retraso

4.2.2. Problema semidinámico

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

- En el instante de realizar la programación, se conoce:

    - La ruta de cada pieza.
    - Las operaciones de que se compone cada pieza.
    - En qué máquina debe realizarse cada operación.
    - La duración correspondiente de cada operación.

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

- FINALIDAD: buscar un programa que optimice un índice de eficiencia establecido, tales como los descritos en el apartado 4.2.1.

4.2.3. Problema dinámico

- El horizonte de funcionamiento del taller es ilimitado hacia el futuro.

- El número de piezas, pero no el de máquinas, es también ilimitado.

- Todas las piezas a tratar en el futuro no están definidas en un momento determinado; la definición de las piezas se conocerá progresivamente a medida que va transcurriendo el tiempo.

- Una pieza queda definida por completo cuando la orden se emite o llega al taller (o anteriormente justo antes de la llegada efectiva).

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

- Un único programa no es suficiente pues su validez queda en entredicho al transcurrir el tiempo y al alterarse la situación. Deben establecerse ciclos de reprogramación.

- FINALIDAD: establecer un procedimiento de programación y juzgar la calidad de un procedimiento. En este caso, los índices de eficiencia, como los expuestos en el apartado 4.2.1, se asocian a las características medias de los programas a lo largo de un intervalo temporal suficiente.

4.2.4. Hipótesis de Conway, Maxwell y Miller

Conway, Maxwell y Miller (1967) señalaron una serie de hipótesis que, generalmente, suelen ser aceptadas en el problema del Taller Mecánico:

1. Cada máquina está siempre disponible continuamente desde un instante $f \geq 0$ hasta *T*, donde *T* es arbitrariamente grande.

2. En las rutas de las piezas no se producen convergencias (montajes o ensamblajes) ni divergencias (partición en lotes). Cada operación tiene una sola operación precedente inmediata (exceptuando 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. Sólo hay una máquina de cada tipo en el taller.

5. Cuando una operación ha comenzado no se admiten interrupciones.

6. No pueden solaparse dos operaciones de la misma pieza, ya sea 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.

Las hipótesis 3 y 4 no se cumplirán en este problema; el resto, en cambio, sí se cumplirán.

4.2.5. Secuencias finitas para una máquina

Si se analiza una única máquina, la secuencia de pedidos que se ejecutarán consecutivamente en ella se puede fijar mediante diversas reglas heurísticas:

- Secuencia conforme a la duración o SPT (*Shortest Processing Time*): corresponde a la regla heurística "ejecutar primero la operación más corta".
- Secuencia conforme a la fecha de vencimiento o EDD (*Earliest Due Date*): corresponde a la regla heurística "ejecutar primero lo más urgente".
- Secuencia conforme al margen o SFT (*Shortest Float Time*): corresponde a la regla heurística "ejecutar primero lo que tiene menos margen".

4.2.6. Critical Ratios

Un *Critical Ratio* (CR) es un índice usado para determinar la evolución temporal de una tarea respecto al tiempo programado. Matemáticamente, se encuentra dividiendo el tiempo efectivo de finalización de una tarea respecto al tiempo que está programado acabarla.

En función del valor que adopte, su significado cambia de acuerdo con lo indicado a continuación:

- Un valor de $CR = 1$ indica que la tarea se ha acabado "a tiempo", es decir, sin faltar ni sobrar tiempo para su realización.
- Un valor de $CR < 1$ indica que la tarea está por detrás de su tiempo programado para su realización, es decir, está atrasada.
- Un $CR > 1$ indica que la tarea se ha adelantado respecto al tiempo proyectado para su finalización.

4.3. Significado gráfico de los conceptos expuestos

Mediante un sencillo ejemplo práctico, se presentan los conceptos de CR, EDD y SFT.

Datos:

- Máquinas: $m = 2$
- Artículos: $n = 2$
- Pedidos: $z = 2$
- Fecha de entrega del pedido 1: $d_1 = 20$
- Fecha de entrega del pedido 2: $d_2 = 15$
- Tiempo de proceso del artículo 1 en la máquina 1: $p_{11} = 4$
- Tiempo de proceso del artículo 2 en la máquina 1: $p_{21} = 5$
- Tiempo de proceso del artículo 1 en la máquina 2: $p_{12} = 10$
- Tiempo de proceso del artículo 2 en la máquina 2: $p_{22} = 10$

Se calcula, en función del tiempo (abscisas) el *Critical Ratio* (CR_{ij}, artículo i en máquina *j*) y el retraso por pedido (T_k del pedido *k*), según:

$$CR_{ij} = \frac{d_k - t}{p_{ij}} \qquad \text{(Ec. 4.1)}$$

$$T_k = \left(t + p_{ij}\right) - d_k \qquad \text{(Ec. 4.2)}$$

El siguiente gráfico muestra la representación de los CR y *T* en función del tiempo de reloj:

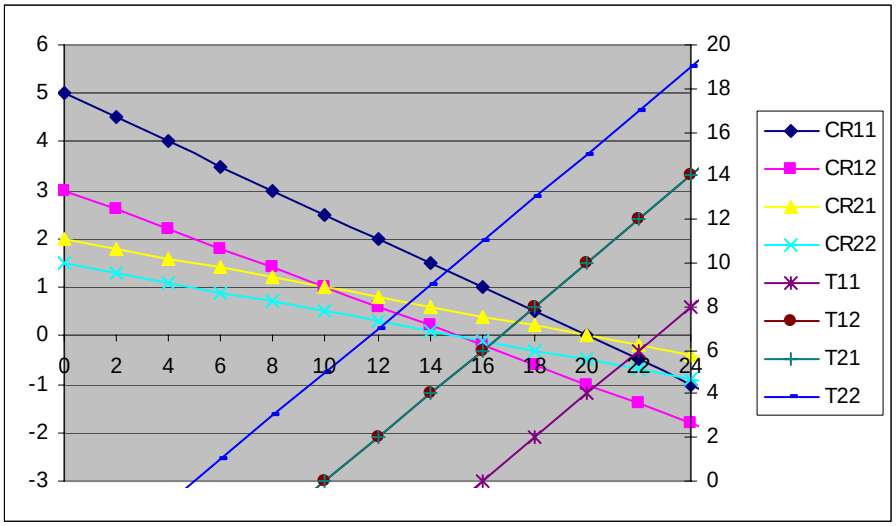


Fig. 4.1. Representación gráfica del *Critical Ratio* (CR) y el Retraso

Como puede observarse, las rectas CR_{ij} son de pendiente negativa (es evidente que conforme pasa el tiempo, estamos más cerca de generar retraso). En el momento en que llegan a un valor de *CR = 1*, el pedido correspondiente empieza a generar retraso, el cual va aumentando conforme va pasando el tiempo.

Igualmente, puede deducirse el *Earliest Due Date* (EDD) de cada pedido, que es la lectura en abscisas del punto en que las rectas CR cortan con el cero. Se observa que el EDD coincide con la fecha de entrega de cada pedido.

	EDD
p_1m_1	20
p_2m_1	15
p_1m_2	20
p_2m_2	15

Tabla 4.1. *Earliest Due Date* de cada pedido en cada máquina

El SFT (*Shortest Float Time*) es el momento en que empieza a generarse retraso en los pedidos, es decir,

$$SFT_{ij} = d_k - \left(t + p_{ij}\right) \qquad \text{(Ec. 4.3)}$$

Gráficamente, corresponde con la lectura en abscisas del momento en que *CR = 1*, o bien cuando se inicia el retraso *T*.

	SFT
p_1m_1	16
p_2m_1	10
p_1m_2	10
p_2m_2	5

Tabla 4.2. *Shortest Float Time* de cada pedido en cada máquina

Evidentemente, el pedido 2 procesado en la máquina 2 es el que empezará a generar retraso antes, y por tanto, sería recomendable empezar a procesarlo lo antes posible.

5. **Programación de operaciones**

5.1. Introducción

La programación de operaciones o *Scheduling* [1] busca definir de forma concreta:

- En cuál de los recursos disponibles debe ejecutarse cada una de las operaciones necesarias para realizar las órdenes de trabajo emitidas.
- Los instantes (fechas) en que debe tener lugar dicha ejecución.

5.2. Subfunciones de la programación de operaciones

Las tres subfunciones de la programación de operaciones son:

- La carga, que asigna a cada operación el recurso (máquina) en que se va a ejecutar. Regla "quién va a hacer qué". Existen tres posibles modelos:
    - Minimizar el coste
    - Maximizar el tiempo libre de las máquinas
    - Lote de piezas de un solo tipo realizado en la misma máquina
- La secuenciación, que define el orden o secuencia en que se ejecutarán las operaciones asignadas al mismo recurso. En el caso del problema del taller mecánico, la secuenciación debe tener en cuenta:
    - *n* piezas (lotes de piezas, órdenes de trabajo) deben realizarse en máquinas (recursos, secciones, puestos de trabajo)
    - La realización de cada pieza implica la ejecución, en orden establecido, de una serie de operaciones prefijadas.
    - Cada operación está asignada a una de las *m* máquinas y tiene una duración (tiempo de proceso) determinada y conocida.
    - Debe establecerse un programa, 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.

- La temporización, que se encarga de establecer las fechas de la ejecución (el calendario).

Los buenos programas implican:

- El desarrollo interrelacionado de las tres subfunciones, pero por las dificultades habitualmente se desarrollan programas en que destaca la independencia entre carga y secuenciación.

- La temporización es imposible desintegrarla de la secuenciación en la mayoría de los casos.

5.3. Programación de operaciones en máquinas en paralelo

En la programación de operaciones en máquinas en paralelo ("*Parallel Machine Scheduling*") es común identificar tres casos diferentes [2]:

- Máquinas idénticas: en este caso, una tarea puede ser realizada en cualquiera de las máquinas en el mismo tiempo.

- Máquinas uniformes: cada máquina puede realizar tareas a diferentes velocidades, esto es, una tarea requerirá menos tiempo de operación en una máquina más rápida que en otra más lenta. En este caso, generalmente, cada máquina lleva asociado un "factor de velocidad".

- Máquinas independientes o inconexas: 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.

El caso expuesto corresponde con el tercer caso: programación de *k* tareas (lotes) en *m* máquinas no homogéneas en paralelo. "No homogéneo" significa que no todos los lotes pueden ser procesados en cualquier máquina, sino sólo en un conjunto predeterminado.

6. Formulación del problema

El caso tratado se centra en una sección específica que consta de *m* máquinas en paralelo, diferentes entre sí, que realizan una misma y única operación.

Se pretende buscar la solución óptima para la fabricación de *k* pedidos, con tamaños de lote y fechas de entrega diferentes.

Cada máquina presenta unos tiempos de preparación al pasar de fabricar un tipo de artículo a otro diferente, que se desglosan en, por ejemplo:

- Paro de la actividad
- Limpieza de los restos del artículo anterior
- Cambio de moldes, recipientes de recepción, etc.
- Reinicio de la actividad

A continuación se exponen los parámetros de que se compone el problema tratado.

6.1. Tiempos de preparación

La estructura de los datos de tiempos de preparación de un artículo a otro se refleja en la siguiente matriz:

$tp_{ii'}$	**Artículo 1**	**Artículo 2**	**Artículo *i***	**Artículo *n***
Artículo 1	0	tp_{21}	tp_{i1}	tp_{n1}
Artículo 2	tp_{12}	0	tp_{i2}	tp_{n2}
Artículo *i'*	$tp_{1i'}$	$tp_{2i'}$	0	$tp_{ni'}$
Artículo *n*	tp_{1n}	tp_{2n}	tp_{in}	0

Tabla 6.1. Tiempos de preparación entre artículos para una misma máquina

Nótese que $tp_{ii'} \neq tp_{i'i}$, es decir, el tiempo de preparación de un artículo *i* a un artículo *i'* no tiene por qué coincidir con el tiempo de preparación de requerido entre el artículo *i'* y el artículo *i*.

6.2. Tiempos de proceso unitarios

Cada máquina *j* requiere unos tiempos de trabajo o procesado para elaborar cada uno de los artículos *i*, p_{ij}.:

p_{ij}	**Artículo 1**	**Artículo 2**	**...**	**Artículo *n***
Máquina 1	p_{11}	p_{21}	p_{i1}	p_{n1}
Máquina 2	p_{12}	p_{22}	p_{i2}	p_{n2}
...	p_{1j}	p_{2j}	p_{ij}	p_{nj}
Máquina *m*	p_{1m}	p_{2m}	p_{im}	p_{nm}

Tabla 6.2. Tiempos de proceso unitarios de cada artículo en cada máquina

6.3. Datos de los pedidos

La estructura de datos de los pedidos que deben fabricarse es la siguiente:

Pedido *k*	**Tamaño de lote q_k**	**Artículo *n***	**Fecha de entrega d_k**
1	q_1	n_1	d_1
2	q_2	n_2	d_2
...	q_k	n_k	d_k
z	q_z	n_z	d_z

Tabla 6.3. Datos de los pedidos

Para cada pedido *k*, se establecerá el tamaño de cada lote, el producto *n* de que se compone cada uno y la fecha de entrega de cada uno de ellos.

El objetivo es encontrar, utilizando diferentes procedimientos (heurísticos, metaheurísticos, exactos), la asignación y la secuencia u orden de trabajo que permita optimizar los tiempos muertos y por tanto, reducir el retraso medio.

Se penaliza la entrega tardía del lote, pues puede ocasionar costes de **diferimiento**: el cliente acepta el lote, pero a cambio de una rebaja en el precio estipulado.

Se puede proponer penalizar la fabricación anticipada del lote, es decir, que el tiempo de fabricación concluya antes del momento de la entrega. Esto supone unos costes de posesión que incluyen:

- Creación y mantenimiento de la capacidad de almacenaje
- Entrada y salida de los artículos en el stock
- Variación del valor de los bienes almacenados: obsolescencia, robos, caducidad
- Costes de seguridad
- Cargas financieras del capital inmovilizado

Esta condición adicional permite operar siguiendo la técnica del *Just In Time* (JIT), esto es, sirviendo los pedidos en el momento en son solicitados por los clientes, ni antes ni después. Esto implica que se pretenda trabajar con un stock cero, por lo menos, de producto acabado.

6.4. Variables, restricciones y función objetivo

A fin de establecer el algoritmo que conduzca a la solución buscada, es preciso definir las variables, las restricciones y la función objetivo.

- Variables:
    - Instante de tiempo: t
    - Tiempo de inicio del pedido *k*: s_{tk}
    - Tiempo de finalización de un pedido *k* (*completion time*): c_k
    - Retraso del pedido *k* (*tardiness*): T_k

- Constricciones – restricciones:
    - Tiempos de fabricación de cada artículo i en cada máquina j (*processing time*): p_{ij}
        - Restricción inamovible si se mantienen las mismas máquinas
    - Tiempos de preparación de un artículo *i'* cuando *i* ha sido la pieza anterior de la secuencia en la máquina *j* (*setup time*): $ST_{(i'\rightarrow i)\,j}$
        - Restricción inamovible para cada artículo
    - Tamaño de cada pedido: q_k
        - Restricción función de la demanda y exigencias del cliente
- Función objetivo: minimización del retraso medio del conjunto de pedidos

$$\left[MIN\right] T_{med} = \frac{1}{z} \cdot \sum_{k=1}^{z} T_k \qquad \text{(Ec. 6.1)}$$

El retraso medio depende del retraso total, de tal manera que minimizar el retraso total implica necesariamente minimizar el retraso medio, pues el número de artículos es el mismo para cada caso.

Se pretende obtener una secuencia de las tareas a realizar que minimice el término T_{med}, es decir, que permita fabricar y servir cada pedido con el menor retraso posible.

7. Heurísticas y metaheurísticas

7.1. Introducción

Los problemas de optimización que implican un gran número finito de alternativas surgen con frecuencia en la industria y la ciencia. En estos problemas, existe un conjunto finito de soluciones, X, y una función real, $f : X \to \mathbb{R}$, y se busca una solución $x^* \in X$, con $f\left(x^*\right) \leq f\left(x\right), \forall x \in X$. Para encontrar la solución optima a un problema de optimización es posible, teóricamente, enumerar las soluciones y evaluar cada una de ellas con respecto al objetivo marcado. Sin embargo, desde un punto de vista práctico, es imposible seguir esta estrategia de resolución, porque el número de combinaciones frecuentemente crece de forma exponencial conforme al tamaño del problema.

