Nuevas Tendencias y Retos en Métodos Heurísticos para Problemas de Scheduling

Ramiro Varela

Grupo de Tecnologías de la Computación.
Departamento de Informática. Centro de Inteligencia Artificial
Universidad de Oviedo. Campus de Viesques, 33271 Gijón, España
Tel. +34-8-5182032. FAX +34-8-5182125.
ramiro@uniovi.es
http://www.aic.uniovi.es/Tc

Abstract. En este artículo consideramos el problema Job Shop Scheduling. En principio hacemos una revisión de la versión del problema con minimización del makespan y de algunos métodos clásicos de resolución como los algoritmos de búsqueda en espacios de estados y los algoritmos evolutivos. A continuación comentamos algunas de las variantes que son objeto de investigación intensiva actualmente.

Keywords Job Shop Scheduling, Heurísticos, Algoritmos Genéticos, Búsqueda en Espacios de Estados, Búsqueda Local

1 Introducción

El Job Shop Scheduling Problem (JSSP) es un paradigma de la familia de problemas de optimización combinatoria que ha interesado a los investigadores durante las últimas décadas. Tradicionalmente, el criterio de optimización es el makespan; para esta versión del problema se han desarrollado numerosos métodos exactos y aproximados, la mayor parte de los cuales se fundamentan en el concepto de *camino crítico*. El método exacto más relevante es el algoritmo de ramificación y poda propuesto por Brucker y otros en [1,2], desarrollado a partir de conceptos y técnicas propuestas también por otros investigadores como Carlier y Pinson [3,4]. Entre los métodos no exactos destacan las técnicas de búsqueda local que utilizan técnicas de vecindad definidas en principio por Dell' Amico y Trubian [5] y desarrolladas por otros investigadores. Normalmente estas técnicas se combinan con otras metaheurísticas como los algoritmos genéticos [6,7] o la búsqueda tabu [8]. Las técnicas basadas en el camino crítico normalmente no son muy eficientes para criterios de optimización diferentes del makespan. Por ejemplo, en [9] se define un esquema de vecindad para el problema JSSP con minimización del tiempo de flujo total que requiere calcular múltiples caminos críticos, dando lugar a un gran número de vecinos para cada solución. En este trabajo comenzamos con el JSSP con minimización del makespan y a continuación introducimos algunas de las variantes con gran interés actual como son el JSSP con minimización del tiempo de flujo total, el JSSP con tiempos de setup y el JSSP con operaciones de duraciones imprecisas. Para estos problemas consideramos la aplicación de algunos métodos exactos, como los algoritmos de ramificación y poda o primero el mejor, y también aproximados, como los algoritmos genéticos y la búsqueda local.

M.A Cruz-Chávez, J.C. Zavala-Díaz (Eds): CICos 2008, ISBN:978-607-00-0165-9. pp. 1-10, 2008

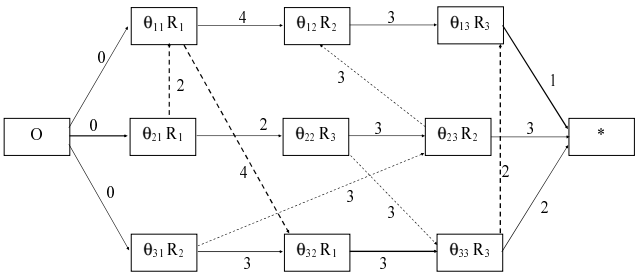

Fig. 1. Grafo solución para una instancia de tamaño 3×3

2 Formulación del Problema

El JSSP requiere planificar un conjunto de N trabajos $\{J_1, \ldots, J_N\}$ sobre un conjunto de M máquinas $\{R_1, \ldots, R_M\}$. Cada trabajo J_i consiste en una serie de operaciones $\{\theta_{i1}, \ldots, \theta_{iM}\}$ que deben ser procesadas secuencialmente. Cada tarea θ_{il} requiere el uso de una máquina $R_{\theta_{il}}$, tiene una duración $p_{\theta_{il}}$ y un tiempo de comienzo $st_{\theta_{il}}$ que se debe determinar. El problema tiene tres restricciones: precedencia, capacidad y no-interrupción. Las restricciones de precedencia se expresan de la forma $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$ e indican que las tareas de cada trabajo se ejecutarán secuencialmente. Las restricciones de capacidad son disyunciones de la forma: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, y expresan que una misma máquina no puede ser compartida de forma simultánea por dos tareas. Por último, la restricción de no-interrupción indica que las tareas deben tener asignada la máquina de forma continua durante su período de procesamiento. El objetivo es encontrar una planificación (schedule) factible, es decir que satisfaga todas las restricciones, para la que el makespan (tiempo de finalización de la última tarea) sea mínimo. Este problema se denota $J//C_{max}$ de acuerdo con la notación $\alpha/\beta/\gamma$ que es muy habitual en la literatura.

