VIII Congreso de Ingeniería de Organización
Leganés, 9 y 10 de septiembre de 2004

Secuenciación de tareas mediante metaheurísticos

Santiago López de Haro[1], Pedro Sánchez Martín[2], Javier Conde Collado[3]

[1] Instituto de Investigación Tecnológica. Escuela Técnica Superior de Ingeniería. Universidad Pontificia Comillas de Madrid. Alberto Aguilera, 23. 28015 Madrid. santiago.lopezdeharo@iit.icai.upco.es
[2] Departamento de Organización Industrial. Escuela Técnica Superior de Ingeniería. Universidad Pontificia Comillas de Madrid. Alberto Aguilera, 23. 28015 Madrid. pedro.sanchez@iit.icai.upco.es
[3] Área de Organización de empresas. Escuela Técnica Superior de Ingeniería de Caminos, Canales y Puertos. Universidad de Castilla La Mancha. Avda. Camino José Cela s/n. 13071 Ciudad Real. javier.conde@uclm.es

Resumen

En el presente artículo se muestra una visión global del problema de la secuenciación de tareas y de su resolución mediante procedimientos metaheurísticos, tanto mediante algoritmos genéticos como mediante procedimientos de búsqueda local tales como el recocido simulado o la búsqueda tabú. Por último, también se muestra una comparativa de los resultados obtenidos por dos implantaciones de los algoritmos anteriormente descritos para problemas de tamaños diferentes.

Palabras clave: Job shop, Metaheurísticos, genéticos, recocido simulado, búsqueda tabú

1. Introducción y descripción del problema

El problema de la secuenciación de tareas consiste en la asignación de recursos a diferentes actividades que se ejecutan simultáneamente a lo largo del tiempo. El rango de aplicación de la teoría de secuenciación abarca áreas de conocimiento desde producción hasta computación pasando por logística, servicios y otros. En adelante se adopta la terminología de producción, donde los trabajos son las actividades descompuestas en tareas u operaciones y las máquinas representan recursos.

El problema de secuenciación de tareas (*job-shop scheduling problem,* JSSP) consiste en la programación temporal de las operaciones o tareas en las que se descomponen un conjunto de trabajos teniendo en cuenta que éstas deben ser ejecutadas en varias máquinas y que cada máquina solamente puede ejecutar una tarea simultáneamente. Se busca aquella solución que dé lugar a un tiempo total de ejecución (C_{max}) mínimo, aunque pueden existir otras funciones objetivo como la minimización de la suma de tiempos de espera o la suma de los tiempos de ejecución. Se trata de uno de los problemas de optimización combinatoria más difíciles de resolver. No sólo es del tipo *NP-Hard*, sino que de entre los que pertenecen a esta tipología, es de los más difíciles de resolver. A continuación se describe el JSSP formalmente:

Sea $V=\{1,...,n\}$ el conjunto de todas las tareas. Sea $M=\{1,...,m\}$ el conjunto de las máquinas en que se deben ejecutar dichas tareas y A el conjunto de pares de operaciones cuya relación de precedencia viene predeterminada por la secuencia técnica de cada trabajo. Cada conjunto E_k contiene las operaciones que deben ejecutarse en la máquina k, es decir, las operaciones que no se pueden superponer temporalmente. Para cada operación i, su duración p_i está fijada (nu-

la para las tareas *0* y *f*) y la optimización debe determinar su tiempo de inicio t_i. La ordenación de todas las operaciones en los conjuntos E_k da lugar a una solución para el JSSP planteado.

El problema puede ser representado mediante un grafo disjunto (Roy y Sussman, 1964), donde existe un nodo por cada operación $i \in V$. *0* y *f* son dos nodos que representan el inicio y final de todas las tareas. Para cada dos operaciones consecutivas en el mismo trabajo $(i,j) \in A$ existe un arco dirigido, las operaciones *0* y *n* son la primera y última operación de todos los trabajos respectivamente. Para cada par de operaciones que emplean la misma máquina $\{i,j\} \in E_k$ existen dos arcos *(i, j)* y *(j, i)* en sentidos opuestos que indican cuál es la que se debe ejecutar antes. Cada nodo *i* tiene asociado un peso p_i que indica el tiempo de ejecución de la operación *i*. La Figura 1, representa el grafo disjunto asociado a un problema de 3 máquinas y 3 trabajos (3 x 3).