Muchos de los trabajos que se han elaborado durante las últimas cinco décadas para el desarrollo de métodos de optimización que no requieran una evaluación exhaustiva de todos los electos del campo de soluciones. Esta búsqueda ha afectado fundamentalmente al campo de la optimización combinatoria y a la capacidad de solucionar problemas complejos de la vida real. Sin embargo, muchos problemas que aparecen en la industria son computacionalmente intratables por su naturaleza, o porque son tan complejos que imposibilitan el uso de algoritmos exactos. En estos casos, los métodos heurísticos se emplean habitualmente para encontrar buenas soluciones, pero no necesariamente garantizan que sean las óptimas. La efectividad de estos métodos depende de su capacidad de adaptarse a cada caso particular y la capacidad para evitar quedar atrapados en óptimos locales. Teniendo en cuenta estas nociones, se ha demostrado que las técnicas heurísticas permiten obtener buenas soluciones a complejos problemas de optimización combinatoria.

7.2. Procedimientos heurísticos

De acuerdo con ANSI/IEEE Std 100-1984 (*American Nacional Standards Institute / Institute of Electrical and Electronic Engineers*), la heurística trata de métodos o algoritmos exploratorios durante la resolución de problemas complejos en los cuales las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final. No se puede garantizar que dichas soluciones sean las óptimas, pero sí razonablemente buenas.

Así pues, la heurística es una serie de procedimientos o estrategias que se supone nos llevan a un destino deseado; pero esto no tiene porque ser siempre cierto. Véase con un

sencillo ejemplo: cuando queremos ir hacia el este, podemos establecer un sistema tal como esperar que salga el sol y caminar en la dirección del nacimiento del sol. Lamentablemente esta estrategia no nos garantiza que podamos esquivar o sortear la presencia de obstáculos durante el camino: muros, ríos, montañas u otros elementos [10].

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

- Cuando el problema o la situación planteada no requieren una solución exacta o precisa, sino sólo aproximada.
- Cuando los datos disponibles son poco fiables.
- Cuando no puede aplicarse un método exacto (*símplex, branch & bound*, etc.) para la resolución del problema, por no existir o porque requiera demasiados recursos (tiempo de cálculo, memoria, presupuesto, etc.).
- 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.

Un esquema del proceso heurístico se representa a continuación:

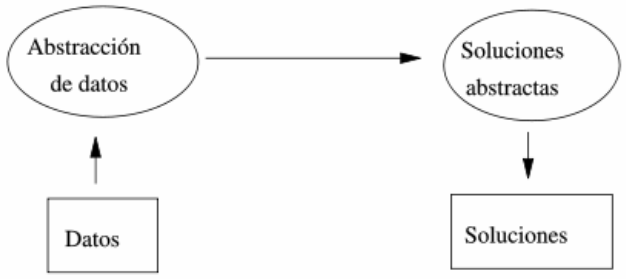


Fig. 7.1. Esquema general de un procedimiento heurístico

El término heurística proviene de la palabra griega "*heuriskein*" relacionado con el concepto de encontrar. Se califica de heurístico a un procedimiento que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad.

Las heurísticas son métodos para resolver problemas complejos en forma aproximada. Además, una ventaja importante es que su complejidad es reducida en comparación con los métodos exactos, por lo que suelen ser más fáciles de entender (por parte de los directivos de las empresas y gente no experta). Son flexibles y tienen como objetivo encontrar soluciones de buena calidad en un tiempo computacional razonable sin mencionar que generalmente ofrecen más de una solución, permitiendo así ampliar las posibilidades de elección.

A pesar de sus ventajas, no cabe duda de que cuando una técnica exacta está disponible debe gozar de preferencia sobre cualquier tipo de heurística, sobre todo cuando los valores económicos manejados sean importantes y el tiempo para resolverlos no esté limitado.

En los últimos años ha habido un crecimiento en el desarrollo de procedimientos heurísticos para resolver problemas combinatorios, debido a la necesidad de disponer de herramientas que permitan ofrecer soluciones rápidas a problemas reales. Los problemas de optimización combinatoria en particular son de gran dificultad debido a su complejidad, ya que crecen exponencialmente con el tamaño del problema, por lo cual se pretende que los métodos heurísticos se acerquen a la solución óptima en un tiempo razonable.

7.2.1. Clasificación de los métodos heurísticos

Existen diferentes tipos de heurísticas, las cuales pueden clasificarse como [4]:

a) Métodos constructivos: Son aquellos 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. Un ejemplo es el GRASP.

b) Métodos de descomposición: 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. Un ejemplo de aplicación en un problema de programación lineal mixta, consistiría en decidir de alguna forma la solución para las variables enteras para luego resolver el problema como un programa lineal.

c) 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.

d) 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.

e) 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.

7.3. Procedimientos metaheurísticos

Una metaheurística es un método heurístico para solucionar una clase muy general de problemas de cómputo combinando los procedimientos de caja negra -generalmente heurísticos- de una manera eficiente. El nombre combina el prefijo griego "*meta*" ("más allá de", aquí en el sentido de "a un nivel superior") y "heurístico" (cuyo origen etimológico ya ha sido comentado en el apartado 7.2).

La metaheurística se aplica generalmente a los problemas para los cuales no hay algoritmo específico satisfactorio o heurístico para su resolución; o cuando no es práctico poner tal método en ejecución. La mayoría de metaheurísticas de uso general se aplican sobre problemas de optimización combinatoria puros o todos aquéllos que puedan adaptarse a esa naturaleza, como por ejemplo solucionar ecuaciones *booleanas*.

Según Osman y Kelly (1995), una definición podría ser la siguiente: "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. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y mecanismos estadísticos".

7.4. Estrategia de secuenciación de la solución inicial

El aspecto fundamental de muchos de los procedimientos heurísticos y metaheurísticos consiste en comenzar desde una solución inicial y generar un conjunto de soluciones vecinas. Según Johnson et al (1989) puede ser ventajoso comenzar con una buena solución

más que con una generada aleatoriamente, es decir, por lo general, se obtiene un mejor resultado final si la solución de partida es buena.

Para el caso que nos ocupa se puede partir, por ejemplo, de cuatro soluciones iniciales diferentes (EDD, SST-EDD, CR1 y CR2) mediante la aplicación de las cuatro heurísticas directas, expuestas a continuación ([5][6]):

7.4.1. EDD (*Earliest Due Date*)

Consiste en ordenar los pedidos, en orden creciente, de acuerdo con la fecha de vencimiento. Para cada pedido *k* se considera la fecha de vencimiento d_k y el tiempo de preparación entre los artículos *i* e *i'* ($ST_{ii'}$).

Dada las características del problema tratado, esta regla favorece los pedidos más prioritarios; sin embargo no aprovecha la ventaja de reducir los tiempos de preparación al procesar, sucesivamente, pedidos de la misma familia de productos.

7.4.2. SST – EDD (*Shortest Setup Time – Earliest Due Date*)

Consiste en ordenar los pedidos por familias de acuerdo con el tiempo de preparación más corto cuando se cambia de una familia a otra (para favorecer un tiempo mínimo de cambios entre familias), y, además, secuenciar los pedidos entre familia por orden creciente de fechas de vencimiento. Para cada pedido *k* se considera la familia b_k, la fecha de vencimiento d_k y el tiempo de preparación $ST_{ii'}$ (dependiente de la familia del producto fabricado en el pedido anterior).

7.4.3. CR1 (*Critical Ratio* 1 o de Cociente)

Índice crítico para cada pieza *i*, basado en De Castro et al (2003), que consiste en calcular el índice de prioridad por medio de la fecha de vencimiento dividida por la suma entre el tiempo de preparación de la familia del pedido *k* (dependiente de la familia del producto fabricado en el pedido anterior) y el tiempo de proceso para el pedido *k*. Los pedidos que tienen el CR1 más pequeños se asignan primero. Para cada pedido *k* se considera la fecha de vencimiento d_k, el tiempo de preparación $ST_{ii'}$ *y el* tiempo de proceso u operación p_i:

$$CR1 = \frac{d_k}{\left(ST_{ii'} + p_{ij}\right)} \qquad \text{(Ec. 7.1)}$$

7.4.4. CR2 (*Critical Ratio* 2 o de Suma ponderada)

Basado en De Castro et al (2003), consiste en asignar los pedidos tomando en cuenta el índice de prioridad para cada pedido *k*, el cual pondera la fecha de vencimiento, el tiempo de preparación de la familia del pedido *k* (dependiente de la familia del producto fabricado en el pedido anterior) y el tiempo de proceso del pedido:

$$CR2 = \alpha d_k + (1-\alpha)\left(p_{ij} + ST_{ii'} \right) \qquad \text{(Ec. 7.2)}$$

7.5. Heurísticas de mejora

A continuación se exponen los principales procedimientos heurísticos y metaheurísticos empleados en la resolución de problemas de optimización combinatoria ([4][6][9]).

7.5.1. Algoritmo Exhaustivo de Descenso (AED)

A partir de la solución en curso se generan y evalúan todos los vecinos. Si el mejor de ellos (en caso de búsqueda de un mínimo el de menor valor) 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.

7.5.2. Algoritmo No Exhaustivo de Descenso (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.

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

La aproximación SA fue desarrollada por Kirkpatrick, Gelatt y Vecchi (1983) y se le conoce como Recocido Simulado dado la analogía entre la simulación del recocido de sólidos y el problema de la resolución de los grandes problemas de optimización combinatoria.

SA parte de una solución inicial determinada por una heurística, y mediante la exploración de su entorno, trata de encontrar la solución óptima. El método consiste de iteraciones, donde se compara la solución en curso con la solución vecina generada aleatoriamente. Si la

solución vecina es mejor que la solución en curso, entonces la solución vecina pasa a ser la solución en curso. En el caso de que la solución vecina sea peor que la solución en curso se puede tomar esta solución vecina como la solución en curso, con una probabilidad que depende de la diferencia entre la solución vecina y la solución en curso y la temperatura *T*.

La regla de generación de la probabilidad de aceptación de la solución vecina que es peor que la solución en curso sigue la distribución Boltzmann:

$$P\left(\partial E\right) = e^{-\frac{\partial E}{kT}} \qquad \text{(Ec. 7.3)}$$

Donde la función tomará valores entre [0,1], y dependerá de la solución vecina, la solución en curso, la constante de Boltzmann *k* y la temperatura *T* en la iteración *n*.

7.5.4. Búsqueda Tabú (TS, *Tabu Search*)

TS es un procedimiento metaheurístico de alto nivel introducido y desarrollado en su forma actual por Fred Glover (1989). Su filosofía se basa en la explotación de diversas estrategias inteligentes para la resolución de problemas, basadas en procedimientos de aprendizaje (Glover y Melián, 2003). A diferencia de otros algoritmos basados en técnicas aleatorias de búsqueda de soluciones cercanas, TS es determinista ya que elimina el azar en sus decisiones y la búsqueda del óptimo está guiada por una estrategia basada en el uso de estructuras de memoria, que guardan soluciones a corto y largo plazo.

El aspecto fundamental del procedimiento consiste en comenzar desde una solución inicial y generar un conjunto de soluciones de su vecindario. De éstas elegir la mejor de las vecinas aún cuando ésta sea peor que la solución en curso. La característica importante de TS es esencialmente la construcción de una lista tabú de movimientos: aquellos movimientos que no son permitidos (movimientos tabú) en la presente iteración. La razón de esta lista es la de excluir los movimientos que pueden hacer regresar a algún punto de una iteración anterior, es decir evitar regresar al mismo óptimo local.

Debido a que la lista tabú puede prohibir soluciones que tengan una calidad superior a la de las soluciones ya conocidas, se hizo uso del concepto “nivel de aspiración”. Primero se comprueba si la solución vecina actual satisface el nivel de aspiración. Si es así, la solución vecina sustituye a la solución en curso y a la mejor. En caso contrario, si dicha solución vecina es igual o peor que la mejor, entonces se compara con los atributos de la lista tabú.

7.5.5. GRASP (*Greedy Randomized Adaptive Search Procedure*)

La técnica de esta metaheurística de tipo iterativo, desarrollada originalmente por Feo y Resende (1989) al estudiar un problema de cobertura de alta complejidad combinatoria, se basa en la combinación de fases de explotación, en las cuales se siguen algoritmos de búsqueda local, con fases de exploración, en los que se usan procedimientos aleatorios para expandir el espacio de búsqueda recorrido.

El algoritmo GRASP mantiene una lista de posibles soluciones a un problema, que se genera aleatoriamente. Se escoge una de las soluciones, y se mejora hasta que no se pueda mejorar más; si es mejor que la mejor almacenada, se actualiza la lista de soluciones almacenadas.

Cada iteración en GRASP consta generalmente de dos pasos: la fase de construcción y el procedimiento de búsqueda local. En el primero se construye una solución tentativa, que luego es mejorada mediante un procedimiento de intercambio hasta que se llega a un óptimo local. En la fase constructiva, GRASP toma en cuenta la función objetivo con la intención de que al término de la iteración se cuente con una solución de alta calidad, sobre la cual se efectúa una mejora o Fase II.

7.5.6. Algoritmo Genético (GA, *Genetic Algorithm*)

El GA es un algoritmo de búsqueda que explora un espacio de solución que simula procesos en un sistema natural hacia la evolución, específicamente aquellos que siguen el principio de la supervivencia en función de la adaptabilidad (Adenso Díaz et al, 1996). Fue desarrollado por Holland (1975), y se distingue muy claramente de todos los anteriores, básicamente por el hecho de que en cada iteración se tiene un conjunto de soluciones, o población en curso y no una única solución en curso. Las soluciones sucesoras se obtienen a partir de parejas constituidas con los elementos de la población y no mediante la transformación de la solución en curso.

Cuando Holland se enfrentó a los algoritmos genéticos, los objetivos de su investigación fueron dos:

- Imitar los procesos adaptativos de los sistemas naturales
- Diseñar sistemas artificiales (normalmente programas) que retengan los mecanismos importantes de los sistemas naturales.

7.5.7. Búsqueda Dispersa (SS, *Scatter Search*)

La Búsqueda Dispersa es un procedimiento metaheurístico basado en estrategias para combinar reglas de decisión, así como en la combinación de restricciones. El método SS opera sobre un conjunto de soluciones, llamado conjunto de referencia, combinando éstas para crear nuevas soluciones de modo que mejoren a las que las originaron. En este sentido decimos que es un método evolutivo.

Sin embargo, a diferencia de otros métodos evolutivos, como los algoritmos genéticos, SS no está fundamentado en la aleatorización sobre un conjunto relativamente grande de soluciones sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño, pues SS se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones puede ser utilizado mediante la combinación de éstas en lugar de aisladamente.

7.5.8. Clasificación de los métodos expuestos

El siguiente cuadro muestra la clasificación de los diversos métodos:

HEURÍSTICAS	METAHEURÍSTICAS
EDD (*Earliest Due Date*)	Búsqueda Tabú (TS, *Tabu Search*)
SST-EDD (*Shortest Setup Time – Earliest Due Date*)	GRASP (*Greedy Randomized Adaptative Search Procedure*)
CR1 (*Critial Ratio 1 o Cociente*)	Recocido Simulado (SA, *Simulated Annealing*)
CR2 (*Critical Ratio 2 o de Suma ponderada*)	Algoritmo genético (GA, *Genetic Algorithm*)
Algoritmo Exhaustivo de Descenso (AED)	Búsqueda dispersa (SS, *Scatter Search*)
Algoritmo No Exhaustivo de Descenso (ANED)	

Tabla 7.1. Cuadro resumen de clasificación de los métodos expuestos

8. **Procedimiento de resolución propuesto**

8.1. Introducción al método GRASP

Los métodos GRASP ([13][14]) fueron desarrollados al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimiento de conjuntos (Feo y Resende, 1989). El término GRASP fue introducido por Feo y Resende (1995) como una nueva técnica metaheurística de propósito general. GRASP es un procedimiento de multi-arranque en donde cada paso consiste en una fase de construcción y una de mejora. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local. La mejor de todas las soluciones examinadas se guarda como resultado final.

La palabra GRASP proviene de las siglas de *Greedy Randomized Adaptive Search Procedure*, que en castellano sería aproximadamente: Procedimientos de Búsqueda basados en funciones ”Voraces” Aleatorizadas que se Adaptan. Veamos los elementos de este procedimiento.

