Programación de pedidos en un taller con máquinas en paralelo y tiempo de preparación

Manuel Mateo
UPC, Barcelona, España, manel.mateo@upc.edu*T*

Mayra D'Armas
UNEXPO, Puerto Ordaz, Venezuela, mdarmas@unexpo.edu.ve, mjdarmasr@yahoo.es

RESUMEN

En este trabajo se resuelve un problema de programación de los pedidos de producción en un sistema productivo con máquinas en paralelo, con inhibiciones en la fabricación y tiempos de preparación, con el objetivo de minimizar el retraso medio en la entrega de dichos pedidos. Para la resolución del problema se aplicó un procedimiento GRASP. A fin de verificar el correcto funcionamiento y la eficiencia del procedimiento de resolución propuesto, se llevó a cabo una experiencia computacional sobre tres juegos de ejemplares, diferenciándose en el número de máquinas disponibles y el número de artículos diferentes susceptibles de ser fabricados: 8 artículos y 3 máquinas, 12 artículos y 6 máquinas y 15 artículos y 9 máquinas. Cada juego consta, a su vez, de 100 ejemplares, conteniendo de entre 15 a 25 pedidos, con sus correspondientes datos. Los resultados se compararon con los obtenidos con Algoritmos Genéticos. El algoritmo GRASP aporta, en líneas generales, mejores resultados que el Algoritmo Genético por lo que respecta al retraso medio para los problemas de menor dimensión. Sin embargo, para problemas de mayor dimensión resulta más apropiado el Algoritmo Genético.

Palabras claves: secuenciación, máquinas en paralelo, tiempos de preparación, GRASP.

ABSTRACT

This paper solves a scheduling problem of orders in a workshop with parallel machines, inhibition constraints in manufacturing and setup times, with the objective of minimizing the mean tardiness in delivering those orders. To solve the problem we applied a GRASP. In order to verify the proper use and efficiency of the proposed procedure a computational experience was carried out on three data sets, with different number of machines and kinds of jobs (or items) to be produced: 8 items and 3 machines, 12 items and 6 machines and 15 items and 9 machines. Each set has 100 instances, containing from 15 to 25 different orders, with their corresponding data. The results were compared with those obtained using Genetic Algorithms. The GRASP has, in general, a better performance than the Genetic Algorithm for the mean tardiness in the smaller instances. Nevertheless, for bigger instances the Genetic Algorithm is more appropriated.
Keywords: scheduling, parallel machines, setup times, GRASP.

1. INTRODUCCIÓN

Uno de los principales problemas que se presenta 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.

Normalmente, en empresas pequeñas y poco informatizadas, el orden de fabricación se establece siguiendo criterios basados en 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 eficiencia elevado. Además, el hecho de

seguir algoritmos concretos permite que las soluciones obtenidas sean objetivas, permitiéndose la comparación entre procedimientos.

El objetivo de este trabajo es desarrollar un procedimiento que establezca asignación y temporización de los pedidos a fabricar entre las máquinas disponibles para reducir los tiempos muertos y acortar tanto como se pueda el tiempo medio de los retrasos en la entrega de los pedidos. El caso concreto corresponde a un problema tipo taller con las máquinas en paralelo. En el ambiente de máquinas paralelas (ver Figura 1), hay *m* máquinas idénticas en paralelo. La pieza *j* requiere de una operación y puede ser procesada en cualquiera de las *m* máquinas (Narasimhan et al, 1995).

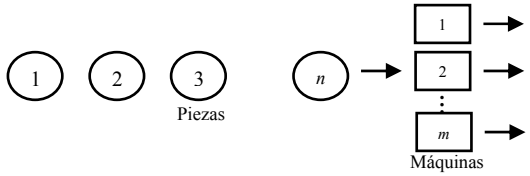

Figura 1. Configuración de *n* piezas, *m* máquinas en paralelo.

De acuerdo con Allahverdi, Gupta y Aldowaisan (1999), en el ambiente de máquinas paralelas, las piezas que llegan pueden ser procesadas en cualquiera de un número de máquinas disponibles. No obstante, aquí cada pieza, con diferentes características, requiere una operación única que no puede realizarse en todas las máquinas. Puede determinarse la programación de piezas para satisfacer un cierto criterio basado en diferentes medidas de eficacia.