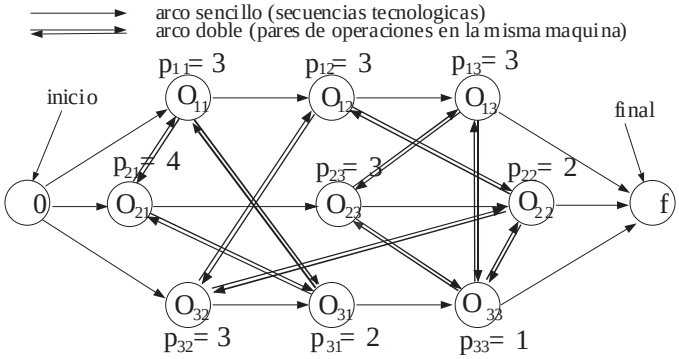

Figura 1. Representación del JSSP en forma de grafo disjunto

El JSSP consiste en seleccionar el orden en que las operaciones deben realizarse en cada máquina, lo cual significa seleccionar uno de cada par de arcos en sentidos opuestos tal que el grafo sea no sea cíclico (no se den conflictos de precedencia entre las operaciones) y que la longitud total del camino más largo (camino crítico) entre el nudo *0* y el nudo *n* sea mínima (se supone que la función objetivo es minimizar C_{max}). Si una orientación de arcos da lugar a un grafo cíclico, la orientación o solución correspondiente se denomina infactible.

En las secciones 2 y 3 se explica el funcionamiento de los algoritmos genéticos y los basados en búsqueda local y se describen los procedimientos aplicados en la literatura para la resolución del JSSP mediante estos metaheurísticos. En la sección 4 se muestran los resultados obtenidos al ejecutar implementaciones codificadas por los autores.

2. Algoritmos genéticos

Desde el punto de vista de la investigación operativa los algoritmos genéticos (AG) pueden ser clasificados como procedimientos inteligentes de búsqueda aleatoria. Están basados en la teoría de la selección natural y tienen la gran ventaja de que gracias a la separación *genotipo-fenotipo* pueden emplearse para buscar soluciones en ámbitos tan diversos como el análisis de imágenes, el procesamiento de señales, el diseño de robots autónomos y la programación automática, siendo también posible su uso en la resolución de problemas combinatoriales como el JSSP. Otra de las ventajas de los AG es la robustez al ser éstos capaces de obtener soluciones subóptimas aún en los casos en los que la solución óptima es muy difícil de encontrar. Los AG fueron desarrollados inicialmente en el libro *Adaptation in Natural and Artificial Systems* (Holland, 1975), donde aparecieron tratados de una forma sistemática por primera vez.

Están basados en un conjunto de individuos o *población* que tienen dos representaciones denominadas *fenotipo* y *genotipo*, siendo la primera una solución potencial al problema que se desea resolver y la segunda una codificación de dicha solución en forma de cromosoma. Un *cromosoma* está formado por una secuencia de *genes* que representan características hereditarias. Los cromosomas de una población *evolucionan* en sucesivas iteraciones, denominadas *generaciones*. Los operadores *cruce* y *mutación* son mecanismos de recombinación genética capaces de generar nuevos cromosomas a partir de otros. Un operador de cruce sencillo podría intercambiar las secuencias que se formarían al cortar ambos cromosomas por un *punto de corte*. Este método suele funcionar bien para una representación binaria, aunque para otras no suele ser el más indicado. La eficacia del algoritmo de cruce está directamente relacionada con su capacidad de recoger las características de los progenitores que los hacen *buenos* y recombinarlas para crear individuos *mejores*. El operador mutación es el encargado de modificar esporádicamente individuos aleatorios para evitar que en el transcurso de las iteraciones el operador de cruce genere individuos cada vez más homogéneos y la población se concentre alrededor de un mínimo local.

En la Figura 2 se hace una clasificación de la bibliografía atendiendo al tipo de representación que emplean.

Representación	Referencia
Binaria basada en pares de trabajos	Nakano y Yamada (1991)
Secuencia de trabajos	Holsapple et al. (1993)
Secuencia de operaciones sin particiones	Fang et al. (1993), Bierwirth (1995)
Secuencia de operaciones con particiones basada en listas de preferencia	Falkenauer y Bouffouix (1991), Croce et al. (1995)
Basada en reglas de prioridad	Dorndorf y Pesch (1995)
Secuencia de operaciones con particiones y operadores especializados	Yamada y Nakano (1992)
Secuencia de operaciones basada en números aleatorios	Bean (1994)