En la fase de construcción se construye iterativamente una solución posible, considerando un elemento en cada paso. En cada iteración la elección del próximo elemento para ser añadido a la solución parcial viene determinada por una función *greedy*. Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir el mejor. Nótese que esta medida es miope en el sentido que no tiene en cuenta qué ocurrirá en iteraciones sucesivas al realizar una elección, sino únicamente en esta iteración. Se dice que la heurística *greedy* se adapta porque en cada iteración se actualizan los beneficios obtenidos al añadir el elemento seleccionado a la solución parcial. Es decir, la evaluación al añadir un determinado elemento a la solución en la iteración j no coincidirá necesariamente con la que se tenga en la iteración $j + 1$.

La heurística es aleatorizada porque no selecciona el mejor candidato según la función *greedy* adaptada, sino que, con el objeto de diversificar y no repetir soluciones en dos construcciones diferentes, se construye una lista con los mejores candidatos de entre los que se toma uno al azar.

Al igual que ocurre en muchos métodos, las soluciones generadas por la fase de construcción de GRASP no suelen ser óptimos locales. Dado que la fase inicial no garantiza la optimalidad local respecto a la estructura de entorno en la que se esté trabajando (cabe notar que hay selecciones aleatorias), se aplica un procedimiento de búsqueda local como

postproceso para mejorar la solución obtenida. En la fase de mejora se suele emplear un procedimiento de intercambio simple con el objeto de no emplear mucho tiempo en esta mejora. Nótese que el GRASP se basa en realizar múltiples iteraciones y quedarse con la mejor, por lo que no es especialmente beneficioso para el método el detenerse demasiado en mejorar una solución dada.

PROCEDIMIENTO ITERATIVO:

GREEDY-ALEATORIZADO-ADAPTATIVO + BUSQUEDA LOCAL

8.2. Expresión algorítmica del GRASP

El siguiente esquema muestra el funcionamiento global del algoritmo GRASP:

Repetir Mientras (No se satisfaga la Condición de parada)

Fase Construcción de la solución *Greedy* Aleatorizada

Seleccionar una lista de elementos candidatos.

Considerar una Lista Restringida de los mejores Candidatos.

Seleccionar un elemento de la Lista Restringida.

Fase de Mejora (Búsqueda Local)

Realizar un proceso de búsqueda local a partir de la solución construida hasta que no se pueda mejorar más.

Actualización

Si la solución obtenida mejora a la mejor almacenada, actualizarla.

Devolver la mejor solución

Fin GRASP

El realizar muchas iteraciones GRASP es una forma de realizar un muestreo del espacio de soluciones. Basándonos en las observaciones empíricas, se ve que la distribución de la muestra generalmente tiene un valor en promedio que es inferior al obtenido por un procedimiento determinista, sin embargo, la mejor de las soluciones encontradas generalmente supera a la del procedimiento determinista con una alta probabilidad.

Las implementaciones GRASP generalmente son robustas en el sentido de que es difícil encontrar ejemplos patológicos en donde el método funcione arbitrariamente mal.

Algunas de las sugerencias de ciertos autores a la hora de implementar el procedimiento son:

- Se puede incluir una fase previa a la de construcción: una fase determinista con el objetivo de ahorrar esfuerzo a la fase siguiente.

- Si se conoce que ciertas subestructuras forman parte de una solución óptima, éstas pueden ser el punto de partida de la fase constructiva.

Tal y como señalan Feo y Resende (1989), una de las características más relevantes de GRASP es su sencillez y facilidad de implementación. Basta con fijar el tamaño de la lista de candidatos y el número de iteraciones para determinar completamente el procedimiento. De esta forma se pueden concentrar los esfuerzos en diseñar estructuras de datos para optimizar la eficiencia del código y proporcionar una gran rapidez al algoritmo, dado que éste es uno de los objetivos principales del método. El enorme éxito de este método se puede constatar en la gran cantidad de aplicaciones que han aparecido en los últimos años.

8.3. Fase de construcción de la Solución Inicial en GRASP

El pseudo – código que se muestra a continuación ilustra el procedimiento algorítmico GRASP para la minimización, en el cual se realizan maxitr GRASP iteraciones.

procedimiento *grasp*$\left(f(\cdot), g(\cdot), maxitr, x^*\right)$

1 $x^* = x$
2 **para** k=1,2,...,maxitr **hacer**
3 *construct*$\left(g(\cdot), a, x\right)$
4 *local*$\left(f(\cdot), x\right)$
5 **si** $f(x) < f(x^*)$ **hacer**
6 $x^* = x$
7 **fin si**
8 **fin para**

fin *grasp*

En la fase de construcción, se “construye” una solución factible iterativamente: un elemento (solución) por cada iteración. A cada iteración, la elección del siguiente elemento que se añadirá se determina ordenando todos los candidatos (por ejemplo, aquéllos que pueden ser añadidos a la solución) en una lista C de candidatos con respecto a la función devoradora o

greedy $g : C \to \mathbb{R}$. Esta función mide el beneficio (miope) de seleccionar cada uno de los elementos. La heurística es adaptativa porque los beneficios asociados a cada elemento son actualizados a cada iteración de la fase de construcción para reflejar los cambios causados por la selección del elemento previo. El componente probabilístico del GRASP se caracteriza por escoger aleatoriamente uno de los mejores candidatos de la lista, pero no necesariamente "el mejor" candidato. La lista de los mejores candidatos se denomina "Lista de Candidatos Restringidos" (LCR) o "*Restricted Candidate List*" (RCL). Esta técnica de selección tiene en cuenta diferentes soluciones obtenidas en cada iteración GRASP, pero no necesariamente compromete el poder del componente *greedy* adaptativo del método. Imaginemos que $\alpha \in [0,1]$ es un parámetro conocido. El pseudo – código que se muestra a continuación describe la fase de construcción de un algoritmo GRASP básico.

procedimiento *construct*$(g(\cdot), a, x)$
1 x = 0
2 *Inicializar juego de candidatos C*
3 **mientras** $C \neq 0$ **hacer**
4 $\underline{s} = \min\{(g(t) | t \in C)\}$
5 $\bar{s} = \max\{(g(t) | t \in C)\}$
6 $RCL = \{s \in C \,|\, g(s) \leq \underline{s} + \alpha(\bar{s} - \underline{s})\}$
7 *Seleccionar s de RCL aleatoriamente*
8 $x = x \cup \{s\}$
9 *Actualizar el juego de candidatos C*
10 **fin mientras**
fin *construct*

El pseudo – código muestra que el parámetro α controla la cantidad de "voracidad" y aleatoriedad en el algoritmo. Un valor de $\alpha = 0$ corresponde a un procedimiento de construcción voraz, mientras que un valor de $\alpha = 1$ produce una construcción totalmente aleatoria.

En el caso que nos ocupa, α toma el valor de cero.

8.4. Fase de Búsqueda Local

Como es el caso de numerosos métodos determinísticos, las soluciones generadas por una construcción GRASP no puede garantizarse que sean óptimo local respecto a simples definiciones del vecindario. Por lo tanto, es casi siempre beneficioso aplicar una búsqueda local para intentar mejorar cada solución construida. Un algoritmo de búsqueda local trabaja

de una forma iterativa, reemplazando sucesivamente la solución actual por otra mejor perteneciente al vecindario de esa solución actual. Este proceso termina cuando no se encuentra una solución mejor en el vecindario. La estructura del vecindario *N* para un problema *P* relaciona una solución del problema con un subgrupo de soluciones *N(s)*. Una solución *s* se dice que es localmente óptima si no hay una solución mejor en el espacio *N(s)*. La clave del éxito para un algoritmo de búsqueda local consiste en una acertada elección de la estructura vecinal, de técnicas eficientes de búsqueda en el vecindario y de la solución inicial.

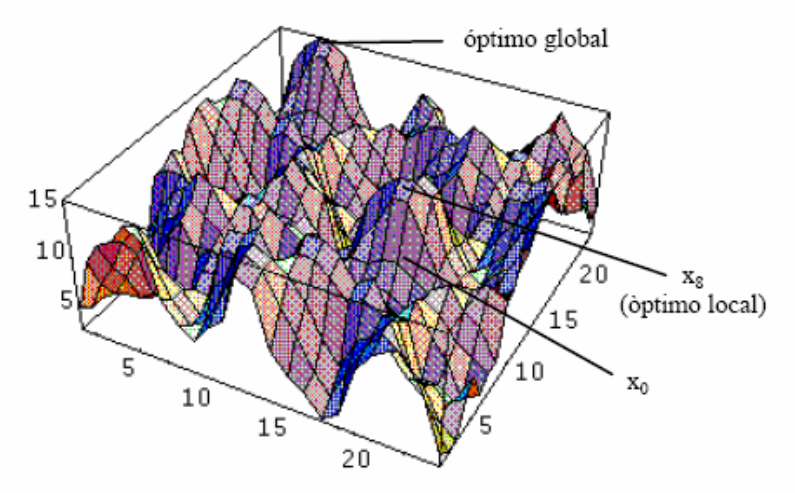


Fig. 8.1. Representación gráfica de los óptimos global y local [9]

Mientras parece que los procedimientos de optimización local pueden requerir un tiempo exponencial de resolución desde un punto de partida arbitrario, empíricamente su eficiencia mejora sensiblemente tan buen punto como mejora la solución inicial. A través del uso de estructuras de datos adaptadas a las necesidades y una implementación cuidadosa, se puede desarrollar una fase de construcción que genere buenas soluciones iniciales para una búsqueda local igual de eficiente. El resultado es que frecuentemente algunas soluciones GRASP están generadas en el mismo período de tiempo requerido para el procedimiento de optimización local para converger desde un único inicio aleatorio. Además, la mejor de esas soluciones GRASP es por lo general significativamente mejor que la solución obtenida desde un punto de inicio aleatorio. El siguiente pseudo – código describe el procedimiento básico de búsqueda local.

procedimiento *local*$(f(\cdot),N(\cdot),x)$

1 $H = \{y \in N(x) \mid f(y) < f(x)\}$

2 **mientras** $|H| > 0$ **hacer**

3 *Select* $x \in H$

4 $H = \{y \in N(x) \mid f(y) < f(x)\}$

5 **fin mientras**

fin *local*

Resulta difícil analizar formalmente la calidad de las soluciones encontradas usando la metodología GRASP. Sin embargo, hay una justificación intuitiva por la que se ve la GRASP como una técnica repetitiva de muestreo. Cada iteración en GRASP produce una solución muestra procedente de una distribución desconocida de todos los resultados obtenibles. La media y la varianza de la distribución son función de la naturaleza restrictiva de la lista de candidatos. Por ejemplo, si la cardinalidad de la LRC se limita a uno, sólo se generará una solución y la varianza de la distribución será cero. Dada una función voraz efectiva, la solución media en este caso debería ser buena, pero probablemente subóptima. Si se impone un límite de cardinalidad menos restrictivo, se generarán soluciones muy diversas que implicarán una elevada varianza. Ya que la función *greedy* es más intermedia en este caso, el valor de la solución media debería descender. Intuitivamente, sin embargo, por orden estadístico y por el hecho de que las muestras se generan aleatoriamente, el mejor valor encontrado debería conducir al valor medio. De hecho, la mayoría de las mejores soluciones encontradas son óptimas.

Una característica especialmente atractiva del GRASP es la facilidad con que puede ser implementado. Sólo se requiere fijar y ajustar unos cuantos parámetros, y por lo tanto el desarrollo puede centrarse en la implementación eficiente de estructuras de datos para asegurar unas iteraciones GRASP rápidas. Finalmente, GRASP puede ser implementado en paralelo. Cada procesador puede ser inicializado con su propia copia del procedimiento, los datos de ejemplo, y número de secuencia independiente aleatorio. Las iteraciones GRASP se ejecutan en paralelo con solo una variable global, requerida para almacenar la mejor solución encontrada por todos los procesadores.

8.5. Diagrama de flujo del procedimiento

A continuación se muestran diversos diagramas de flujo del algoritmo GRASP, divididos en sus dos fases: Fase I de Construcción y Fase II de Búsqueda local.