Una solución factible se representa mediante un grafo G_s que indica el orden de procesamiento de las tareas. Este grafo incluye tantos nodos como tareas tiene el problema, más dos correspondientes a tareas ficticias O y $*$ que tienen duración nula. Cada arco indica que la tarea destino puede comenzar tan pronto como la tarea origen finalice. El coste de cada arco es la duración de la tarea en el nodo origen. Con esta representación, el makespan es el coste del camino más largo entre la tarea O que precede a la primera tarea de cada trabajo y la tarea $*$ que es sucesora de la última tarea de cada trabajo. Este camino se denomina *camino crítico* y a una subsecuencia maximal de nodos en este camino correspondiente a tareas que requieren la misma máquina se le denomina *bloque crítico*. La Figura 1 muestra una solución para un problema de tamaño 3×3 (3 trabajos y 3 máquinas). El camino crítico viene dado por la secuencia $\theta_{21}, \theta_{11}, \theta_{32}, \theta_{33}, \theta_{13}$; tiene dos bloques críticos y su makespan es 12.

3 Espacios de Búsqueda para el JSSP

En la búsqueda de soluciones para problemas de scheduling, se suelen considerar tres tipos de planificaciones: *semi-activas*, *activas* y *densas* o *non-delay*. En las planificaciones semi-activas, para que una tarea pueda comenzar a procesarse en un tiempo más

temprano al que tiene asignado, es preciso intercambiar el orden de al menos dos tareas. En una planificación activa es necesario retrasar el instante de comienzo de al menos otra tarea para poder adelantar la ejecución de una tarea. Y las planificaciones densas se caracterizan por el hecho de que nunca hay una máquina inactiva en un instante en el que una tarea está en condiciones de procesarse en esa máquina. Las planificaciones densas son un subconjunto de las planificaciones activas, y éstas a su vez lo son de las planificaciones activas. De estos tres espacios, los únicos que son dominantes para las funciones objetivo convencionales, es decir que contienen al menos una solución óptima, son los espacios de planificaciones activas y semi-activas. No obstante, para instancias grandes para las que no es razonable buscar soluciones óptimas, suele resultar más eficiente la búsqueda en un subconjunto de las planificaciones activas, como el espacio de las planificaciones densas.

4 Métodos Heurísticos para el JSSP

El JSSP es un paradigma de la familia de problemas de satisfacción de restricciones con optimización para el que se han aplicado una gran variedad de métodos heurísticos. En esta sección revisaremos algunos de estos métodos.

4.1 Algoritmos Voraces

Los algoritmos voraces (greedy) han sido muy utilizados para resolver problemas de scheduling. Entre este tipo de algoritmos se encuentran las populares reglas de prioridad (dispatching rules). Estas reglas se utilizan para seleccionar el siguiente trabajo a procesar entre un conjunto de trabajos que están listos para ser procesados. Posiblemente la regla más utilizada sea la SPT (Shortest Processing Time). Con esta regla se selecciona el trabajo con menos tiempo de procesamiento sobre su máquina. Otra regla es la LWKR (Least WorK Remaining) que selecciona el trabajo con menos tiempo de procesamiento en el resto de las máquinas. En [10] se describen varias reglas de prioridad y se propone un método basado en redes neuronales para seleccionar la mejor regla para cada una de las máquinas de un problema JSSP. La gran ventaja de las reglas de prioridad es que su aplicación es muy poco costosa computacionalemte, pero como contrapartida suelen proporcionar resultados de calidad baja.

Dentro de los algoritmos voraces tiene una especial importancia el algoritmo $G\&T$ (véase Algoritmo 1) propuesto por Giffler y Thomson en [11]. Se trata de un algoritmo no-determinista que calcula planificaciones activas. Mediante el parámetro δ se controla el espacio de búsqueda del algoritmo. Cuando $\delta = 1$ se explora el espacio completo de las planificaciones activas, y cuando $\delta = 0$ la búsqueda se restringe al espacio de las planificaciones densas. Por su naturaleza no-determinista, el algoritmo $G\&T$ se puede adaptar a muchas situaciones diferentes. Por ejemplo se puede utilizar como algoritmo de decodificación dentro de un algoritmo genético, o bien si se condideran todas las opciones en el momento de la selección no-determinista (paso 8 en el Algoritmo 1) se obtiene un esquema de ramificación apropiado para un algoritmo de búsqueda en espacios de estados.