Figura 2. Tipos de representación

La justificación teórica sobre la convergencia de los algoritmos genéticos está fundamentalmente basada en codificaciones binarias. Así, Nakano y Yamada (1991) propusieron una *representación binaria* para el JSSP formada por una secuencia de bits para cada máquina en la que cada bit representaba cuál de las dos orientaciones de cada arco elegir. Así pues, basta con una cadena binaria de longitud $m \cdot n \cdot (n-1)/2$, para designar a todas las soluciones posibles de un problema de tamaño $m \times n$. La máxima desventaja de esta codificación es que no aprovecha el conocimiento del problema y de las restricciones del mismo, dando así lugar a que un gran número de las nuevas soluciones obtenidas de operaciones de cruce y mutación no especializadas resulten infactibles, es decir, que no puedan ser traducidas en una programación válida. Aunque los autores resolvían esta dificultad mediante un proceso denominado *armonización* consistente en deshacer los ciclos en el grafo disjunto correspondiente.

El óptimo global de cualquier problema JSSP se encuentra dentro del conjunto de las programaciones activas (aquellas tales que no existen tareas que puedan ser adelantadas sin retrasar el inicio de las operaciones previas en el recurso), que a su vez es un subconjunto contenido

en el de las programaciones semiactivas (aquellas tales que ninguna tarea puede adelantarse sin modificar las precedencias o que se produzca un solapamiento dentro del recurso).

Holsapple, Jacob et al. (1993) propone una *representación basada en una secuencia de trabajos* que indica la prioridad con la que se deben programar las tareas de éstos. La longitud del cromosoma en este caso se reduce a *m* números de trabajo. Sin embargo, no todas las soluciones posibles del problema pueden ser representadas de esta forma y, por tanto, no está garantizado que lo pueda ser el óptimo global. Los resultados de la Figura 4 muestran que éste un buen método para encontrar una solución inicial razonable en poco tiempo.

Dorndorf y Pesch (1995) propone un algoritmo genético con un cromosoma basado en reglas de prioridad con criterios que determinan la tarea prioritaria de entre aquéllas que están en la cola de espera de una determinada máquina. Así pues, el cromosoma es una secuencia de longitud *m* x *n* de números que indican reglas de prioridad. Obsérvese que no hay garantía de que todas las programaciones posibles pueden ser obtenidas mediante esta representación.

Dado un problema de tamaño *m* x *n*, las codificaciones basadas en secuencias de tareas emplean un cromosoma formado por una serie de *n* números de trabajo repetidos *m* veces. Dentro de este grupo se pueden clasificar atendiendo a la existencia o no de particiones en la secuencia. Fang, Ross et al. (1993), Reeves (1995) y Bierwirth (1995) emplean una formulación sin particiones en la que el número en cada posición indica el siguiente trabajo a programar. La primera operación todavía no programada en la secuencia técnica de dicho trabajo determina la máquina donde se ejecuta. De este modo se garantiza la imposibilidad de generar genotipos infactibles y que todas las programaciones semiactivas pueden ser representadas. La Figura 3 muestra un ejemplo de este tipo de representación en el que un cromosoma de 3 x 3 determina una ordenación de las tareas en cada máquina. Inicialmente, se debe programar el trabajo 1, como la primera tarea no programada del trabajo 1 se ejecuta en M_1 ésta es la máquina en la que se debe situar la operación, luego primera tarea del trabajo 3 se ejecuta en M_2, etc... Obsérvese la inversión de los dos primeros genes del cromosoma de la Figura 3 daría lugar a la misma ordenación de las tareas en los recursos, por lo que el número de cromosomas posibles es mayor que el número de programaciones semiactivas y aún mayor que el de programaciones activas, lo cual relentiza la convergencia del algoritmo genético.

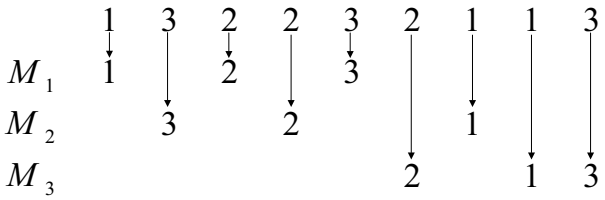

Figura 3. Interpretación de secuencia sin particiones

Algunos autores emplean una representación basada en una secuencia con particiones para cada máquina, representando cada una de estas subsecuencias el orden en que las tareas deben ser ejecutadas en dicha máquina de modo que cada programación semiactiva sólo pueda ser representada por un único cromosoma. La desventaja de este tipo de representación es que existen cromosomas infactibles. Falkenauer y Bouffouix (1991) y Croce, Tadei et al. (1995) solucionan este problema interpretando la secuencia de cada máquina como una lista de preferencia, para que en caso de conflicto con la secuencia técnica de los trabajos, sea ésta la predominante. Yamada y Nakano (1992) garantiza que los cromosomas generados siempre son

factibles y que representan programaciones activas gracias a un operador de cruce especializado para el JSSP, el GTX.