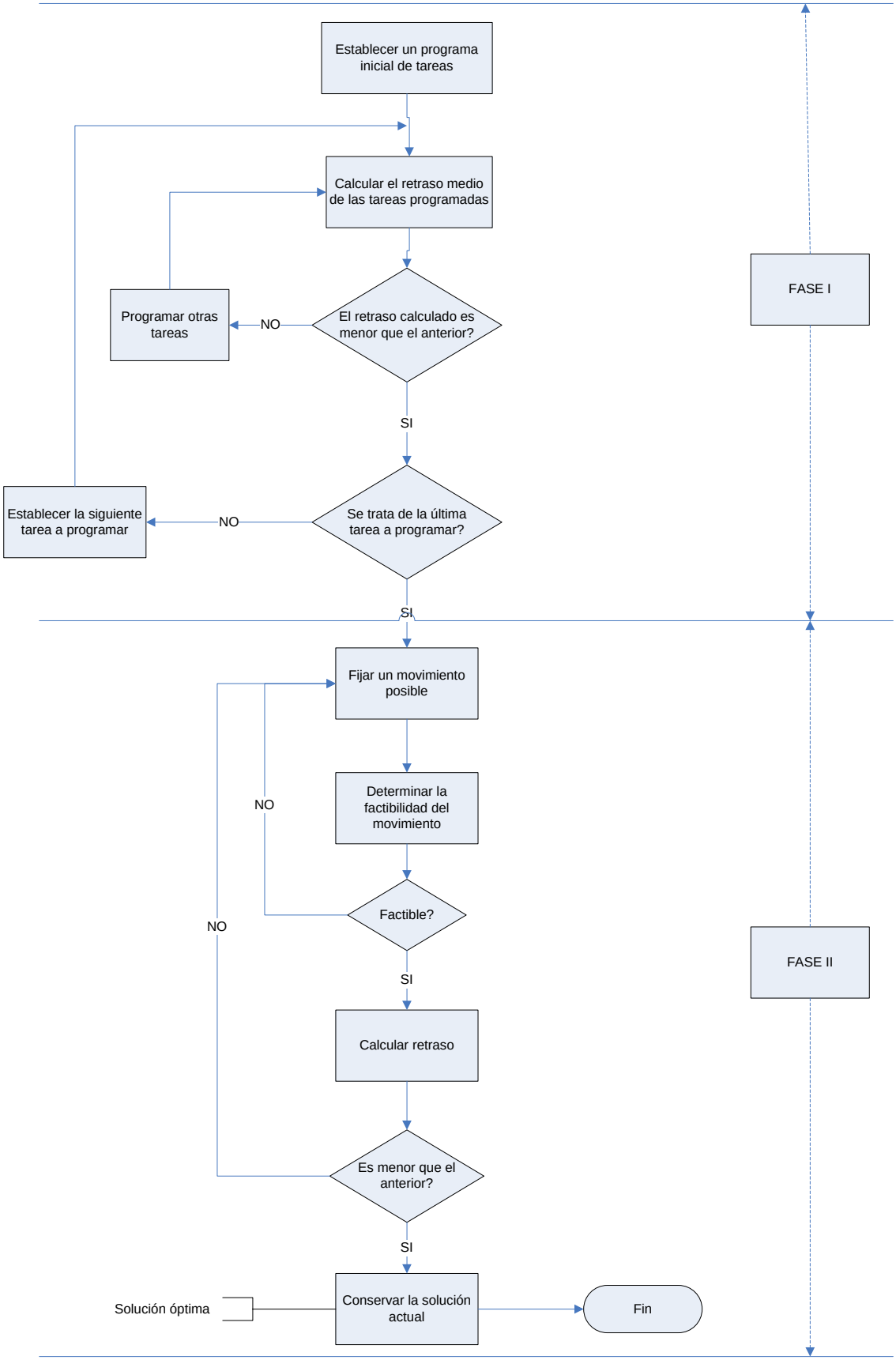


Fig. 8.3. Diagrama de flujo del algoritmo GRASP

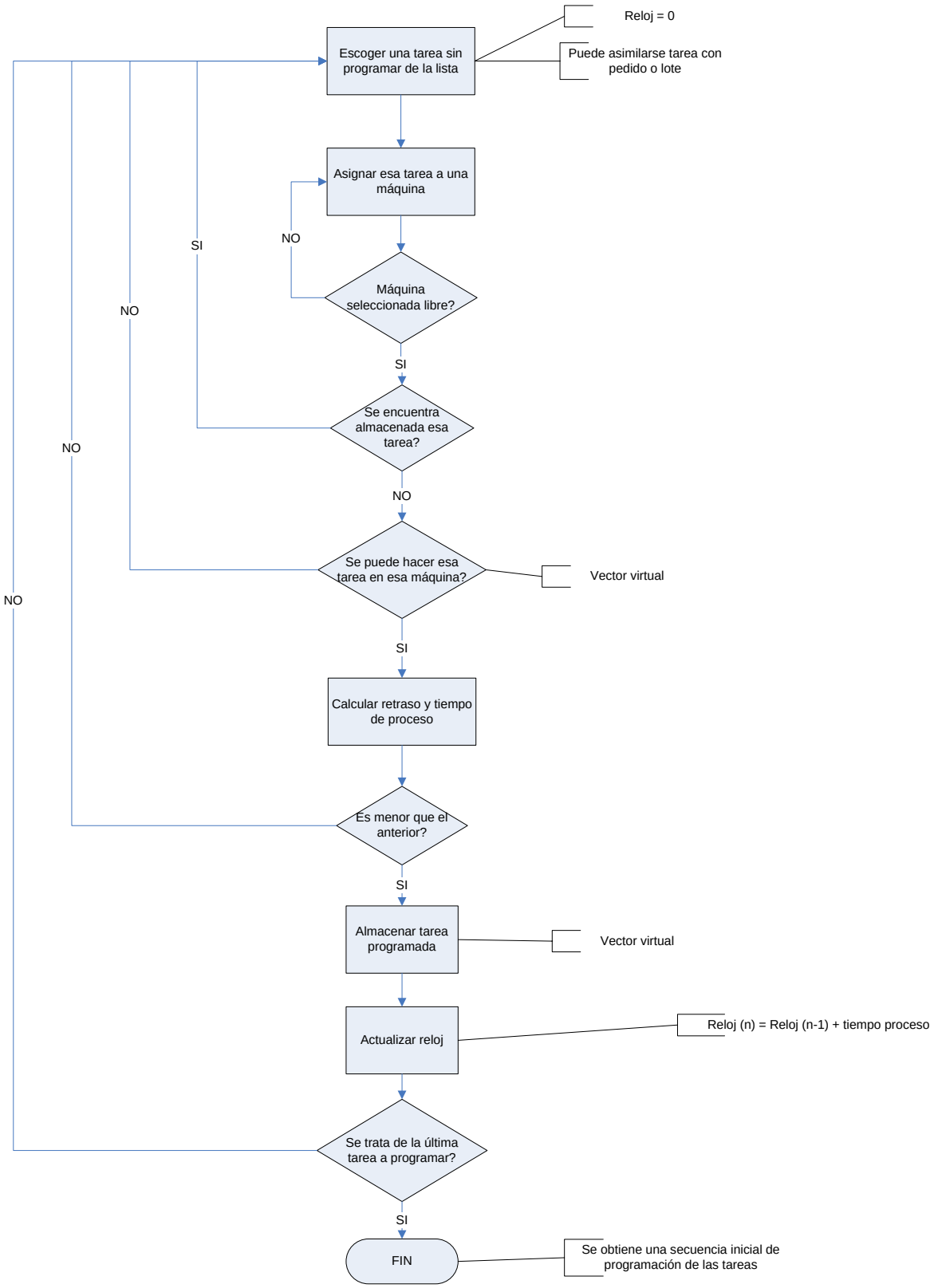


Fig. 8.4. Diagrama de flujo de la Fase I del algoritmo GRASP

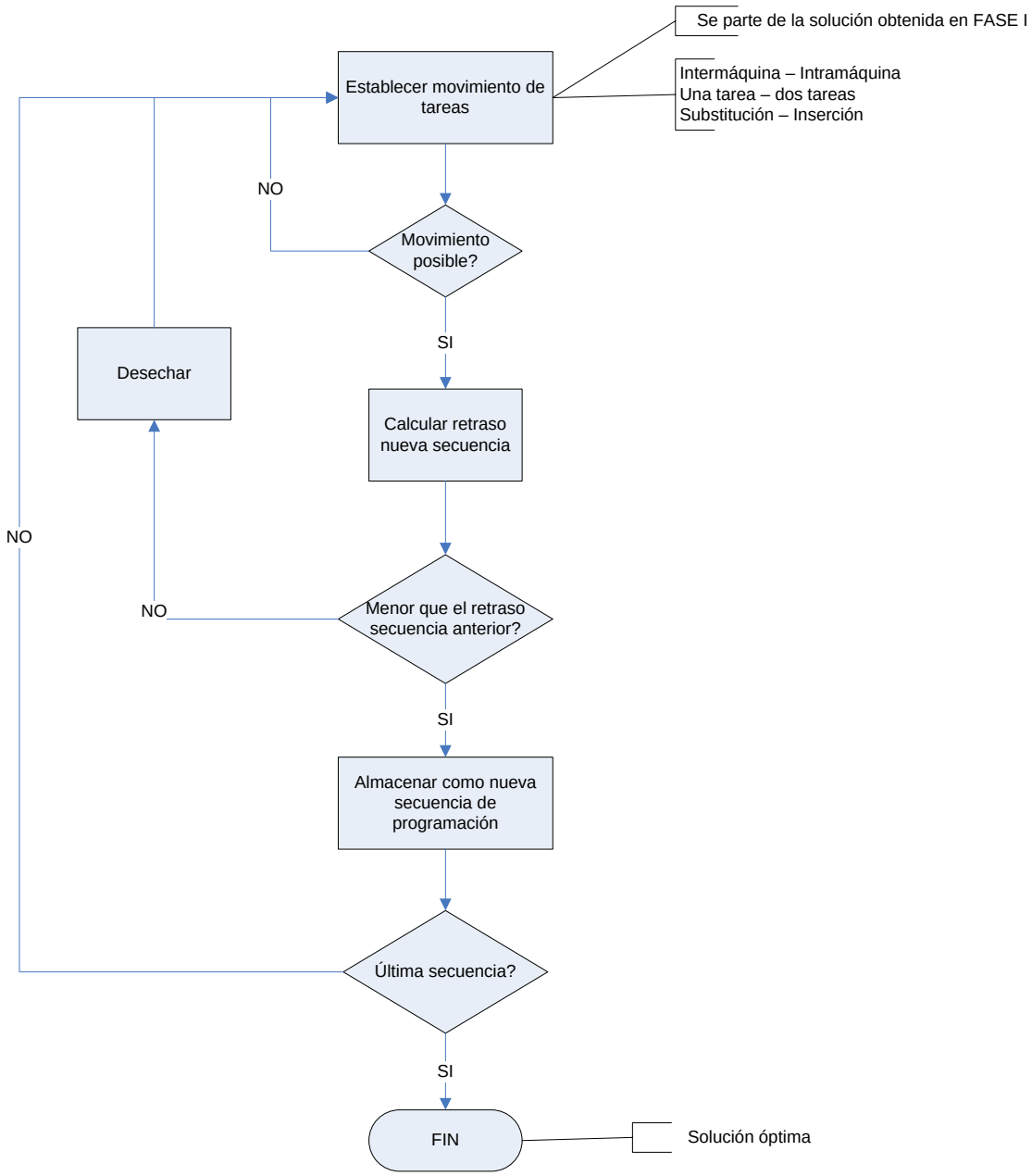


Fig. 8.5. Diagrama de flujo de la Fase II del algoritmo GRASP

8.6. Búsqueda de los vecinos

El orden en el que se busca un vecindario puede reducir el tiempo requerido para verificar la factibilidad si éste mejora la eficiencia con la que se actualizan los tiempos *forward slacks* y los tiempos de inicio. La siguiente clasificación es utilizada en nuestra implementación:

8.6.1. Reubicación de tareas programadas

Se asume que la secuencia actual sea $(0,...,n+1)$, donde 0 y $n+1$ son trabajos ficticios que representan el primer y el último trabajo en la secuencia, respectivamente. La reubicación implica la inserción de la secuencia $(i_1,...,i_2)$ entre las tareas $(j, j+1)$, lo cual puede suceder en la misma máquina o en máquinas diferentes.

1. Reubicación de tareas en la misma máquina

Se selecciona una secuencia de tareas $(i_1,...,i_2)$ en una máquina *m*, que comienza con el primer trabajo $i_1 = 1$ (*outer loop*). Tras fijar la secuencia, los puntos de intersección $(j, j+1)$ en la misma máquina *m* se consideran consecutivos (*inner loop*). Cuando la localización del punto de intersección se encuentra en una posición anterior a la secuencia actual, se habla de *backward relocation*; cuando el punto de intersección se encuentra en una posición adelantada respecto a la de la secuencia actual, se denomina *forward relocation*.

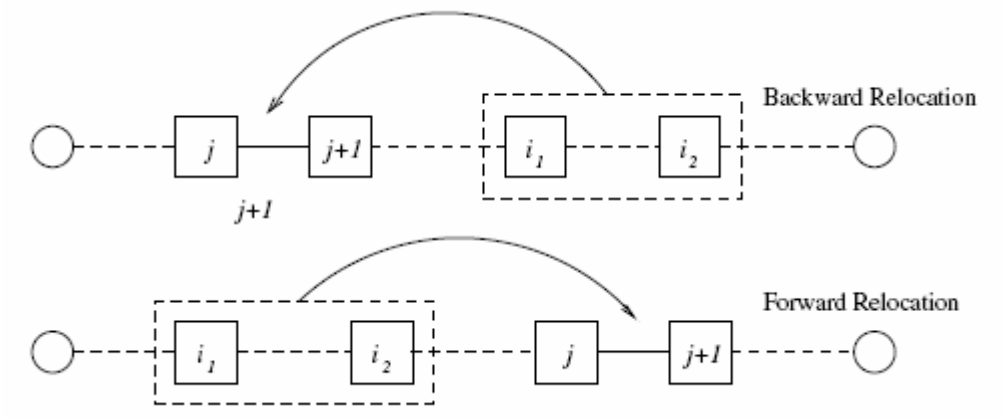


Fig. 8.6. Reubicación hacia atrás y hacia adelante (*backward* y *forward relocation*) intramáquina

En el primer caso (*backward relocation*), las tareas seleccionadas se reubican en posiciones de la secuencia de programación más retrasadas en el tiempo. En el ejemplo propuesto, se selecciona el conjunto formado por las tareas i_1 e i_2 y se emplazan entre las tareas *j* y *(j+1)*, situadas en posiciones anteriores, pudiéndose dar el caso de llegar a situarlas al inicio de la secuencia. En el segundo caso (*forward relocation*), las tareas escogidas se sitúan en posiciones más adelantadas respecto a la secuencia establecida inicialmente. En el ejemplo, los trabajos i_1 e i_2 se emplazan entre *j* y *(j+1)*, que se ejecutaban a posteriori, pudiendo llegar, incluso, a ubicarse al final de la secuencia.

2. Reubicación de tareas de una máquina a otra

La secuencia de tareas a ser reubicadas desde la máquina inicial k_0 es $(i_1,...i_2)$, y el punto de intersección en la máquina k_d de destino es $(j, j+1)$, como muestra la siguiente figura. En la *outer loop*, se selecciona una secuencia para ser reubicada en orden inverso empezando por el último trabajo de k_0. En el *inner loop*, los puntos de intersección se consideran secuencialmente en el orden inverso a la secuencia de tareas de k_d. El destino se escoge como la siguiente máquina en el esquema numerado. El *inner loop* se ejecuta en sucesivas máquinas hasta que todos los candidatos se han tenido en cuenta. El *outer loop* es incrementa para asegurar que se han tenido en cuenta todas las combinaciones. El tamaño de las secuencias son uno, dos y tres, respectivamente.

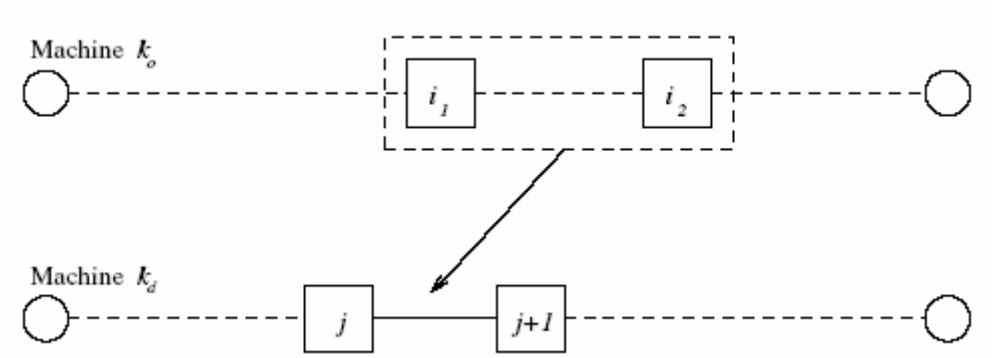


Fig. 8.7. Reubicación de tareas intermáquina

8.6.2. Intercambio de tareas programadas

Existen dos casos para el intercambio de tareas *i* y *j*.

1. En la misma máquina

Empezando por la máquina 1, las tareas se escogen una cada vez en el orden inverso al de la secuencia (*outer loop*). En el *inner loop*, las tareas que preceden a la seleccionada se consideran secuencialmente en el orden inverso de la secuencia para un posible intercambio. El proceso se repite para todos los trabajos y todas las máquinas.

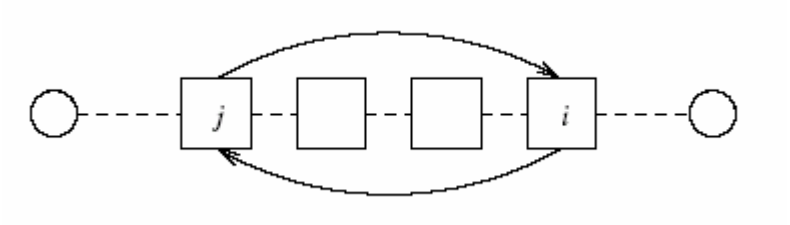


Fig. 8.8. Intercambio de tareas intramáquina

2. Dos máquinas diferentes

El orden de búsqueda es similar a lo definido para la reubicación de tareas de una máquina a otra. Las diferencias son:

(i) Sólo se tienen en cuenta las operaciones de una tarea.

(ii) Los puntos de intersección se reemplazan por las tareas intercambiadas.

(iii) Los posibles $\left(\frac{m}{2}\right)$ pares de máquinas, antes que los posibles $m(m-1)$ pares ordenados, son candidatos para los intercambios. Después de fijar la tarea *i* en la máquina k_0 en el *outer loop*, se selecciona la tarea *j* en el orden inverso al de los trabajos programados en la máquina k_d en el *inner loop*.

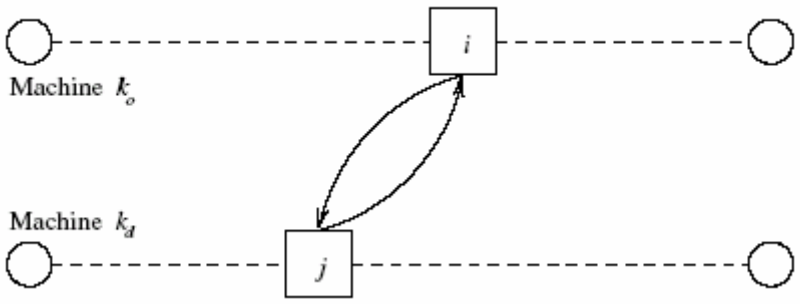


Fig. 8.9. Intercambio de tareas intermáquina

8.7. **Reencadenamiento de trayectorias (*Path Relinking*, PR)**

Una de las principales desventajas del GRASP puro es su falta de estructuras de memoria. Las iteraciones de GRASP son independientes y no utilizan las observaciones hechas durante iteraciones previas. Un remedio para esto es el uso del reencadenamiento de trayectorias con GRASP. El PR fue propuesto originalmente por Glover (1997) como una forma de explorar las trayectorias entre soluciones élite obtenidas por búsqueda tabú o búsqueda dispersa. Usando una o más soluciones élite, se exploran las trayectorias en el espacio de soluciones que conducen a otras soluciones élite para buscar mejores soluciones. Para generar trayectorias, los movimientos se seleccionan para introducir atributos en la solución actual que estén presentes en la solución élite guía.