La programación de operaciones en máquinas paralelas ha sido ampliamente estudiada (Chen y Sin, 1990). Diversos métodos se han aplicado para resolverlo: métodos de descomposición (Ovacik y Uzsoy, 1996), heurísticas simples (Dunstall yWirth, 2005), Búsqueda Tabú (Logendran et al, 2007), Branch-and-Bound en problemas de dimensiones reducidas (Rocha et al, 2008).

Para la resolución del problema se aplicó una heurística GRASP (Greedy Randomized Adaptive Search Procedure), y se llevó 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: 8 artículos y 3 máquinas, 12 artículos y 6 máquinas y 15 artículos y 9 máquinas. Cada juego consta, a su vez, de 100 instancias diferentes, conteniendo de entre 15 a 25 pedidos diferentes, con sus correspondientes datos. Finalmente, los resultados obtenidos se compararon con los propuestos por otro procedimiento, un Algoritmo Genético (García, 2006), lo que permitió conocer el comportamiento de los algoritmos GRASP y Genético frente al tipo de problema expuesto y a los ejemplares tratados, y permitiendo discernir cuál es el que proporciona mejores resultados en términos generales.

2. HEURÍSTICA GRASP

Los métodos GRASP fueron desarrollados al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimiento de conjuntos (Resende, 2000). El término GRASP fue introducido por Feo y Resende (1989) 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 adaptativas "Voraces" Aleatorizadas. 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
Repetir mientras haya elementos en la lista de candidatos
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

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 éxito de este método se puede constatar en la gran cantidad de aplicaciones que han aparecido en los últimos años.

2.1. FASE DE CONSTRUCCIÓN DE LA SOLUCIÓN INICIAL EN GRASP

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 se 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 un programa de fabricación con un retraso mínimo. 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 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 empleó el criterio EDD (menor fecha de entrega). A cada iteración, del conjunto de candidatos de la LRC evaluados, se asigna a una máquina aquél que presente el menor valor resultado de la diferencia entre el momento de 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 asigna a aquella máquina que presente una menor fecha de entrega.

2.2 FASE DE BÚSQUEDA LOCAL

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 de nuevo el método 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, Métodos Alternativos o Secundario y Método Aleatorio. El 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 re-calculando 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. Los Métodos Alternativos son: (a) Mayores retrasos primero: este procedimiento consiste en una reubicación intramáquina, coloca sucesivamente en las primeras posiciones los pedidos que presenten un mayor retraso, re-calculando la nueva secuencia; (b) Primero por Último: consiste en la sustitución inequívoca del primer pedido procesado por el último (sustitución intramáquina); (c) 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). El Método Aleatorio: permite ejecutar automática y aleatoriamente los tres 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.

3. PLANTEAMIENTO DEL PROBLEMA

El caso tratado se centra en un taller que consta de m máquinas en paralelo, diferentes entre sí, que realizan una misma y única operación, capaces de fabricar hasta n artículos diferentes *bajo pedido*. Se pretende buscar la solución óptima para la fabricación de z pedidos. Cada pedido k consta de un lote, del cual deben servirse q_k unidades de un mismo artículo i y una fecha de entrega d_k, cada uno de los cuales consta de un artículo con el tamaño de lote solicitado y la fecha de entrega. Cada máquina j presenta unos tiempos de preparación al pasar de fabricar un tipo de artículo i a otro diferente i', $s_{(ii')j}$, que contemplan actividades como la limpieza de los restos del artículo anterior, los cambio de moldes, recipientes de recepción, etc. La fabricación de un artículo i de un pedido k en una máquina j lleva asociado un tiempo de preparación y un tiempo de operación p_{ij}. Estos tiempos de operación son diferentes según cada máquina para un mismo artículo. Los parámetros asociados al problema son:

m: número de máquinas; Conjunto de máquinas $j = \{1,...,m\} \subset M$
n: número de artículo; Conjunto de artículos: $i = \{1,..., n\} \subset N$
z: número de pedidos o lotes; Conjunto de pedidos: $k = \{1,..., z\} \subset Z$
$s_{(ii')j}$: tiempo de preparación (cambio entre productos i e i' para cada máquina j)
p_{ij} : tiempo de proceso unitario de cada producto i en cada máquina j
σ_j : secuencia de programación factible para cada máquina j

Para cada pedido k, se establece:
q_k: tamaño de lote o cantidad del artículo solicitado
n_k: artículo i que se solicita en el pedido (cada pedido se compone de un solo tipo de artículo)
d_k: fecha de entrega o de vencimiento
r_k: instante de disponibilidad de cada pedido; dato que se supone r_k=0.