El mecanismo de cruce más habitualmente empleado en codificaciones binarias es el SGA, consistente en el intercambio de subsecuencias entre dos cromosomas. En estos casos, el operador de mutación comúnmente empleado es el SGA, basado en negar un bit aleatorio. En codificaciones basadas en las operaciones sin particiones en la secuencia se pueden emplear los cruces GOX, GPMX y PPX (descritos por Bierwirth, 1995), siendo éste el que conserva la máxima cantidad de información de los cromosomas cruzados, tal y como demuestra experimentalmente Bierwirth. El cruce PPX consiste en inicializar vacío el cromosoma generado y emplear un vector binario que selecciona cuál de los dos cromosomas originales predomina en cada iteración. Luego se ejecuta un proceso iterativo que va insertando en cada posición del cromosoma generado el gen correspondiente al progenitor predominante y eliminando dicho gen de ambos progenitores para que no pueda volver a ser seleccionado. El operador mutación comúnmente aplicado en estos casos es el OBM, que intercambia los genes en dos posiciones aleatorias del cromosoma. El algoritmo GTX, anteriormente mencionado, comienza igual que el PPX, pero se ejecuta un proceso iterativo similar al algoritmo de Giffler y Thompson (1960) en el que es el cromosoma preponderante el que determina cuál de las operaciones en el conjunto de conflicto debe ser seleccionada. La mutación se realiza cuando en lugar de ser el gen preponderante el que selecciona la tarea prioritaria del conjunto de conflicto, ésta se selecciona aleatoriamente.

En la Figura 4 se resumen la codificación y los operadores de cruce y mutación empleados por cada referencia de la bibliografía en las primeras columnas. Las columnas *tamaño* y *comp.* muestran el tamaño del espacio de búsqueda para un problema de tamaño (10 x 10) y si está garantizado que en dicho espacio se encuentra el óptimo global. Las columnas finales indican para cada problema de comparación *ft06*, *ft10* y *yn1* (extraídos de Beasley, 1990), los miles de soluciones evaluadas (Eval), el resultado obtenido (valor de C_{max}), y el tiempo de ejecución (Tpo).

Obsérvese que la codificación propuesta por Yamada y Nakano (1992) es la que mejores resultados obtiene de todas las mostradas, esto es en parte debido a que es la más especializada para el JSSP, ya que su operador de cruce solamente genera soluciones activas reduciendo así el espacio de búsqueda.

						ft06(6x6)			ft10(10x10)			yn1(20x20)		
Codificación	Representación	Cruce	Mut.	Tamaño	Comp.	Eval x1000	Res.	Tpo.	Eval x1000	Res.	Tpo.	Eval x1000	Res.	Tpo.
Nakano y Yamada (1991)	binaria	SGA	MGA	$2^{900} \simeq 10^{271}$	Sí									
Holsapple et al.(1993)	secuencia de trabajos	GOX GPMX PPX	OBM	$10! \simeq 10^{7}$	No	10	58	9"[a]	10	58	9"[a]	10	1301	44"[a]
Dorndorf y Pesch (1995)	secuencia de reglas de prioridad	GOX GPMX PPX	OBM	$7^{100} \simeq 10^{84}$	No	10	65	6"[a]	10	1313	8"[a]	10	1484	107"[a]
Fang et al (1993)	secuencia de operaciones sin particiones	GOX GPMX PPX	OBM	$\frac{100!}{10!^{10}} \simeq 10^{92}$	Sí	10	69	5"[a]	10	1636	7"[a]	10	1857	22"[a]
Falkenauer y Bouffoix (1991)	secuencia de operaciones con particiones listas de preferencia	GOX GPMX PPX	OBM	$10 \cdot 10! \simeq 10^{8}$	Sí	10	58	9"[a]	10	1052	15"[a]	10	1012	55"[a]
Yamada y Nakano (1992)	secuencia de operaciones con particiones y operadores especializados	GTX		$10 \cdot 10! \simeq 10^{8}$	Sí		55	10"[o]		930	10"[o]	400	967	

[a] Resultados obtenidos por Ponnanbalam et al (2001) en la plataforma informática Intel Celeron MMX CPU 256MHz 32MbRam
[o] Resultados obtenidos por Yamada y Nakano (1992) en la plataforma informática Sparc Station 2