El reencadenamiento de trayectorias en el contexto de GRASP fue introducido por Laguna y Martí (2000). Desde entonces han aparecido numerosas extensiones, mejoras, y aplicaciones exitosas. Ha sido usado como un esquema de intensificación, en el que las soluciones generadas en cada iteración de GRASP se reencadenan con una o más soluciones de un conjunto élite de soluciones, o como en una fase de post-optimización, donde se reencadenan pares de soluciones del conjunto élite.

Consideremos dos soluciones x_s y x_t en las cuales queremos aplicar reencadenamiento de trayectorias desde x_s hacia x_t. A continuación se ilustra el procedimiento de reencadenamiento de trayectorias con su pseudo – código.

procedimiento *PR*
Se requieren: x_s,x_t
1 *Calcular la diferencia simetrica* $\Delta(x_s,x_t)$
2 $x = x_s$
3 $f^* = \min\{f(x_s),f(x_t)\}$
4 $x^* = \arg\min\{f(x_s),f(x_t)\}$
5 **mientras** $\Delta(x_s,x_t) \neq 0$ **hacer**
6 $m^* = \arg\min\{f(x \oplus m), \forall m \in \Delta(x,x_t)\}$
7 $\Delta(x \oplus m^*,x_t) = \Delta(x,x_t) \setminus \{m^*\}$
8 $x = x \oplus m^*$
8 **si** $f(x) < f^*$ **entonces**
9 $f^* = f(x)$
10 $x^* = x$
11 **fin si**
10 **fin mientras**
fin *PR*;

El procedimiento se inicia calculando la diferencia simétrica $\Delta(x_s, x_t)$ entre las dos soluciones, i.e. el conjunto de movimientos necesarios para alcanzar x_t desde x_s. Se genera entonces una trayectoria de soluciones encadenando a x_s con x_t. El algoritmo devuelve la mejor solución encontrada en esta trayectoria. En cada paso, el procedimiento examina todos los movimientos $m \in \Delta(x_s, x_t)$ desde la solución actual x y elige aquél que resulta en la solución menos costosa, por ejemplo, aquél que minimiza $f(x \oplus m)$, donde $x \oplus m$ es la solución resultante de aplicar el movimiento *m* a la solución *x*. Se efectúa el mejor movimiento m^* produciendo $x \oplus m^*$ y el movimiento m^* se elimina de la diferencia simétrica de $\Delta(x \oplus m^*, x_t)$. En caso necesario, la mejor solución x^* se actualiza. El procedimiento termina cuando se alcanza x_t, i.e. cuando $\Delta(x, x_t) = 0$.

La Fig. 8.10 ilustra el reencadenamiento de trayectorias. En el grafo representado, los nodos corresponden a soluciones, y los arcos corresponden a movimientos que permiten que una solución se alcance a partir de otra. Supóngase que dos soluciones, A y D, se reencadenan. Sea A la solución inicial y D la solución meta, y supóngase que la diferencia simétrica $\Delta(A, D) = 3$. Existen tres posibles movimientos partiendo de A. Se elige el mejor movimiento, el cual produce la solución B. En este punto, la diferencia simétrica $\Delta(B, D) = 2$, y por lo tanto existen dos movimientos posibles. De nuevo, se elige el mejor movimiento, el cual produce la solución C. Finalmente, en este punto hay un solo movimiento posible, el cual conduce a la solución meta D. Este esquema de reencadenamiento de trayectorias genera una "trayectoria" $A \to B \to C \to D$, la cual puede ahora ser evaluada.

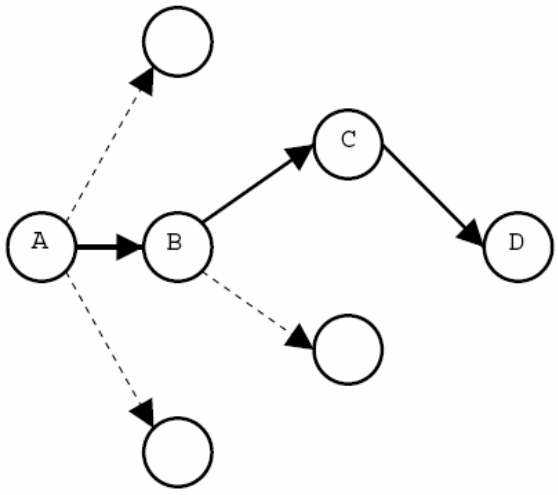


Fig. 8.10. Reencadenamiento de trayectorias

9. **Desarrollo del algoritmo**

Es fundamental definir los *inputs* o entradas que serán solicitados por el programa. Estos son los siguientes:

- Situación inicial de las máquinas, es decir, el artículo para el que están preparadas para fabricar. Determina si es necesario un cambio de formato (con el consiguiente tiempo de preparación) en esa máquina o no
- Matriz de tiempos de preparación para cada máquina al pasar de fabricar un artículo a otro diferente
- Matriz de tiempos de proceso unitarios de cada artículo en cada máquina
- Datos de cada pedido recibido, como son:
    - Artículo que contienen
    - Tamaño de lote
    - Fecha de entrega

Los *outputs* o resultados de salida proporcionados por el software serán, como mínimo, los siguientes:

- La secuencia de lotes a fabricar y el artículo asociado a dicho lote
- El retraso de cada lote y el retraso total y medio

De manera opcional, se puede proporcionar el tiempo de reloj de salida de cada lote. Es de suponer que esa secuencia es la que da lugar al mínimo retraso.

9.1. Datos

El taller consta de m máquinas diferentes[1] en paralelo, capaces de fabricar i artículos diferentes **bajo pedido**. Cada artículo lleva asociados unos tiempos de preparación $tp_{(i'\to i)j}$ (como consecuencia de adaptar la máquina j a la fabricación del artículo i tras fabricar el artículo i') y unos tiempos de operación p_{ij}, diferentes en cada máquina para un mismo artículo.

Cada pedido consta de k lotes, cada uno de los cuales contiene q_k unidades de un mismo artículo i y una fecha de entrega máxima de d_k días.

Los parámetros asociados al problema (agrupados en dos categorías: Constantes (C) y Variables (V)) son los siguientes:

- (V) Número de máquinas: m
    - Conjunto de máquinas $j = \{1,...,m\} \in M$
- (V) Estado inicial de las máquinas
- (C) Tiempo de preparación (cambio entre productos para cada máquina): $tp_{(i'\to i)j}$
- (C) Tiempo de proceso unitario de cada producto en cada máquina: p_{ij}
- (V) Número de artículos: n
    - Conjunto de artículos: $i = \{1,...,n\} \in N$
- (V) Número de pedidos o lotes: z

[1] Esto es, no todas las máquinas son capaces de fabricar cualquier artículo, existiendo además diferencia en los tiempos de producción de un mismo artículo en las diferentes máquinas

- Conjunto de pedidos: $k = \{1,...,z\} \in Z$
- Para cada pedido, se establece:
  - (V) Tamaño de lote o cantidad de cada artículo: q_k
  - (V) Producto que contiene cada lote[2]: p_k
  - (V) Fecha de entrega máxima o período de vencimiento: d_k

- (V) Secuencia de programación factible para cada máquina: σ_j

Por consiguiente, el tiempo de proceso total es el producto del tiempo unitario de proceso por el tamaño de lote:

$$P_{ij} = p_{ij} \cdot q_k \qquad \text{(Ec. 9.1)}$$

Según la secuencia escogida, el instante en que un pedido k sale del taller, c_k, viene dado por:

$$c_k = r_k + w_k + P_{ij} + tp_{ij} \qquad \text{(Ec. 9.2)}$$

donde: r_k: tiempo de llegada del pedido *k* a la máquina

w_k: tiempo de espera del pedido *k* hasta que puede ser procesada

[2] Se considera que cada lote se compone únicamente de un solo tipo de artículo.

y su retraso T_k , viene dado por:

$$T_k = \max\{0, c_k - d_k\} \qquad \text{(Ec. 9.3)}$$

10. Procedimiento de resolución propuesto

10.1. Descripción del procedimiento

Como se ha comentado anteriormente, la heurística GRASP se compone de dos fases.

10.1.1. Fase I: Algoritmo *Greedy*

En primer lugar, es necesario obtener una secuencia de partida que facilite los procedimientos de que consta la segunda parte de la heurística. Es lo que conoce como Fase I del GRASP, que se asemeja a un algoritmo *greedy*.

Para ello, inicialmente, es preciso establecer una "Lista Restringida de Candidatos" (LRC), formada por los elementos sobre los que se aplicará el algoritmo *greedy*, y sujetos a ciertas restricciones.

En el caso tratado, la LRC está configurada por los pedidos que deben asignarse a las m máquinas disponibles a fin de obtener una secuencia de fabricación con un retraso aceptable. La restricción a la que están sujetos es la imposibilidad de ciertas máquinas de fabricar el artículo de que se componga un pedido.

En el punto de partida, la LRC está compuesta por la totalidad de los pedidos a fabricar. A cada iteración[3] del algoritmo *greedy*, la LRC se actualiza, eliminando de la misma aquellos candidatos convenientemente asignados a una máquina.

El criterio empleado para determinar qué candidato se asigna a cada máquina es el retraso total, es decir, la diferencia entre el momento en que efectivamente se finaliza un pedido y cuando se debe finalizar (fecha de entrega). En caso que coincidan dos o más pedidos en la misma máquina según este criterio, se empleará el criterio EDD (menor fecha de entrega).

A cada iteración, del conjunto de candidatos de la LRC evaluados, se asignará a una máquina aquél que presente el menor valor resultado de la diferencia entre el momento de

[3] Asimilamos iteración a cada momento en que se produce un acontecimiento en el sistema, como por ejemplo la entrada o salida de pedidos del mismo

salida del sistema y fecha de entrega. En caso de coincidencia entre máquinas (que se obtenga el mismo retraso en dos o más máquinas diferentes), se asignará a aquella máquina que presente una menor fecha de entrega.

10.1.2. Fase II: Búsqueda y evaluación de vecinos

Una vez finalizado el algoritmo *greedy*, esto es, cuando ya han sido asignados todos los elementos de la LRC, se procede a la Fase II del procedimiento GRASP: la búsqueda y evaluación de vecinos.

Tomando como punto de partida la secuencia de programación resultante de la Fase I, se aplican sobre ella diversos métodos preestablecidos que dan lugar a una nueva solución derivada de la primera (vecino).

El objetivo consiste en evaluarla y efectuar una comparación entre la solución inicial y la actual. Si el retraso medio de la nueva solución es menor, la solución final se actualiza a la actual; en caso contrario, se procede a aplicar un nuevo método u otro diferente para obtener nuevos vecinos.

El resultado final será aquella secuencia de operaciones que dé lugar al menor retraso medio de entre todas las posibles soluciones (no necesariamente la solución óptima).

Los métodos de búsqueda y evaluación de vecinos son los siguientes:

- Método principal – Sustituir Mayor por Menor: este método es una sustitución intramáquina: reemplaza inequívocamente el pedido que tiene mayor retraso por el que tiene mayor adelanto respecto a su fecha de entrega, recalculando entonces la nueva secuencia y proporcionando nuevos resultados.

  Este método adquiere la categoría de principal por ser el que, en líneas generales, se ha observado que proporciona mejores resultados.

- Métodos alternativos o secundarios:

  - Mayores retrasos primero: este procedimiento consiste en una reubicación intramáquina: coloca sucesivamente en las primeras posiciones los pedidos que presenten un mayor retraso, recalculando la nueva secuencia.

  - Primero por Último: consiste en la sustitución inequívoca del primer pedido procesado por el último (sustitución intramáquina).

- Orden por Retraso: ordena los pedidos por orden de retraso decreciente, es decir, los más retrasados se sitúan en las primeras posiciones y los más adelantados en los últimos lugares (reubicación intramáquina).

Método Aleatorio 3 métodos: permite ejecutar automática y aleatoriamente los tres primeros métodos anteriormente presentados un número de veces igual al indicado por el usuario. Se ha apreciado mediante experimentación, que a partir de 500 iteraciones los resultados mejoran en muy pequeña proporción y suponen un incremento innecesario del tiempo de cálculo de la CPU.

Para más información, consultar el Anexo II "Manual de usuario de *ProScheduling v2.0*", donde se detallan los procedimientos empleados.

10.2. Ejemplo de procedimiento

Mediante un sencillo ejemplo, que consta de 5 artículos, 2 máquinas y 8 lotes, se pretende explicar el procedimiento de resolución propuesto. Los datos son los siguientes:

		Artículo 1	Artículo 2	Artículo 3	Artículo 4	Artículo 5
Máquina 1	Se fabrica ese art. en esa máquina?	1	0	1	1	1
	Tiempo unitario de proceso	1,8	0,7	0,8	0,2	0,5
	Tiempo de preparación	39	38	21	38	47
Máquina 2	Se fabrica ese art. en esa máquina?	1	1	1	1	1
	Tiempo unitario de proceso	1,4	1	0,1	0,5	1,5
	Tiempo de preparación	23	29	34	43	45

Tabla 10.1. Tiempos de proceso unitarios, tiempos de preparación y limitaciones por máquina y artículo

Pedido	Tamaño lote	Artículo	Fecha entrega
1	141	4	60
2	149	2	30
3	165	1	80
4	270	3	70
5	283	4	60
6	217	2	30
7	282	3	60
8	225	5	10

Tabla 10.2. Datos de los pedidos

Máquina 1	Máquina 2
4	3

Tabla 10.3. Estado inicial de las máquinas

Cálculo de los tiempos de proceso totales para cada máquina:

$$TP_1 = tp_{i1} \cdot q_k$$
$$TP_2 = tp_{i2} \cdot q_k \qquad \text{(Ec. 10.1)}$$

Pedido	Tamaño lote	Artículo	Fecha entrega	TP1	TP2
1	141	4	60	28,2	70,5
2	149	2	30		149
3	165	1	80	297	231
4	270	3	70	216	27
5	283	4	60	56,6	141,5
6	217	2	30		217
7	282	3	60	225,6	28,2
8	225	5	10	112,5	337,5

Tabla 10.4. Tiempos de proceso totales por pedido

10.2.1. Fase I: Algoritmo *Greedy*

Se inicia la simulación, con el reloj a 0.

Se valoran todos los lotes/pedidos en cada máquina, teniendo en cuenta el estado inicial de las máquinas y los artículos que pueden y no pueden fabricarse en cada máquina.

Así pues, se calcula el momento de salida de cada lote y su retraso correspondiente, según Ec. 9.1, Ec. 9.2 y Ec. 9.3:

$$Fin = reloj + tiempo\ \ preparacion + tiempo\,proceso$$
$$Fin = reloj + tp_{(i \to i')j} + P_{ij} \qquad \text{(Ec. 10.2)}$$

$$Retraso = Fin - Fecha\ entrega$$
$$T_k = c_k - d_k \qquad \text{(Ec. 10.3)}$$

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
0	**1**	**4**	**0**	**28,2**	**-31,8**	1	4	0	113,5	53,5
	3	1	0	336	256	2	2	0	178	148
	4	3	0	237	167	3	1	0	254	174
	5	4	0	56,6	-3,4	**4**	**3**	**0**	**27**	**-43**
	7	3	0	246,6	186,6	5	4	0	184,5	124,5
	8	5	0	159,5	149,5	6	2	0	246	216
						7	3	0	28,2	-31,8
						8	5	0	382,5	372,5

Puesto que el lote que da lugar al mínimo retraso en la máquina 1 es el número 1 ($T = -31{,}8$) y en la máquina 2 es el número 4 ($T = -43$), serán los lotes que se asignarán a las respectivas máquinas, eliminándose de la lista. Se actualiza el reloj, sumando el tiempo de reloj anterior más el momento de salida del lote que acaba de procesarse antes (en este caso, el lote 4 en la máquina 2 [27 vs. 28,2]).

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
27	**1**	**4**	**0**	**28,2**	**-31,8**	2	2	27	205	148
	(Se sigue procesando el lote 1)					3	1	27	281	201
						5	4	27	211,5	151,5
						6	2	27	273	243
						7	**3**	**27**	**55,2**	**-4,8**
						8	5	27	409,5	399,5

En este caso, a la hora de calcular el momento de salida de cada lote, debe tenerse en cuenta cómo ha quedado preparada la máquina en el caso anterior.

Obsérvese también que el lote 1 también se ha eliminado de la lista para la máquina 2, pues ya ha sido asignado a la número 1.