El tiempo de proceso total es el producto del tiempo unitario de proceso por el tamaño de lote: $P_{kj} = p_{ij} \cdot q_k$ (1)

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

$$c_k = r_k + w_k + P_{kj} + s_{(hi)j} \quad (2)$$

Donde w_k: tiempo de espera del pedido k hasta que puede ser procesado
h: artículo producido justo antes del artículo i, solicitado en el pedido k

Y su retraso T_k , viene dado por: $T_k = \max[0; c_k - d_k]$ (3)

3.1 TIEMPOS DE PREPARACIÓN

La estructura de los datos de tiempos de preparación de un artículo a otro en cada máquina ($s_{(ii')j}$) se refleja en la siguiente matriz. Nótese que $s_{(ii')j} \neq s_{(i'i)j}$, 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.

Tabla 1: Matriz tiempos de preparación

S_j	Artículo 1	Artículo 2	…	Artículo i'	…	Artículo n
Artículo 1	0	$S_{(12)j}$		$S_{(1i')j}$		$S_{(1n)j}$
Artículo 2	$S_{(21)j}$	0		$S_{(2i')j}$		$S_{(2n)j}$
…						
Artículo i	$S_{(i1)j}$	$S_{(i2)j}$		$S_{(ii')j}$		$S_{(in)j}$
…						
Artículo n	$S_{(n1)j}$	$S_{(n2)j}$		$S_{(ni')j}$		0

3.2 TIEMPOS UNITARIOS DE PROCESO

Para elaborar cada uno de los artículos *i*, cada máquina *j* requiere unos tiempos de trabajo o proceso p_{ij}.

Tabla 2: Matriz tiempo unitario de procesamiento

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

3.3 DATOS DE LOS PEDIDOS

Para cada pedido *k*, se establecerá el tamaño de cada lote, el artículo *i* que se pide y la fecha de entrega. El objetivo es encontrar, utilizando diferentes procedimientos (heurísticos, metaheurísticos, exactos), la asignación y la secuencia u orden de trabajo que permita reducir el retraso medio entre todos los pedidos. Se penaliza la entrega tardía del lote, pues puede ocasionar costos de diferimiento: el cliente acepta el lote, pero a cambio de una rebaja en el precio estipulado. La estructura de datos de los pedidos que deben fabricarse se muestra en la Tabla 3.

Tabla 3: Estructura de los datos

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

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 costos 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), costos 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 que 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.

3.4 VARIABLES, RESTRICCIONES Y FUNCIÓN OBJETIVO

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

Variables: - Instante de inicio del pedido *k*: t_k ($t_k \geq 0$)
-Instante de finalización del pedido *k* (*completion time*): c_k ($c_k \geq 0$)
- Retraso del pedido *k* (*tardiness*): T_k

Restricciones: Dada una secuencia $\sigma_j = \{\sigma_{j1}, \sigma_{j2}, \ldots\}$ en cada máquina; sea *k* un pedido tal que $k \in \sigma_j$ y ocupa la posición *l* en σ_j:

$$c_{\sigma j0} = 0 \qquad \forall j \tag{4}$$

$$t_k \geq r_k \qquad \forall k \tag{5}$$

$$t_k \geq c_{\sigma jl-1} \qquad \forall k \in \sigma_j \tag{6}$$

$$c_k = t_k + p_{ij} \cdot q_k + s_{(\sigma jl-1\ \sigma jl)j} \qquad \forall k \in \sigma_j \tag{7}$$

$$T_k = max[0; c_k - d_k] \qquad \forall k \tag{8}$$

Función Objetivo: minimización del retraso medio del conjunto de pedidos. $[MIN] T_{med} = \frac{1}{z} \sum_{k=1}^{z} T_k$ (9)

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.