Figura 4. Comparación de las diferentes codificaciones

3. Búsqueda local

Los algoritmos de recocido simulado y búsqueda tabú explicados más abajo se clasifican entre los procedimientos de búsqueda local. La cual está basada en la idea de que una determinada solución puede ser mejorada realizando cambios pequeños. Sea un conjunto □ de *soluciones factibles* cuya bondad viene dada por una función de coste *c:*□ $\rightarrow \mathcal{R}$, que corresponde a la función que se desea optimizar (se supone que la dirección de optimización es de minimización). Se define para cada solución $x \in$□ un *vecindario N(x)*□□ donde cada solución contenida en *N(x)* es denominada *vecino* y puede ser alcanzada mediante un *desplazamiento unitario*. Así pues, una ejecución de un algoritmo de búsqueda local determina un camino por el que cada solución visitada es vecina de la inmediatamente anterior.

Al igual que en algoritmos genéticos, un elemento fundamental de todos los algoritmos de búsqueda local es la representación de las soluciones. Tanto los algoritmos de recocido simulado aplicados al JSSP como los de búsqueda tabú explicados más abajo emplean una representación basada en el orden de las operaciones en las máquinas. Sin embargo el elemento determinante en los algoritmos de búsqueda local es la creación de soluciones cercanas o *vecinas*. Antes de explicar los diferentes algoritmos de generación de vecindarios explicados en la literatura es necesario entender varios conceptos:

Sean *v* y *w* dos tareas que emplean el mismo recurso. Éstas son adyacentes (*v*→*w*) si *w* comienza inmediatamente después de terminar *v*. Un bloque es una secuencia de tareas adyacentes en el mismo recurso. Además, se dice que una tarea es *interna* en un bloque si no es la primera ni la última en ejecutarse.

Se deben tener en cuenta las siguientes propiedades a la hora de generar un vecindario:

1. Dadas dos tareas adyacentes *v* y *w* (v→w) pertenecientes al camino crítico de una solución factible *S*. La inversión del arco orientado en el grafo correspondiente (*w*→*v*) da lugar a otra solución factible *S′*.

2. Si la inversión de cualquier arco entre dos operaciones no pertenecientes al camino crítico diera lugar a otra solución S' factible. Entonces $C_{max}(S') \geq C_{max}(S)$.
3. Sean v y w ($v \rightarrow w$) dos operaciones internas y adyacentes en un bloque perteneciente al camino crítico. La inversión del arco que las une ($w \rightarrow v$) da lugar a otra solución factible S' tal que $C_{max}(S') \geq C_{max}(S)$.
4. Sean v y w ($v \rightarrow w$) dos operaciones adyacentes en un bloque perteneciente al camino crítico. Si una de ellas es interna en el bloque y la otra se encuentra situada en cualquiera de los extremos del camino crítico la inversión del arco que las une ($w \rightarrow v$) da lugar a otra solución factible S' tal que $C_{max}(S') \geq C_{max}(S)$.

3.1. Recocido simulado

Desde la década de los 80 el algoritmo de recocido simulado (SA, *Simulated Annealing*), desarrollado por Kirkpatrick (1984), ha sido empleado para resolver una gran variedad de problemas de optimización complejos, tanto basados en variables continuas como discretas. Entre éstos se pueden incluir el diseño de bases de datos distribuidas, configuración de aplicaciones multiprocesador, diseño de filtros digitales, problemas electrostáticos y análisis de imágenes.

Típicamente, el algoritmo comienza con la generación de una solución aleatoria para el problema, y el cálculo del coste de dicha solución. El algoritmo evoluciona realizando desplazamientos unitarios que son aceptados si su coste es inferior a la solución anterior. En caso contrario la solución es aceptada con una probabilidad que decrece exponencialmente con el cociente de la diferencia de costes ΔC y un parámetro T que representa la temperatura. El hecho de que la probabilidad de aceptar soluciones peores no sea nula permite que el algoritmo no quede atrapado en mínimos locales. Típicamente, si la nueva solución no fuera aceptada, se probaría un nuevo movimiento desde la solución anterior.

A lo largo de una ejecución, la temperatura T va disminuyendo de forma geométrica, reduciéndose así la probabilidad de desplazamientos ascendentes (aquellos que aumentan el coste) y manteniendo a la solución en valores óptimos. El algoritmo termina cuando no se ha obtenido ninguna mejora en un número determinado de desplazamientos aunque se han empleado criterios de finalización más complicados. Van Laarhoven, Aarts et al. (1992) demuestra que, en las condiciones apropiadas, este algoritmo converge a un mínimo global.