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
28,2	3	1	28,2	364,2	284,2	**7**	**3**	**27**	**55,2**	**-4,8**
	5	**4**	**28,2**	**84,8**	**24,8**	(Se sigue procesando el lote 7)				
	8	5	28,2	187,7	177,7					

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
55,2	**5**	**4**	**28,2**	**84,8**	**24,8**	**2**	**2**	**55,2**	**233,2**	**203,2**
	(Se sigue procesando el lote 5)					3	1	55,2	309,2	229,2
						6	2	55,2	301,2	271,2
						8	5	55,2	437,7	427,7

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
84,8	3	1	84,8	420,8	340,8	**2**	**2**	**55,2**	**233,2**	**203,2**
	8	**5**	**84,8**	**244,3**	**234,3**	(Se sigue procesando el lote 2)				

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
233,2	**8**	**5**	**84,8**	**244,3**	**234,3**	**3**	**1**	**233,2**	**487,2**	**407,2**
	(Se sigue procesando el lote 8)					6	2	233,2	450,2	420,2

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
244,3	(Salida del lote 8)					**3**	**1**	**233,2**	**487,2**	**407,2**

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
487,2						**6**	**2**	**487,2**	**733,2**	**703,2**

	MÁQUINA 1					***MÁQUINA 2***				
Reloj	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*	*Lote*	*Artículo*	*Inicio*	*Fin*	*Retraso*
733,2						(Salida lote 6)				

Tabla 10.5. Desarrollo completo del algoritmo *greedy*

Como resultado de la simulación, se obtiene la secuencia de programación, el retraso total y la duración total del proceso de fabricación.

Máquina 1	1	5	8		
Máquina 2	4	7	2	3	6

Tabla 10.6. Secuencia de programación por lotes

Máquina 1	4	4	5		
Máquina 2	3	3	2	1	2

Tabla 10.7. Secuencia de programación por artículos

Máquina 1	0	24,8	234,3		
Máquina 2	0	0	203,2	407,2	703,2

Tabla 10.8. Retrasos asociados a la secuencia

$$\text{Retraso total} = \sum_{k=1}^{z} \text{Retraso unitario} = 1572,7$$

$$\text{Retraso medio} = \frac{\text{Retraso total}}{\text{nº lotes}} = \frac{1572,7}{8} = 196,59$$

Duración total = 733,2

La descompensación en los pedidos procesados por cada máquina es debida a que la máquina 1 no es capaz de fabricar el artículo 2, por lo que los lotes compuestos por ese artículo deben ser procesados exclusivamente en la máquina 2.

Deben tenerse en cuenta dos consideraciones:

- Inicialmente, con un reloj = 0, deben considerarse todas las opciones posibles en todas las máquinas.
- En el caso que para dos o más máquinas el retraso mínimo se obtenga al procesar el mismo lote, éste se asignará a aquella máquina para la que el retraso sea menor (minimización de los mínimos). En consecuencia, el resto de máquinas deberán procesar el lote que posea el segundo retraso más pequeño.
- Lo mismo sucederá si se diera el caso de dos o más lotes procesados en máquinas diferentes y que acaben en el mismo tiempo de reloj.

10.2.2. Fase II: Búsqueda de vecinos

A partir de la secuencia obtenida en la fase I, se procede a la búsqueda de vecinos y al cálculo del retraso asociado a cada uno de ellos. Puesto que las combinaciones resultantes de los múltiples cambios (intermáquina, intramáquina, substitución, inserción, etc) son muy numerosas y el tiempo de valoración de cada una de ellas sería muy elevado, es importante fijar una serie de “criterios” que permitan reducir el tamaño del campo vecinal.

- Un criterio es situar juntos aquellos lotes que contengan el mismo tipo de artículos, pues esto permitirá ahorrar los tiempos de preparación asociados.
- Otro criterio es el dividir el vector secuencia en subvectores, y realizar los movimientos entre ellos.

Por ejemplo, la sustitución intramáquina de los dos últimos artículos a procesar en la máquina 2, da lugar al siguiente vecino (Tabla 10.10). Nótese que se propone procesar los lotes que contengan el mismo artículo de forma consecutiva con la intención de evitar los tiempos de preparación.

Máquina 1	4	4	5		
Máquina 2	3	3	2	**1**	**2**

Tabla 10.9. Solución inicial del algoritmo *greedy* por artículos

Máquina 1	4	4	5		
Máquina 2	3	3	2	**2**	**1**

Tabla 10.10. Vecino generado por sustitución intramáquina

El retraso asociado a la nueva secuencia es:

MÁQUINA 1					
Reloj	**Lote**	**Artículo**	**Inicio**	**Fin**	**Retraso**
0	1	4	0	28,2	-31,8
28,2	5	4	28,2	84,8	24,8
84,8	8	5	84,8	244,3	234,3

MÁQUINA 2					
Reloj	**Lote**	**Artículo**	**Inicio**	**Fin**	**Retraso**
0	4	3	0	27	-43
27	7	3	27	55,2	-4,8
55,2	2	2	55,2	233,2	203,2
233,2	6	2	233,2	450,2	420,2
450,2	3	1	450,2	704,2	624,2

Tabla 10.11. Evaluación del vecino generado

Retraso total =1506,7
Retraso medio = 188,33

Así pues, puesto que el retraso obtenido es menor que el inicial, se actualiza la solución en curso con esta nueva solución.

Se repite el procedimiento hasta que, una vez valorados todos los vecinos posibles, no se pueda garantizar que exista una solución mejor.

11. **Experiencia computacional**

La experiencia computacional se basa en la ejecución de tres procedimientos diferentes:

1. Aplicación del algoritmo *greedy* o Fase I del GRASP, que constituye el punto de partida para los dos métodos siguientes.

2. Aplicación del procedimiento "masivo rápido", que consiste en realizar una secuencia de 10 iteraciones, aplicando alternativamente dos métodos de búsqueda de vecinos: el método principal (sustitución del pedido con mayor retraso por el pedido con mayor adelanto, en la secuencia completa) y un método secundario (colocar los pedidos con mayor retraso en primer lugar de la secuencia).

3. Aplicación del procedimiento "masivo reiterativo", el cual implica llevar a cabo una secuencia alternativa de 500 iteraciones totalmente al azar de los tres métodos que mejores resultados ofrecen: sustitución del pedido con mayor retraso por el pedido con menor retraso, colocar los pedidos con mayor retraso en primer lugar de la secuencia y sustituir el primer pedido por el último.

El hecho de repetir consecutivamente los métodos expuestos sobre el mismo conjunto de datos no reporta ninguna utilidad para el análisis, pues no se aprecian grandes variaciones desde una experiencia a la siguiente. Esto es debido a que el procedimiento "rápido" es inflexible (representa la aplicación de dos métodos concretos alternativamente).

Por su lado, el procedimiento "reiterativo", a pesar de ser el que mayor aleatoriedad presenta, propicia poca variabilidad de los resultados al modificar el número de iteraciones. De todo el conjunto de soluciones que el programa es capaz de calcular, éste es el que mejores resultados proporciona al usuario (lo cual no debe interpretarse como soluciones óptimas globales, sino como óptimos locales del conjunto de soluciones dadas por el software). Este hecho hace que no sea precisa la repetición de experiencias.

El conjunto de datos sobre los que se ha implementado el programa y se ha determinado su eficiencia se ha dividido en tres ficheros de datos, diferenciados en el número de máquinas disponibles y la gama de artículos diferentes a fabricar.

- Caso 8 artículos y 3 máquinas

- Caso 12 artículos y 6 máquinas

- Caso 15 artículos y 9 máquinas

Cada juego de datos consta de una parte fija (los tiempos de preparación, los de proceso unitarios y la disponibilidad de cada máquina para fabricar cada artículo) y una parte variable (la asociada a los datos de los pedidos y el estado inicial de las máquinas). Esta última contiene 100 ejemplares diferentes, con un número de pedidos variable entre 15 y 25.

11.1. Software *ProScheduling v2.0*

A fin de agilizar y automatizar los cálculos asociados al método de resolución propuesto, se ha desarrollado la aplicación informática *ProScheduling v2.0*.

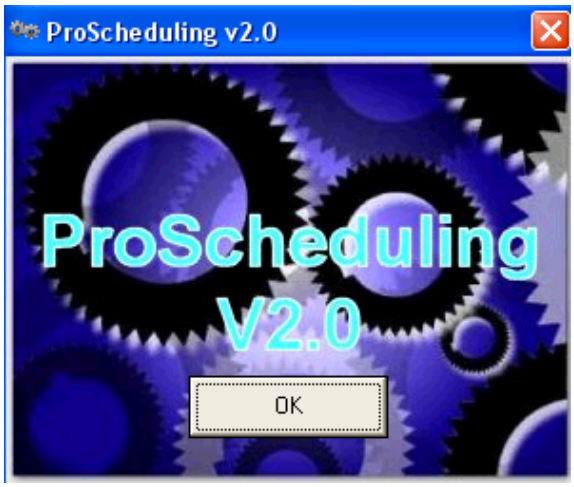


Fig. 11.1. Slash o pantalla de presentación

Este software, desarrollado en *Visual Basic 6* (0[15][12][8][11][16]), ha sido ejecutado en un ordenador con procesador Intel Pentium Centrino a 1400 MHz y 512 Mb de memoria SDRAM.

Los pasos a seguir para la obtención de resultados son los siguientes:

1. En primer lugar, es necesario seleccionar el juego de datos que desea evaluarse. Para ello, basta con navegar mediante el Explorador de Windows hasta la ubicación exacta del fichero.

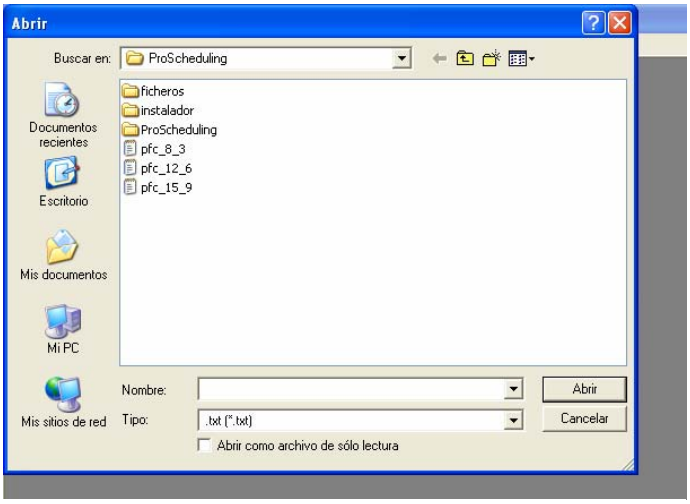


Fig. 11.2. Apertura de ficheros

2. A continuación, el menú “Simulación” muestra todas las opciones de cálculo disponibles. Desde aquí se ejecutan las fases I (algoritmo *greedy*) y II (búsqueda y evaluación de vecinos). Los procedimientos han sido expuestos en los apartados 10.1.1, 10.1.2 y 11. Cabe destacar la posibilidad de tratar cada ejemplar de los 100 disponibles individualmente o de forma masiva.

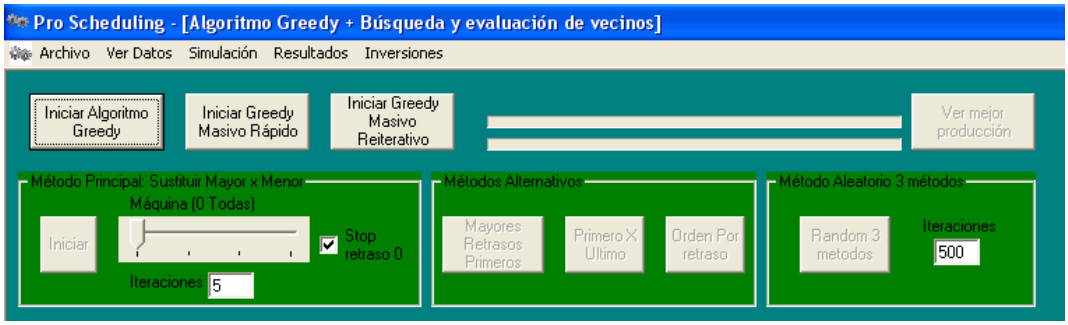
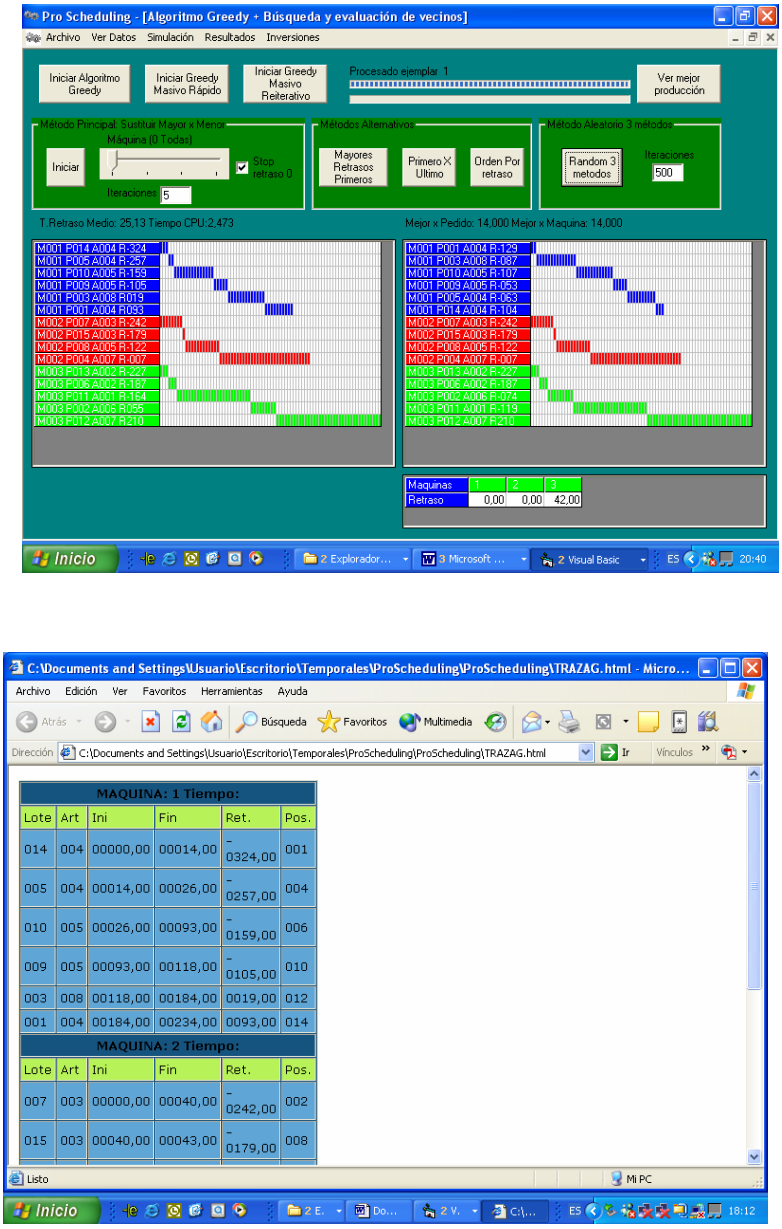


Fig. 11.3. Menú "Simulación"

3. Tras la ejecución, se presentan los resultados. Éstos se muestran tanto visualmente (asociados al diagrama Gantt) como en tablas de resultados (en el menú “Resultados”).



MAQUINA: 1 Tiempo:

Lote	Art	Ini	Fin	Ret.	Pos.
014	004	00000,00	00014,00	-0324,00	001
005	004	00014,00	00026,00	-0257,00	004
010	005	00026,00	00093,00	-0159,00	006
009	005	00093,00	00118,00	-0105,00	010
003	008	00118,00	00184,00	0019,00	012
001	004	00184,00	00234,00	0093,00	014

MAQUINA: 2 Tiempo:

Lote	Art	Ini	Fin	Ret.	Pos.
007	003	00000,00	00040,00	-0242,00	002
015	003	00040,00	00043,00	-0179,00	008

Fig. 11.4. Visualización de resultados: Gantt (arriba) y tablas (abajo)

Para más información, consultar los Anexos I “Estructura de los ficheros de datos” y II “Manual de usuario de *ProScheduling v2.0*”.