4. METODOLOGÍA

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

- Aplicación del algoritmo *greedy* o Fase I del GRASP, que constituye el punto de partida para los dos métodos siguientes.
- 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).
- Aplicación del procedimiento "masivo reiterativo", el cual implica llevar a cabo una secuencia alternativa de 500 iteraciones totalmente al azar de los tres métodos: 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).

El conjunto de datos sobre los que se implementó el programa y se determinó su eficiencia se dividió en tres conjuntos de instancias, diferenciados en el número de máquinas disponibles y la gama de artículos diferentes a fabricar: (1) 8 artículos y 3 máquinas; (2) 12 artículos y 6 máquinas; y (3) 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). Cada juego de datos contiene 100 ejemplares diferentes, con un número de pedidos variable entre 15 y 25. A fin de agilizar y automatizar los cálculos asociados al método de resolución propuesto, se desarrolló una aplicación informática en lenguaje Visual Basic 6, y la experiencia computacional se ejecutó en un ordenador con procesador Intel Pentium Centrino a 1400 MHz y 512 Mb de memoria SDRAM. Finalmente, los resultados obtenidos con GRASP se compararon con los obtenidos por García (2006), quien había resuelto los mismos ejemplares del problema mediante Algoritmos Genéticos.

5. RESULTADOS

La siguiente tabla presenta los resultados proporcionados por el algoritmo greedy, el método masivo "rápido" y el método masivo "reiterativo" para cada uno de los casos estudiados, comparando 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.

Tabla 4: Resultados promediados para los tres casos analizados

Casos estudiados	Nro. de ejemplares	Greedy	Masivo Rápido			Masivo Reiterativo		
		Retraso medio	Retraso medio	Porcentaje de mejora	Tiempo CPU [s]	Retraso medio	Porcentaje de mejora	Tiempo CPU [s]
8 art. y 3 máq.	100	47,98	11,75	77,07%	0,012	8,49	83,67%	0,668
12 art. y 6 máq.	100	22,28	11,69	49,33%	0,013	11,32	51,20%	0,140
15 art. y 9 máq.	100	19,99	13,63	33,27%	0,014	13,56	33,57%	0,150

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. 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.

En la Figura 2 se puede observar en qué medida (porcentaje) el método masivo "reiterativo" mejora la solución obtenida mediante el método masivo "rápido". Es decir, en los ejemplares 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 ejemplares de 15 artículos y 9 máquinas. La explicación a este hecho es que de un caso a otro de mayor complejidad aumenta el número de artículos y de máquinas, pero no el de pedidos. Por tanto, al aumentar el número de máquinas disponibles manteniendo el de pedidos, existe menos margen para realizar nuevas combinaciones en los pedidos procesados por cada máquina.

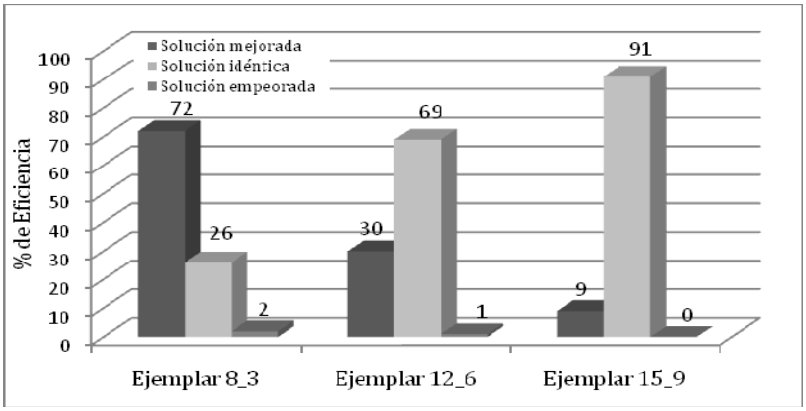

Figura 2. Porcentaje de eficiencia del método "reiterativo" con respecto al método "rápido"

La Figura 3 muestra para cuántos ejemplares, de los 100 disponibles por cada conjunto, se obtiene la mejor solución global conocida, es decir, un retraso nulo según la programación de los pedidos. 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.