Van Laarhoven, Aarts et al. (1992) emplean el algoritmo de recocido simulado con un vecindario obtenido mediante la inversión de los arcos entre operaciones adyacentes que se ejecutan en la misma máquina y pertenecen al camino crítico, aprovechando así la propiedad 1 que indica que las soluciones así generadas siempre serán factibles y la propiedad 2, que indica que son las más prometedoras.

Matsuo, Chang Juck Suh et al. (1989) elimina de dicho vecindario las inversiones de arcos entre operaciones internas en un bloque, que según la propiedad 3, dan lugar a soluciones menos prometedoras. Además, crea una variante basada en mejorar las soluciones rechazadas por el criterio del recocido simulado mediante un algoritmo de mejora iterativa (*steepest descent*) hasta que se alcance un mínimo local. Si la solución encontrada es mejor que la que se ha encontrado hasta el momento, se guarda, en caso contrario ésta resulta eliminada.

3.2. Búsqueda tabú

La búsqueda tabú (TS, *Taboo search*) es un procedimiento metaheurístico diseñado para encontrar soluciones subóptimas en problemas de optimización combinatorial. Fue diseñado inicialmente por Glover (1990) y ha sido aplicado a varios problemas de optimización combinatoria como programación de horarios y rutas y secuenciación de tareas.

Al igual que el procedimiento de recocido simulado, se trata de un algoritmo de búsqueda local. Para cualquier solución se genera un conjunto de desplazamientos que definen su vecindario, y se selecciona el mejor de ellos para la siguiente iteración. Para evitar que el procedimiento iterativo entre en un bucle y que se quede estancado en óptimos locales, se mantiene en memoria el histórico de la búsqueda. Se suele distinguir dos tipos de memoria: a *corto plazo* para los movimientos más recientes y a *largo plazo* para la parte de la búsqueda más distante. La memoria a corto plazo esta formada por una *lista tabú* que contiene información sobre las últimas soluciones visitadas, de modo que desplazamientos hacia soluciones vecinas que se correspondan con la lista tabú se prohiban. Típicamente, la lista tabú *T* consiste en una lista FILO (First In First Out) de longitud definida que contiene los últimos desplazamientos unitarios prohibidos, de modo que en cada iteración se introduce el desplazamiento inverso al realizado al principio de la lista *T* y se elimina el desplazamiento que se encuentra al final.

El algoritmo de búsqueda tabú aplicada al JSP de Taillard (1994) genera vecinos mediante la inversión de arcos que unen tareas adyacentes en el camino crítico. Después de que un arco $(v \rightarrow w)$ ha sido invertido, se introduce el paso inverso $(w \rightarrow v)$ en la lista tabú, consistente en una lista de longitud definida que contiene los desplazamientos prohibidos. Cada vez que se realiza un número *n* de iteraciones, se modifica aleatoriamente la longitud de la lista tabú. En cada iteración se evalúan todas las soluciones del vecindario que no se encuentran en la lista tabú, calculando para cada una el valor de C_{max} y seleccionándose la más prometedora.

Barnes y Chambers (1995) emplea el mismo mecanismo para generar soluciones cercanas, sin embargo la longitud de su lista tabú tiene una longitud determinada. Además, parten de la mejor solución obtenida mediante algoritmos heurísticos más sencillos.

Nowicki y Smutnicki (1996) emplea el mismo vecindario eliminando las opciones obtenidas invirtiendo el arco entre dos operaciones adyacentes e internas o tales que siendo una de ellas interna, la otra se encuentra en uno de los extremos del camino crítico. Esto es debido a que la propiedad 3 explicada en la sección 3 nos garantiza que dichas soluciones son poco prometedoras. La aportación fundamental de ésta implementación es que define un procedimiento para los casos en los que el algoritmo deba retroceder a soluciones anteriormente exploradas para seguir buscando otras soluciones del vecindario (*backtracking*).

En la Figura 5 se muestra una comparativa de los algoritmos de recocido simulado y búsqueda tabú extraída de Vaessens, Aarts et al. (1996). Se muestran los resultados de C_{max} que los algoritmos propuestos en la bibliografía obtienen para algunos problemas publicados en la biblioteca OR (Beasley,1990). Obsérvese que generalmente los resultados obtenidos por los algoritmos de búsqueda tabú son mejores que aquéllos obtenidos mediante recocido simulado, tanto respecto a los tiempos de cálculo como respecto al valor de C_{max}. Además, dentro de los algoritmos de búsqueda tabú, el de Nowicki y Smutnicki (1996) es el más prometedor debido a la implantación del *backtracking*.