4.2 Algoritmos Genéticos

Los Algoritmos Genéticos (AGs) [12], [13], [11], [6], [14] constituyen un método muy prometedor debido en parte a la facilidad con la que estos algoritmos se pueden combinar con otras técnicas como la búsqueda local. Además, los AGs permiten explotar

Algorithm 1 Algoritmo G&T hibrido.

1: $A = \{\theta_{j0}, 0 \leq j < N\}$; {$N$ es el número de trabajos}
2: **while** $A \neq \emptyset$ **do**
3: Sea st_θ el tiempo de inicio más temprano si θ se planifica a continuación;
4: $\theta_1 = argmin\{st_\theta + p_\theta, \theta \in A\}$;
5: $B = \{\theta \in A : R_\theta = R_{\theta_1}, st_\theta < st_{\theta_1} + du_{\theta_1}\}$;
6: $\theta_2 = argmin\{st_\theta, \theta \in B\}$; {el tiempo de inicio más temprano de cualquier operación en B está en $[st\theta_2, st\theta_1 + du\theta_1]$}
7: Reducción de B: $B = \{\theta \in B : st_\theta \leq st_{\theta_2} + \delta((st_{\theta_1} + du_{\theta_1}) - st_{\theta_2}), \delta \in [0, 1]\}$; {el intervalo se reduce a $[st_{\theta_2}, st_{\theta_2} + \delta((st_{\theta_1} + du_{\theta_1}) - st_{\theta_2})]$}
8: Elegir $\theta^\star \in B$ con algún criterio y planificarla en el instante $st_{\theta^\star}$;
9: $A = A \setminus \{\theta^\star\} \cup \{SUC(\theta^\star)\}$; {$SUC(\theta)$ es la sucesora de θ en su trabajo, si existe}
10: **end while**

cualquier clase de conocimiento heurístico sobre el dominio del problema. De este modo son realmente competitivos con los mejores métodos para resolver el JSSP.

En principio consideramos un AG convencional con las siguientes características. La codificación de cromosomas se hace con el método de permutaciones con repetición propuesto por Bierwirth en [12]: un cromosoma es una permutación del conjunto de tareas, en la que cada tarea viene representada por el número de su trabajo. Por ejemplo, el cromosoma *(2 1 1 3 2 3 1 2 3)* representa la permutación de operaciones $(\theta_{21}, \theta_{11}, \theta_{12}, \theta_{31}, \theta_{22}, \theta_{32}, \theta_{13}, \theta_{23}, \theta_{33})$. Esta permutación expresa un orden de ejecución para las tareas de cada máquina, siendo los ordenes correspondientes a dos máquinas compatibles entre si. El esquema de permutaciones con repetición presenta una serie de características que lo hacen realmente interesante. Es un esquema que resulta fácil de evaluar con distintos algoritmos de decodificación y permite también el uso de diferentes operadores genéticos de cruce y mutación. Además, tiene una tendencia a representar ordenes naturales entre las tareas, de modo que es más eficiente que otros esquemas basados en permutaciones, como por ejemplo el clásico esquema de permutaciones (sin repetición). En [15] se hace un estudio de estos y otros esquemas y la conclusión es que el esquema de permutaciones con repetición es el que produce una convergencia más adecuada del algoritmo genético.

Como algoritmo de evaluación elegimos el algoritmo $G\&T$ descrito en la sección anterior. Así, el proceso de decodificación consiste en ejecutar este algoritmo y elegir la tarea $\theta^\star$ que se encuentra más a la izquierda en el cromosoma. De este modo se respeta el orden de tareas del cromosoma, siempre y cuando permita generar una planificación activa. En la fase de selección, los cromosomas se organizan en pares de forma aleatoria. Cada uno de estos pares se cruza y/o muta de acuerdo con las probabilidades de cruce y mutación. De acuerdo con nuestra experiencia, el mejor operador de cruce es el JOX (Job Order Crossover) descrito en [12]: dados dos padres, JOX selecciona un subconjunto aleatorio de trabajos y copia los genes correspondientes del primer padre al hijo, el resto de los genes se toman del segundo padre, manteniendo su orden relativo. La mutación consiste en intercambiar dos genes consecutivos elegidos aleatoriamente. Finalmente, la aceptación de cromosomas para la población resultante se hace por torneo entre cada par de padres y sus dos hijos (es decir se eligen los dos mejores de los cuatro).