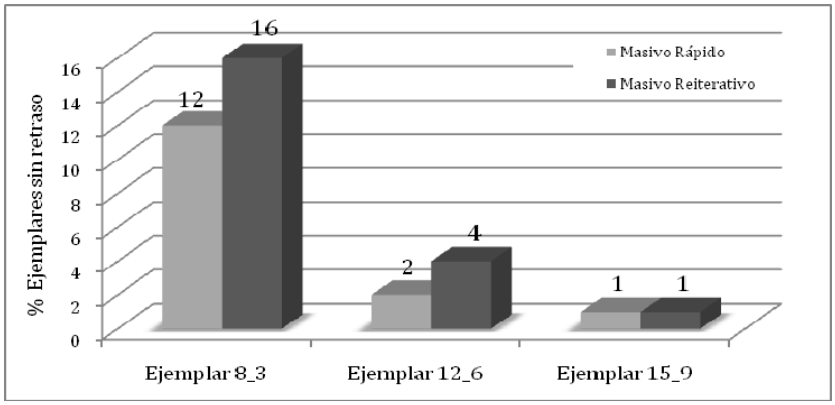

Figura 3. Porcentaje de ejemplares para los que no se ha generado retraso

5.1 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 ejemplares 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 incrementa considerablemente, tal y como muestra la Tabla 5. Como puede observarse, al incrementar el número de pedidos a fabricar, el tiempo de cálculo aumenta 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 de artículos, cuando el número de máquinas es mayor se requiere 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 5.

Tabla 5: Evolución de los tiempos de cálculo al aumentar el número de pedidos

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

5.2 COMPARACIÓN DE LOS RESULTADOS GRASP VS. ALGORITMO GENÉTICO

Los resultados obtenidos por el algoritmo GRASP desarrollado en el presente trabajo se compararon con los resultados obtenidos para el mismo conjunto de ejemplares por García (2006) mediante la aplicación de un Algoritmo Genético. De dicha comparación se analizó 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. Los resultados de la comparación se muestran promediados (respecto a los 100 ejemplares disponibles) derivados de la comparación de las soluciones ofrecidas por los algoritmos GRASP de este trabajo y el Algoritmo Genético, aplicados ambos sobre el mismo conjunto de ejemplares. Las Tablas 6, 7 y 8 muestran el retraso medio promediado, el tiempo de cálculo, la cantidad de mejores resultados obtenidos y el número de ejemplares con retraso nulo que se han obtenido de los 100 disponibles, para los casos de 8 artículos y 3 máquinas; 12 artículos y 6 máquinas; y 15 artículos y 9 máquinas, respectivamente. Además, en la Figura 4 muestra gráficamente el porcentaje de ejemplares en los que se ha obtenido un retraso nulo tanto con GRASP como con el Algoritmo Genético.

Tabla 6: Comparación GRASP vs. Algoritmo Genético. Caso 8 artículos y 3 máquinas

	Algoritmo Genético	Algoritmo GRASP	% de diferencia
Retraso medio	13,04	8,50	-34,85%
Tiempo CPU	16,19	0,668	
Mejores resultados	38	62	

Ejemplares con retraso nulo	8	16

Tabla 7: Comparación GRASP vs. Algoritmo Genético. Caso 12 artículos y 6 máquinas

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

Tabla 8: Comparación GRASP vs. Algoritmo Genético. Caso 15 artículos y 9 máquinas