	FT10		LA21		LA29		LA38	
tamaño	10x10		15x10		20x10		15x15	
mínimo	930*		1046*		1153		1196*	
	resultado	tiempo(s)	resultado	tiempo(s)	resultado	tiempo(s)	resultado	tiempo(s)
Recocido simulado								
Van Laarhoven et al. (1992)								
promedio 5 ejecuciones	969	99	1083	243	1235	636	1256	597
Matsuo et al. (1989)								
1 ejecución	946	987	1071	205	1231	672	1235	603
Búsqueda tabú								
Taillard (1994)								
mejor de 5	930		1047		1170		1202	
Barnes y Chambers (1995)								
mejor de 2	930	450	1050	480	1194	600	1211	540
Nowicki y Smutnicki (1996)								
1 ejecución	930	30	1055	21	1164	493	1209	165
mejor de 3	930		1047		1160		1196	

* La solución es óptima

Figura 5. Comparativa algoritmos de búsqueda local

4. Resultados experimentales

Paralelamente a la revisión bibliográfica referente a métodos de resolución del JSSP se han desarrollado implantaciones informáticas basadas en algoritmos genéticos (AG) y recocido simulado (SA).

Para la búsqueda mediante algoritmos genéticos se ha empleado la librería GAlib (Wall, 1996) que contiene código genérico para los algoritmos genéticos. Únicamente ha sido necesario implementar las funciones de cruce, mutación y cálculo de la bondad de cromosomas, al ser estas específicas para el JSSP. Las soluciones se representan en forma de secuencia de números de trabajo sin particiones, de modo que no haya permutaciones infactibles. Se han empleado los operadores de cruce y mutación PPX y OBM, respectivamente.

En la implantación del algoritmo de recocido simulado se ha empleado el algoritmo genérico de Stiles y Lee (2003), que controla la evolución de la temperatura a lo largo de la ejecución y obtiene automáticamente resultados estadísticos. Sin embargo, ha sido necesario implantar la función que genera el vecindario. Ésta se ha obtenido (a partir de una solución dada) invirtiendo arcos entre tareas adyacentes que se encuentran en el camino crítico.

La Figura 6 muestra los resultados obtenidos al ejecutar ambos algoritmos en problemas de tamaño creciente que abarcan desde 15 x 15 hasta 100 x 20. Las curvas superiores muestran el valor de C_{max} encontrado por ambos métodos (C_{max}-*GA* y *Cmax-SA*) y la mejor solución encontrada para el problema hasta el momento (C_{max}-*OPT*). Debajo se muestra el error relativo correspondiente, que suele encontrase en un entorno alrededor del 10%. Obsérvese que en la mitad de los 8 problemas probados, el resultado obtenido por la implantación basada en AG es mejor que el obtenido por la basada en SA por lo que no puede deducirse que un procedimiento sea siempre mejor que el otro. Sin embargo, destaca el 20% de error obtenido por AG en el problema de 30 x 20, lo que nos lleva a pensar que la basada en SA es la más robusta.