11.2. Resultados

A continuación se muestran las tablas de resultados proporcionados por *ProScheduling v2.0*. Éstos se ofrecen ya tratados y promediados respecto a los 100 ejemplares existentes. Para más información, se puede consultar el Anexo III "Resultados obtenidos y comparación".

11.2.1. Tratamiento de los resultados

Las siguientes tablas presentan los resultados promediados proporcionados por el algoritmo *greedy*, el método masivo "rápido" y el método masivo "reiterativo" para cada uno de los tres ficheros de datos, según lo indicado en el apartado 11.

La segunda columna recoge el retraso medio promedio del algoritmo *greedy*. A continuación, tanto para el método masivo "rápido" como para el método masivo "reiterativo" se muestran el retraso medio, el porcentaje de mejora que supone sobre el resultado ofrecido por el algoritmo *greedy* y el tiempo (en segundos) requerido por la CPU del computador para ejecutar el algoritmo.

RESULTADOS PROMEDIADOS	***Greedy***	**Masivo Rápido**			**Masivo Reiterativo**		
	Retraso medio	Retraso medio	Porcentaje de Mejora	Tiempo CPU [s]	Retraso medio	Porcentaje de Mejora	Tiempo CPU [s]
Caso 8 art. y 3 máq.	**47,98**	**11,75**	**-77,07%**	**0,012**	**8,49**	**-83,67%**	**0,668**
Caso 12 art. y 6 máq.	**22,28**	**11,69**	**-49,33%**	**0,013**	**11,32**	**-51,20%**	**0,140**
Caso 15 art. y 9 máq.	**19,99**	**13,63**	**-33,27%**	**0,014**	**13,56**	**-33,57%**	**0,150**

Tabla 11.1. Resultados promediados para los 100 ejemplares y los tres casos analizados

En el juego de datos de 8 artículos y 3 máquinas, el porcentaje de mejora obtenido entre la solución inicial proporcionada por el algoritmo *greedy* y la Fase II de búsqueda y evaluación de vecinos es muy elevado. Ello es debido, en parte, a que el retraso promedio que proporciona la solución del algoritmo *greedy* (47,98) es elevado, y por tanto, es fácil de mejorar.

En ningún caso, la solución propuesta por el algoritmo *greedy* da lugar a un retraso medio igual a cero. Es decir, de su aplicación nunca se obtiene la solución óptima.

La solución obtenida con la aplicación del método masivo "rápido" mejora, en el 100% de los casos, la solución inicial propuesta por el algoritmo *greedy*.

La solución obtenida por el método masivo "reiterativo" pierde eficiencia a medida que aumenta la variedad de artículos a fabricar y el número de máquinas disponibles.

La Tabla 11.2 y Fig. 11.5 muestran en qué medida (porcentaje) el método masivo "reiterativo" mejora la solución obtenida mediante el método masivo "rápido".

	Caso 8 art. y 3 máq.	**Caso 12 art. y 6 máq.**	**Caso 15 art. y 9 máq.**
Solución mejorada	72	30	9
Solución invariada	26	69	91
Solución empeorada	2	1	0

Tabla 11.2. Mejoras relativas entre métodos masivo "rápido" y "reiterativo"

Es decir, en el ejemplar de 8 artículos y 3 máquinas el método masivo "reiterativo" mejora la solución final propuesta por el método masivo "rápido" en un 72% de los casos, permanece inalterada en el 26% de los casos y la empeora en el 2% restante.

A medida que aumenta la complejidad del problema, el método masivo "reiterativo" no tiene suficiente potencia para mejorar la solución del método masivo "rápido", de ahí que ambos ofrezcan la misma solución en un 91% de los casos.

La explicación a este hecho es que de un caso a otro de mayor complejidad aumentan el número de artículos y el de máquinas, pero el de pedidos permanece inalterado. Por tanto, al aumentar el número de máquinas disponibles pero mantenerse el de pedidos, existe menos margen para realizar combinaciones en los pedidos procesados por cada máquina.

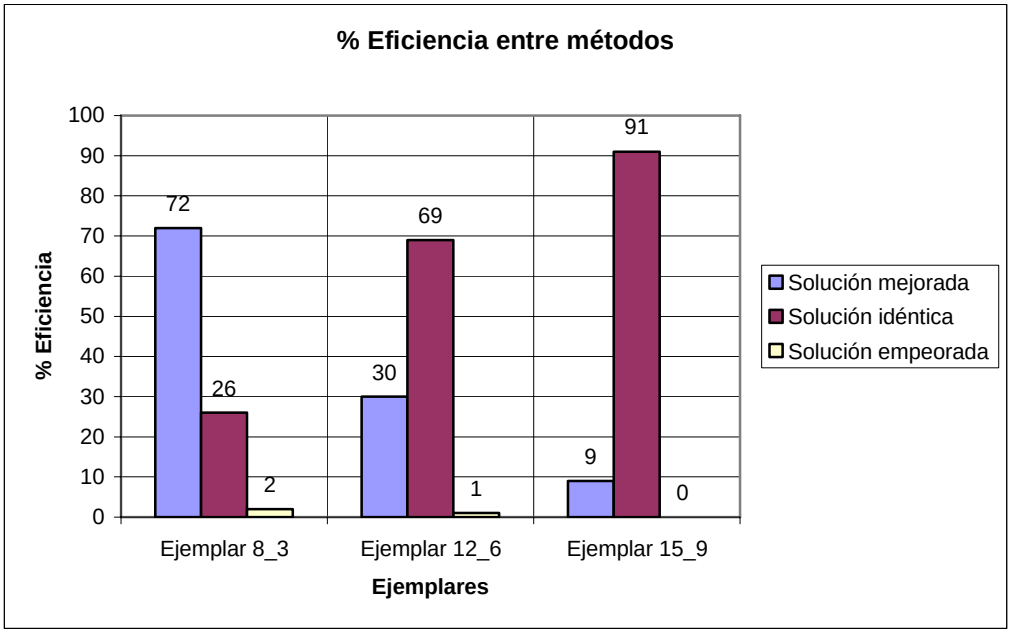


Fig. 11.5. Porcentaje de eficiencia superior del método "reiterativo" frente al "rápido"

La Tabla 11.3 y Fig. 11.6 muestra para cuántos ejemplares, de los 100 disponibles por cada fichero, se obtiene la mejor solución global conocida, es decir, un retraso nulo en la programación de los pedidos recibidos.

	Caso 8_3	**Caso 12_6**	**Caso 15_9**
Retraso 0 Masivo Rápido	12	2	1
Retraso 0 Masivo Reiterativo	16	4	1

Tabla 11.3. Número de ejemplares obtenidos con retraso nulo

Como se observa, el método masivo "reiterativo" obtiene retraso nulo en el 16% de los casos para el caso 8 artículos y 3 máquinas. Como se ha indicado anteriormente, el aumento de complejidad en el caso tratado implica la reducción drástica en el porcentaje de ejemplares con retraso cero.

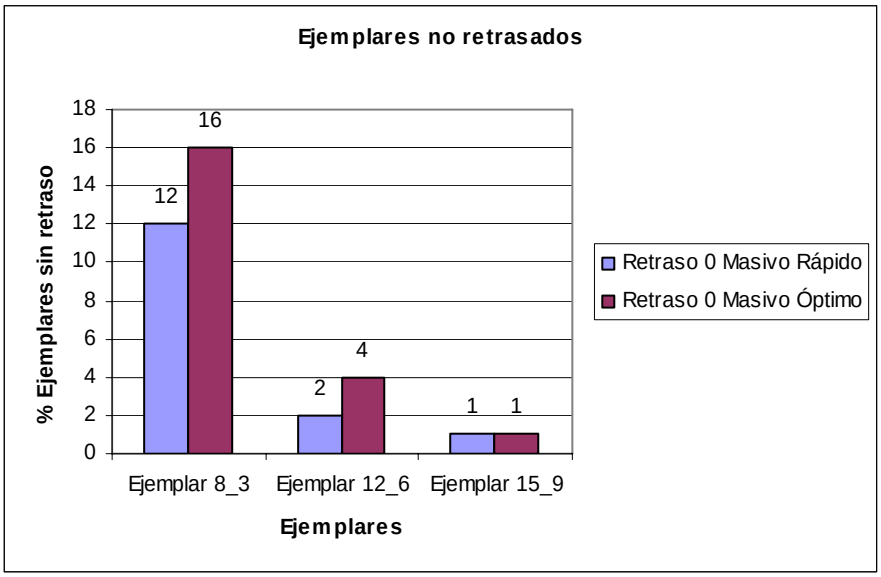


Fig. 11.6. Porcentaje de ejemplares para los que no se ha generado retraso

11.3. Evolución de los tiempos de cálculo de la CPU

Se pretende determinar cómo evolucionan los tiempos de cálculo de la unidad aritmeticológica de la CPU de la computadora al tratar ejemplares de mayores dimensiones.

El tiempo de proceso es función directa, principalmente, de cuatro parámetros:

- El número de ejemplares que deben analizarse: evidentemente, cuanto mayor sea este número, mayor será también el tiempo de cálculo total.
- El número de pedidos que deben programarse: cuanto mayor sea la cantidad de pedidos recibidos, mayor será el tiempo de cálculo requerido para programarlos adecuadamente.

  Los casos experimentados contienen un número de pedidos que oscilan entre los 15 y los 25. Si este valor asciende a, por ejemplo, 99 pedidos, el tiempo se incremente notablemente, tal y como muestra la Tabla 11.4.

	Tiempo CPU del método Masivo Rápido [s]		Tiempo CPU del método Masivo Reiterativo [s]	
Casos tratados	De 15 a 25 pedidos	99 pedidos	De 15 a 25 pedidos	99 pedidos
Caso 8 art. y 3 máq.	0,012	0,080	0,668	0,791
Caso 12 art. y 6 máq.	0,013	0,120	0,140	0,671
Caso 15 art. y 9 máq.	0,014	0,150	0,150	0,661

Tabla 11.4. Evolución de los tiempos de cálculo al aumentar el número de pedidos

Como puede observarse, al incrementar el número de pedidos a fabricar, el tiempo de cálculo se alarga notablemente, especialmente en los casos 12 artículos y 6 máquinas y 15 artículos y 9 máquinas.

- El número de iteraciones a realizar por el programa en el método masivo “reiterativo” que el usuario establezca: cuanto mayor sea, más se incrementará el tiempo requerido por la CPU.

- La relación entre el número de máquinas disponibles y el de pedidos a procesar: a igualdad de pedidos a fabricar, y con independencia de la variedad disponible de artículos diferentes, los casos en los que el número de máquinas es mayor requieren menores tiempos de cálculo.

  Ello es debido a que, al repartir los mismos pedidos entre un mayor número de máquinas, los pedidos asignados a cada máquina son escasos. Esto provoca que el margen de que dispone el método masivo “reiterativo” para la búsqueda de vecinos por sustitución y reubicación intramáquina es menor, proporcionando los resultados en menos tiempo. Esta aseveración puede observarse en la Tabla 11.1 y en la Tabla 11.4.

11.4. Comparación de los resultados GRASP vs. Genético

Partiendo del Proyecto Final de Carrera que lleva por título “Procediments heurístics de programació de comandes en diferents màquines amb temps de preparació dependents de la seqüència” (García Lausán, 2006), se pretende comparar los resultados ofrecidos por un

algoritmo genético frente a los ofrecidos por el algoritmo GRASP desarrollado en el presente trabajo.

De dicha comparación se intentará discernir cuál de los dos ofrece los mejores resultados al tipo de problema tratado. Por este motivo, los datos empleados en la implementación del programa son idénticos en ambos trabajos.

11.4.1. Resultados de la comparación

Se muestran los resultados promediados (respecto a los 100 ejemplares disponibles) derivados de la comparación de las soluciones ofrecidas por los algoritmos GRASP de este proyecto y el algoritmo genético, aplicados ambos sobre el mismo conjunto de datos.

11.4.2. Juego de datos de 8 artículos y 3 máquinas

La Tabla 11.5 muestra el retraso medio promediado, el tiempo de cálculo, en cuántos casos un algoritmo obtiene mejor resultado respecto al otro y cuánto ejemplares con retraso nulo se han obtenido de los 100 disponibles, para el caso 8 artículos y 3 máquinas.

	Alg. Genético	**Alg. GRASP**	**Porcentaje de Diferencia**
Retraso medio promedio	13,04	8,50	-34,85%
Tiempo CPU	16,19	0,668	
Mejores resultados	38	62	
Ejemplares con retraso nulo	8	16	

Tabla 11.5. Comparación entre algoritmos en el caso 8 art. y 3 máq.

11.4.3. Juego de datos de 12 artículos y 6 máquinas

La Tabla 11.5 muestra el retraso medio promediado, el tiempo de cálculo, en cuántos casos un algoritmo obtiene mejor resultado respecto al otro y cuánto ejemplares con retraso nulo se han obtenido de los 100 disponibles, para el caso 12 artículos y 6 máquinas.

	Alg. Genético	Alg. GRASP	Porcentaje de Diferencia
Retraso medio promedio	13,52	11,32	-16,27%
Tiempo CPU	30,24	0,1392	
Mejores resultados	48	52	
Ejemplares con retraso nulo	0	4	

Tabla 11.6. Comparación entre algoritmos en el caso 12 art. y 6 máq.

11.4.4. Juego de datos de 15 artículos y 9 máquinas

La Tabla 11.7 muestra el retraso medio promediado, el tiempo de cálculo, en cuántos casos un algoritmo obtiene mejor resultado respecto al otro y cuánto ejemplares con retraso nulo se han obtenido de los 100 disponibles, para el caso 15 artículos y 9 máquinas.

	Alg. Genético	Alg. GRASP	Porcentaje de Diferencia
Retraso medio promedio	9,22	13,56	47,20%
Tiempo CPU	21,11	0,1500	
Mejores resultados	76	24	
Ejemplares con retraso nulo	0	1	

Tabla 11.7. Comparación entre algoritmos en el caso 15 art. y 9 máq.

Se observa que aplicando el algoritmo GRASP sobre el conjunto de datos como único método de resolución para la búsqueda de una secuencia de operaciones a fin de minimizar el retraso medio, se obtienen unos resultados absolutos mejores que los aportados por el algoritmo genético.

Esto es, si el algoritmo genético se emplea como método único, sin aplicar otra heurística anteriormente que otorgue un adecuado punto de partida, proporciona peores resultados que el algoritmo GRASP ejecutado bajo las mismas condiciones.

El siguiente gráfico de barras muestra el porcentaje de ejemplares en los que se ha obtenido un retraso nulo.

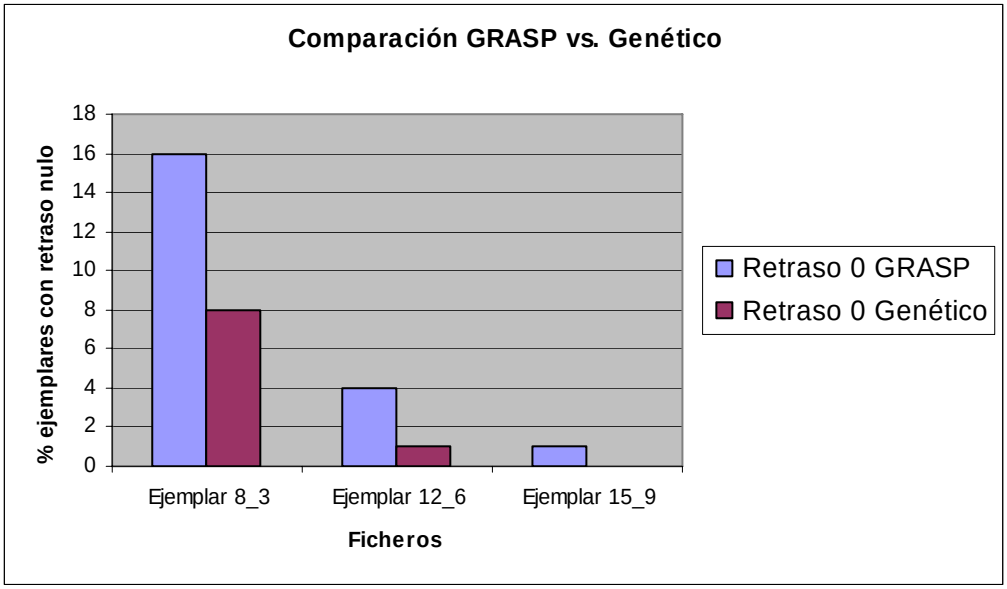


Fig. 11.7. Comparación entre algoritmos GRASP y genético

Este hecho hace pensar que si el algoritmo genético se combinara con métodos previos que proporcionen una solución de partida, los resultados obtenidos serían asimismo mejores que los otorgados por la combinación de AED + genético.