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

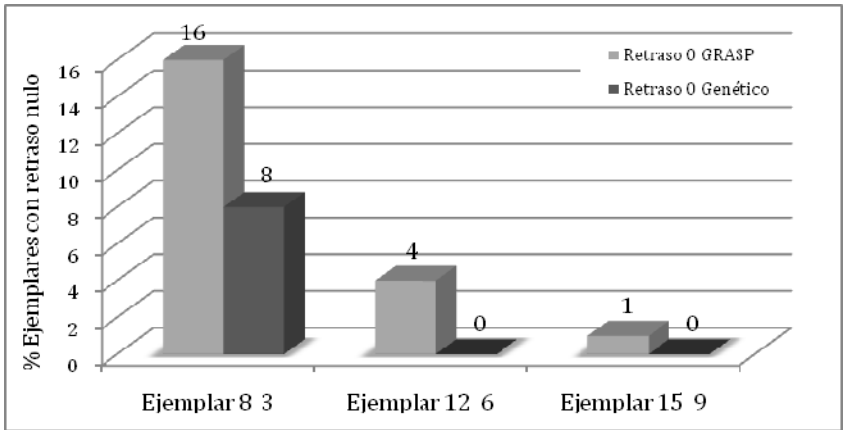

Figura 4. Comparación retraso nulo: Algoritmo GRASP vs. Algoritmo Genético

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. 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 (Algoritmo Exhaustivo de Descenso) + Algoritmo 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.

6. 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 desarrolló un algoritmo GRASP, implementado sobre un conjunto de instancias con el objetivo de comprobar su eficiencia tanto absoluta como relativa al Algoritmo Genético. 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.

Los retrasos medios obtenidos mediante el algoritmo greedy son más pequeños, en promedio, para el conjunto de ejemplares con 15 artículos y 9 máquinas, mientras que los peores resultados se obtienen para los ejemplares de 8 artículos y 3 máquinas. Esto es debido a que, el número de máquinas libres susceptibles de procesar un pedido es mayor. 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 de 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 en cada máquina. Por contra, el caso de 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 inhibiciones a la hora de procesar ciertos artículos que presentan las máquinas.

En cuanto a la comparación del GRASP vs. Algoritmo Genético, se observa que tanto en los resultados proporcionados por el algoritmo GRASP como en los proporcionados por el Algoritmo Genético, el número de ejemplares con retraso medio nulo se reduce progresivamente de un conjunto de ejemplares 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 de 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 otro lado, se encontró que el Algoritmo Genético se muestra más eficiente que el GRASP para problemas de mayor magnitud, es decir, cuando los de pedidos a fabricar y 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, por lo que se sugiere incluir los cambios intermáquina en la Fase II de búsqueda de vecinos.

REFERENCIAS

Allahverdi A., Gupta J. y Aldowaisan T. (1999). A review of scheduling research involving setup considerations. Omega, *International Journal of Management Science*. Vol. 27, p. 219-239

Chen,T.C.E., y Sin, C.C.S. (1990). A state-of-the-ar t review of parallel-machine scheduling research. *European Journal of Operational Research.* Vol. 47, pp. 271-292.

Dunstall S. yWirth A. (2005).Heuristic methods for the identical parallel machine flowtime problem with setup times, *Computers and Operational Research*.Vol. 32, pp. 2479–2491.

Feo T. y Resende. (1989), "A probabilistic heuristic for a computationally difficult set covering problem", *Operations Research Letters*. Vol. 8, pp.67-71

García M. (2006). Procediments heuristics de programació de comandes en diferents maquines amb tems de preparació depenents de la sequencia. Proyecto de Fin de Carrera. Universidad Politécnica de Cataluña.

Logendran R., Mcdonell B. y Smucker, B. (2007). Scheduling unrelated machines with sequence-dependent setups. *Computers and Operations Research*. Vol. 34 (11), pp. 3420-3438.

Narasimhan S., McLeavey D. y Billington P. (1995). Production Planning and Inventory Control. Prentice Hall. 2da edición. New Jersey.

Ovacik I.M. y Uzsoy R. (1996).Decomposition Methods for Scheduling Semiconductor Testing Facilities. *International Journal of Flexible Manufacturing Systems*. Vol. 8, pp. 357-388.

Resende M. (2000). GRASP: Greedy Randomized Adaptative Search Procedure. A methaheuristic for combinatorial optimization. New Jersey.

Rocha, P.L., Ravetti,M.G., Mateus,G.R, y Pardalos, P.M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine- dependent setup times. *Computers and Operations Research*. Vol. 35 (4), pp. 1250-1264.

Autorización y Renuncia

Los autores autorizan a LACCEI para publicar el escrito en las memorias de la conferencia. LACCEI o los editores no son responsables ni por el contenido ni por las implicaciones de lo que esta expresado en el escrito.