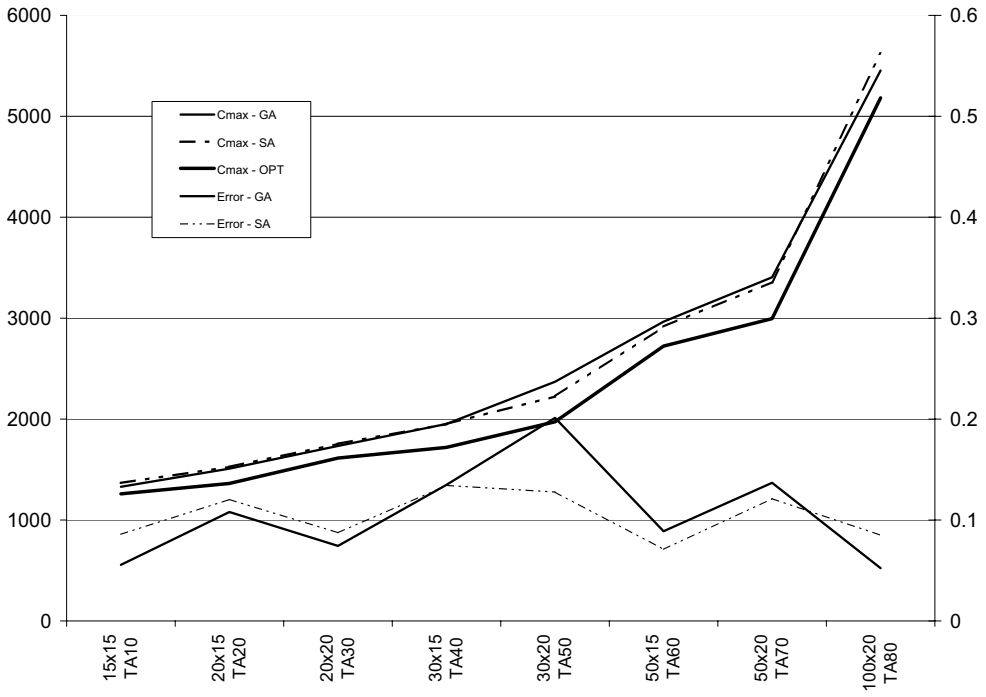

Figura 6. Comparativa entre AG y SA

Agradecimientos

Agradecemos su colaboración a la Ingeniera en Organización Industrial Dª Beatriz Pacheco Aparicio, por su aportación al desarrollo del código de GA y por las ejecuciones de casos realizadas.

Referencias

Barnes, J. W. y J. B. Chambers (1995). " Solving the job shop scheduling problem with tabu search " IIE Transactions **27**(2): 257-63

Beasley, J. E. (1990). Collection of test data sets for a variety of Operations Research (OR) problems. http://www.ms.ic.ac.uk/info.html

Bierwirth, C. (1995). "A generalized permutation approach to job shop scheduling with genetic algorithms " OR Spektrum, p. 87-92 **17**: 2-3.

Croce, F., R. Tadei, et al. (1995). "A genetic algorithm for the job shop problem? ." Computers & Operations Research **22**(1): 15-24

Dorndorf, U. y E. Pesch (1995). "Evolution based learning in a job shop scheduling environment. ." Computers & Operations Research

Falkenauer, E. y S. Bouffouix (1991). A genetic algorithm for job shop IEEE International Conference on Robotics and Automation, Los Alamitos, CA, USA IEEE Comput. Soc

Fang, H.-L., P. Ross, et al. (1993). A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems. Fifth International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufmann.

Giffler, B. y G. L. Thompson (1960). "algorithms for Solving Production Scheduling Problems. ." Operations Research

Glover, F. (1990). "Tabu search: a tutorial " Interfaces **20**(4): 74-94

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems, Ann Arbor.

Holsapple, C. W., V. S. Jacob, et al. (1993). "A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts " IEEE Transactions on Systems, Man and Cybernetics **23**(4): 953-72
Kirkpatrick, S. (1984). "Optimization by simulated annealing: quantitative studies " Journal of Statistical Physics **34**: 5-6.
Lee, F. H. y G. S. Stiles (1989). Parallel simulated annealing: several approaches the North American Transputer Users Group, Durham, NC, USA, 18-19 Oct, ios
Matsuo, H., Chang Juck Suh, et al. (1989). " A controlled search simulated annealing method for the single machine weighted tardiness problem " Annals of Operations Research **21**(1-4): 85-108
Nakano, R. y T. Yamada (1991). Conventional genetic algorithm for job-shop problems. International Conference on Genetic Algorithms.
Nowicki, E. y C. Smutnicki (1996). "A fast taboo search algorithm for the job shop problem. ." Management Science
Reeves, C. R. (1995). " A genetic algorithm for flowshop sequencing " Computers \& Operations Research **22**(1): 5-13
Roy, B. y B. Sussman (1964). "Les problèmes d'ordonnancement avec contraintes disjonctives." SEMA.
Stiles, G. S. y F. H. Lee (2003). Robust Simulated Annealing.
http://www.engineering.usu.edu/ece/faculty/Gstiles/index.html
Taillard, E. D. (1994). "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. ." Journal on Computing
Vaessens, R. J. M., E. H. L. Aarts, et al. (1996). "Job shop scheduling by local search " INFORMS Journal on Computing **8**(3): 302-17
Van Laarhoven, P. J. M., E. H. L. Aarts, et al. (1992). "Job Shop Scheduling by Simulated Annealing. ." Operations Research
Wall, M. (1996). GAlib: A C++ Library of Genetic Algorithm Components.
http://lancet.mit.edu/ga/
Yamada, T. y R. Nakano (1992). "A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems." Parallel Problem Solving from Nature **2**: 281-290.