Aunque esto resulta complicado por el hecho de que el GRASP incluye, como parte fundamental, el algoritmo *greedy*, que comporta un efecto similar: proporciona un punto de partida para facilitar la segunda fase.

11.4.5. Conclusiones de la comparación

Comparamos el algoritmo GRASP con el algoritmo genético en igualdad de condiciones, es decir, aplicándolos como único procedimiento de resolución, sin emplear previamente otras heurísticas que proporcionen un punto de partida para el algoritmo.

En estas circunstancias, a tenor de los resultados, se pueden extraer ciertas conclusiones:

- El algoritmo GRASP aporta, en líneas generales, mejores resultados que el algoritmo genético por lo que respecta al retraso medio para los problemas más simples (dentro de la complejidad que presentan), como es el caso de determinar la

secuencia de programación para fabricar de 15 a 25 pedidos con 8 artículos diferentes y 3 máquinas disponibles.

- Para problemas de complejidad intermedia, ambos algoritmos son apropiados, con una cierta ventaja para el GRASP.

- Para problemas con mayor complejidad, resulta más apropiado el algoritmo genético, pues en general, con él se obtienen un menor retraso medio promedio.

- En cambio, el algoritmo GRASP proporciona, en todos los casos, un mayor número de ejemplares planificados para un retraso medio nulo que el genético.

- El tiempo de proceso de la CPU del algoritmo GRASP es considerablemente menor que el requerido por el genético, siendo similar en todos los niveles de complejidad de los datos de entrada. Esto hace que, ante dos soluciones similares aportadas por ambos algoritmos, el tiempo de cálculo sea el factor diferencial que determine la elección de uno sobre el otro.

Por lo que respecta al retraso medio, en líneas generales, si se compraran en las mismas condiciones ambos métodos, es decir, se emplean como método único de resolución sin la aplicación previa de otro método o heurística que otorgue un buen punto de partida y potencie el algoritmo, son adecuados para la obtención de una secuencia de programación de operaciones aceptable.

Por lo que respecta a la obtención de programas de producción con retraso nulo, el algoritmo GRASP aporta sensiblemente mejores resultados.

12. **Presupuesto**

El presupuesto asociado al proyecto se calcula en base al desarrollo del software *ProScheduling v2.0*. Para dicha tarea, se han requerido dos profesionales:

- Ingeniero en Organización Industrial
    - Análisis de la situación actual
    - Estudio de los posibles métodos de resolución
    - Fijar el procedimiento de resolución propuesto (GRASP)
    - Indicar al programador los pasos que debe llevar a cabo el algoritmo
    - Analizar los resultados proporcionados por la aplicación desarrollada
    - Extraer conclusiones y proponer mejoras al proceso
- Informático programador
    - Programar el software en el idioma *Visual Basic 6*, siguiendo las instrucciones del Ingeniero.
    - Realizar pruebas para las distintas situaciones que puedan darse en el programa, evitando estados que puedan comprometer el funcionamiento del programa.

CATEGORÍA PROFESIONAL	SALARIO (€/h)	TIEMPO DEDICADO (h)	COSTE (€)
Ingeniero en Organización Industrial	60 € / hora	68 h	4 080 €
Informático / programador	30 € / hora	108 h	3 240 €
TOTAL (IVA incluido)			**7 320 €**

Tabla 12.1. Cálculo del presupuesto del proyecto

Para más información, consultar el Anexo IV “Timing”.

13. **Evaluación del impacto ambiental**

13.1. Introducción

La implantación en la industria del programa propuesto implica ciertos aspectos medioambientales que cabe considerar. Estos aspectos abarcan desde la propia área de fabricación hasta el transporte de producto acabado.

13.2. Beneficios medioambientales de la implantación

La búsqueda de un programa de fabricación que mejore el retraso medio total puede implicar a su vez la reducción de la duración total del proceso, lo que en términos de ahorro energético supone una reducción en el uso de fuentes de energía, con la reducción en última instancia del consumo de combustibles fósiles, altamente contaminantes para la atmósfera.

Del mismo modo, si la secuencia de programación implica la fabricación consecutiva de pedidos que contengan el mismo artículo, los tiempos de preparación se eliminan, con todo lo que ello implica: se evita tener que limpiar la máquina y/o los moldes u otros complementos necesarios para la fabricación, de tal modo que no se generan residuos derivados de esas operaciones.

Las industrias que tengan concertado un servicio de transporte periódico de producto acabado podrán ajustar más sus previsiones. De este modo, al fabricar los pedidos sin retraso, los camiones podrán transportar la totalidad del pedido. De lo contrario, es posible que el camión deba volver vacío o a media carga, con el consiguiente consumo innecesario de combustible y las emisiones en gases de efecto invernadero.

Finalmente, el hecho de disponer de una herramienta informática para establecer los programas de producción puede implicar la no necesidad de un soporte físico para los datos y cálculos, con la consiguiente reducción en el consumo de papel.

14. **Conclusiones**

El problema tratado en este trabajo es la programación de una serie de pedidos en un conjunto de máquinas, donde son importantes los tiempos de preparación antes de fabricar un producto, con el objetivo de minimizar el retraso medio. A tal efecto, se ha desarrollado un algoritmo GRASP, implementado sobre los datos disponibles con el objetivo de comprobar su eficiencia tanto absoluta como relativa al algoritmo genético.

De la observación de los diagramas de Gantt se aprecia que las máquinas trabajan a *full-time* desde el inicio de la fabricación hasta su finalización, la cual cosa es lógica, pues el número de pedidos es en todos los casos superior al número de máquinas disponibles.

Todos los ejemplares mejoran la solución inicial propuesta por el algoritmo *greedy* al aplicar la segunda fase del GRASP, es decir, la búsqueda y evaluación de los vecinos de dicha solución inicial.

La fase II daría mejores resultados si las máquinas no dispusieran de limitaciones en la fabricación, de tal manera que existe mayor libertad para asignar los pedidos a fabricar a las diversas máquinas disponibles.

Los retrasos medios obtenidos mediante el algoritmo *greedy* son más pequeños, en promedio, para el conjunto de datos del caso 15 artículos y 9 máquinas, mientras que los peores resultados se obtienen para el juego de datos 8 artículos y 3 máquinas. Esto es debido a que, el número de máquinas libres susceptibles de procesar un pedido es mayor, de manera que en el instante inicial es posible procesar 9 de los de entre 15 a 25 pedidos posibles. Es decir, cuanto mayor es el número de máquinas disponibles, con igualdad del número de pedidos a procesar, los resultados obtenidos mediante el algoritmo *greedy* son mejores.

En cambio, conforme aumenta el número de artículos y máquinas disponibles, el porcentaje de mejora al aplicar la fase II (búsqueda y evaluación de vecinos) es inferior. Esto es debido, principalmente, a que en el caso 8 artículos y 3 máquinas, el número de pedidos procesados por cada máquina es mayor, lo que otorga más posibilidades a la hora de modificar el orden de procesado por cada máquina. Por contra, el caso 15 artículos y 9 máquinas es mucho más rígido en este aspecto, de modo que cada máquina, al procesar un conjunto más reducido de pedidos, permite menos combinaciones, y por tanto, un menor número de vecinos.

Este efecto se ve magnificado por las limitaciones a la hora de procesar ciertos artículos que presentan las máquinas.

Como se observa tanto en los resultados proporcionados por el algoritmo GRASP como en los proporcionados por el algoritmo genético, el número de ejemplares con resultado óptimo (es decir, habiendo obtenido un retraso medio nulo) se reduce progresivamente de un conjunto de datos al siguiente, de complejidad creciente.

En líneas generales, el algoritmo GRASP planteado ofrece mejores resultados que el algoritmo genético planteado, tanto por lo que respecta al retraso medio como por lo que respecta al porcentaje de ejemplares no retrasados obtenido. Se muestra especialmente eficiente para el caso 8 artículos y 3 máquinas, debido a que, a igualdad de pedidos, el disponer de menos máquinas hace que los métodos de búsqueda y evaluación de vecinos sean más eficientes, al generar un amplio vecindario por reubicación y sustitución intramáquina.

Por otro lado, se ha observado que el algoritmo genético se muestra más eficiente que el GRASP para problemas de mayor complejidad, es decir, cuando a igualdad de pedidos a fabricar, el número de máquinas es mayor.

Finalmente, es preciso matizar que el algoritmo planteado puede potenciarse incorporando nuevos métodos que permitan buscar y evaluar un mayor número de vecinos, incrementando así su eficiencia. Algunas de esas mejoras se presentan a continuación.

14.1. Sugerencias y mejoras propuestas

El software *ProScheduling v.2.0* admite mejoras para posteriores versiones. Las sugeridas por los autores son las siguientes:

- Llevar a cabo el algoritmo *greedy* pudiendo seleccionar entre varios criterios de asignación de los candidatos de la LRC a cada máquina diferentes del empleado actualmente, como los descritos en el apartado 7.4, así como combinaciones de los mismos. Esto posibilita efectos comparativos.

- Incluir los cambios intermáquina en la fase II de búsqueda de vecinos.

- Incluir una opción que permita al usuario decidir si desea penalizar la entrega anticipada de pedidos (criterio *Just in Time*), de manera que en el programa solución propuesto los pedidos se sirvan en el momento lo más próximo posible a la fecha de entrega correspondiente.

Bibliografía

Referencias bibliográficas

[1] BAUTISTA VALHONDO, JOAQUÍN; MATEO DOLL, MANEL; TUA COSTA, CARLES; IBAÑEZ GINER, JOSÉ Mª; COMPANYS PASCUAL, RAMÓN. *Dirección de Operaciones*. CPDA – ETSEIB (Publicacions d'Abast, S.L.L.): 2003. Pág. 1670-1930.

[2] BARD, JONATHAN y ROJANASOONTHON, SIWATE. REVISTA INFORMS JOURNAL ON COMPUTING. *A GRASP for parallel machine scheduling with time windows. Journal on Computing*. Pág. 10-14. Texas, 2005.

[http://www.me.utexas.edu/~bard/Bard_Papers/GRASP%20Parallel%20Machines%20(IJOC).pdf, 13 de Marzo de 2007].

[3] BLÁZQUEZ IGLESIAS, MATIAS. *Microsoft Visual Basic 6 (Manuales avanzados).* Anaya Multimedia – Anaya Interativa. 1ª Edición. Madrid: 1999.

[4] COBOS ZABALETA, NADIA. UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN. *Búsqueda tabú para un problema de diseño de red multiproducto con capacidad finita en las aristas*. Pág. 6-11. Nuevo León, 2004.

[http://yalma.fime.uanl.mx/~pisis/ftp/pubs/thesis/msc/2004-ncz/tesis-2004-ncz.pdf, 25 de Febrero de 2007].

[5] D'ARMAS, MAYRA y COMPANYS, RAMÓN. *Procedimientos de exploración de entornos para la programación de operaciones en una máquina con tiempos de preparación*. Barcelona, 2005.

[https://e-prints.upc.edu/bitstream/2117/533/1/Procedimientos+de+exploraci%C3%B3n+de+entornos+para+la+programaci%C3%B3n+de+operaciones+en+una+m%C3%A1quina+con+tiempos+de+preparaci%C3%B3n.pdf, 26 de Febrero de 2007].

[6] D'ARMAS, MAYRA y COMPANYS, RAMÓN. *Programación de pedidos en una máquina de la vida real con tiempos de preparación dependientes de la secuencia*. Barcelona, 2005.

[https://e-prints.upc.edu/bitstream/2117/534/1/Programaci%C3%B3n+de+pedidos+en+una+m%

C3%A1quina+de+la+vida+real+con+tiempos+de+preparaci%C3%B3n+dependientes+de+la+secuencia.pdf, 26 de febero de 2007].

[7] GARCIA, MODEST. Procediments heurístics de programació de comandes en diferents màquines amb temps de preparació depenents de la seqüència: memòria, annex I y annex II. UPC: 2006.

[8] MARQUIS, HANK; SMITH, ERICA y WHISLER, VALOR. *El libro de Micrososft: Visual Basic 6*. Anaya Multimedia – Anaya Interactiva. 1ª Edición. Madrid: 1999.

[9] MARTÍ, RAFAEL. UNIVERSITAT DE VALÈNCIA. *Procedimientos Metaheurísticos en optimización combinatoria*. Pág. 14-44. Valencia, 2001.

[http://www.uv.es/~rmarti/paper/docs/heur1.pdf, 31 de Marzo de 2007].

[10] Metaheurísticas

[http://barbacana.net/blog/node/345, 27 de Febrero de 2007].

[11] MICROSOFT DEVELOPER NETWORK. *Forum*

[http://www.microsoft.com/spanish/msdn/default.mspx, 12 de Junio de 2007].

[http://forums.microsoft.com/MSDN-ES/ShowForum.aspx?ForumID=297&SiteID=11, 15 de Junio de 2007].

[12] NAVARRO, LUIS y ARBOLES, SERGIO. *Visual Basic 6 a fondo*. Infor Books. 1ª Edición. Barcelona: 1999.

[13] RESENDE, MAURICIO G.C. *Greedy Randomized Adaptative Search Procedures (GRASP)*. Nueva Jersey, 1998.

[http://cabrillo.lsi.uned.es:8080/aepia/Uploads/19/26.pdf, 14 de Marzo de2007].

[14] RESENDE, MAURICIO G.C. *GRASP: Greedy Randomized Adaptative Sarch Procedure. A methaheuristic for combinatorial optimization*. Nueva Jersey, 2000.

[http://cabrillo.lsi.uned.es:8080/aepia/Uploads/19/26.pdf, 14 de Marzo de 2007].

[15] VV.AA. *Pack Visual Basic 6.0 – Edición de aprendizaje*. McGraw – Hill / Interamericana de España, S.A. Madrid: 1998.

[16] WEB EL GUILLE. *La web de Visual Basic, C#, .NET y más*.

[http://elguille.org/vb, 4 de Julio de 2007]

Bibliografía complementaria

[17] BRUCKER, PETER. *Scheduling algorithms*. Springer. 2ª Edición. Berlin: 1998.

[18] COMPANYS PASCUAL, RAMON y COROMINAS SUBIAS, ALBERT. *Organización de la producción I: diseño de sistemas productivos*. Edicions UPC. Barcelona: 1993–1994.

[19] COMPANYS PASCUAL, RAMON y COROMINAS SUBIAS, ALBERT. *Organización de la producción II: dirección de operaciones*. Edicions UPC. Barcelona: 1995–1996.

[20] COMPANYS PASCUAL, RAMON. *Planificación y programación de la producción*. Marcombo-Boixareu editores. Barcelona: 1989.

[21] GÓMEZ, PEDRO, ANDRÉS, CARLOS y STANIA, LARISSA. *Estudio experimental de un taller cerámico de máquinas paralelas con secuenciación dinámica*. Valencia, 2006.

[http://io.us.es/cio2006/docs/000041_final.pdf, 27 de Febrero de 2007].

[22] GUTIÉRREZ, ELIÉCER y MEJÍA, GONZALO. UNIVERSIDAD DE LOS ANDES. *Evaluación de algoritmos genéticos para el problema de máquinas en paralelo con tiempos de alistamiento dependientes de la secuencia y restricciones en las fechas de entrega*. Colombia.

[http://dspace.uniandes.edu.co:5050/dspace/bitstream/1992/812/1/Evaluacion+de+algoritmos+geneticos+para++el+problema+de+maquinas+en+paralelo.pdf, 27 de febrero de 2007].

[23] MERELO, J.J. *Técnicas heurísticas de resolución de problemas: computación evolutiva y redes neuronales*.

[http://geneura.ugr.es/~jmerelo/tutoriales/heuristics101, 27 de Febrero de 2007].

[24] SEVAUX, MARC y THOMIN, PHILIPPE. *Parallel Machine Scheduling : A (Metha)Heuristic Computational Evaluation*. Oporto, 2001.

[http://web.univ-ubs.fr/lester/~sevaux/Publications/inp-sevaux-01a-slides.pdf, 27 de Febrero de 2007].