4.3 Búsqueda Local

En muchas ocasiones, la eficiencia de un AG se puede mejorar si se combina con un método de búsqueda local. Estos métodos se basan en definir una vecindad para cada solución (cromosoma) mediante una regla de transformación simple. Un cromosoma se reemplaza por uno de sus vecinos si éste cumple el criterio de aceptación que se utilice. El proceso se inicia con cada cromosoma generado por los operadores de cruce y mutación, y continúa durante un número de iteraciones o cuando ningún vecino cumple el criterio de aceptación. En este trabajo consideramos los esquemas de vecindad denominados N_1, N_2 y N_3 in [6]. En los tres casos las reglas de transformación consisten en hacer cambios en el camino crítico. La factibilidad de los vecinos se realiza mediante un algoritmo eficiente a costa de descartar algunos vecinos válidos y el criterio de aceptación se basa en estimaciones del makespan resultante. Los esquemas de vecindad se definen del siguiente modo.

Definition 1 (N_1). *Sean v y w dos operaciones que se encuentran al principio de un bloque crítico, entonces se genera un vecino intercambiando el orden de estas operaciones salvo que el bloque crítico sea el primero del camino crítico. Análogamente, si las operaciones v y w se encuentran al final de un bloque crítico que no es el último del camino crítico.*

Definition 2 (N_2). *Si la operación v pertenece a un bloque b de la forma $b = (b'vb'')$. En una solución vecina, v se mueve hasta la posición factible más próxima al principio de b (o al final).*

Definition 3 (N_3). *Si (PM_v, v, w) las tres últimas operaciones de un bloque crítico, todas las permutaciones de $\{PM_v, v, w\}$ en las que (v, w) se invierte, se consideran vecinos. Análogamente, si (v, w, SM_w) son las tres primeras operaciones de un bloque crítico. Si el bloque crítico solo tiene dos tareas, se invierten para generar un vecino.*

4.4 Búsqueda Heurística en Espacios de Estados

Entre este tipo de algoritmos de búsqueda, el más popular es el algoritmo de ramificación y poda propuesto pro Brucker en [1]. Este algoritmo fue desarrollado a partir de ideas propuestas también por otros autores [3,4] sobre criterios de ramificación y cálculo de cotas inferiores. El esquema de ramificación es muy similar al modo de generar los vecinos en los esquemas comentados en la sección anterior. Concretamente, en cada estado de la búsqueda hay un subconjunto de arcos disyuntivos fijados (inicialmente ninguno). El primer paso consiste en generar una solución mediante un algoritmo voraz inspirado en el algoritmo $G\&T$. De esta solución se obtiene un camino crítico a partir del cual se generan los sucesores del estado actual: para cada bloque crítico B se recorren todas sus tareas, y cada tarea t se mueve al principio y al final de B fijando los arcos correspondientes a este movimiento en el estado sucesor. El algoritmo se complementa con un método de propagación de restricciones denominado *selección inmediata* que permite fijar un número importante de arcos adicionales y con un método de cálculo de cotas inferiores basado en relajaciones a problemas de secuenciamiento de una máquina y en eliminar la restricción de no-interrupción de la máquina. El algoritmo de Brucker es muy eficiente ya que permite resolver de forma exacta problemas de tamaño 10×10 e incluso superiores. Pero tiene el inconveniente de que no se puede generalizar, de forma simple, para resolver el problema JSSP con funciones objetivo distintas de la

minimización del makespan, ya que el espacio de búsqueda que recorre solamente es dominante para esta función objetivo.

Un alternativa al esquema de ramificación del algoritmo de Brucker es el que se deriva del algoritmo $G\&T$ si consideramos todas las opciones posibles en el paso 8 del algoritmo 1. En este caso el conjunto de soluciones alcanzables se corresponde con el espacio de planificaciones activas que es dominante para las funciones objetivo clásicas como el makespan, el flujo total o el retardo total. Este espacio se puede recorrer con algoritmos de ramificación y poda, o bien con algoritmos tipo El Mejor Primero (Best First) como por ejemplo el algoritmo A* [16]. Estos algoritmos son claramente menos eficientes que el algoritmo de Brucker para minimizar el makespan, pero cuando se trata de minimizar otras funciones objetivo, como por ejemplo el tiempo de flujo total, son competitivos con cualquier otro método exacto, o incluso con algunos métodos aproximados como por ejemplo los heurísticos del tipo Shifting Bottleneck y los métodos de búsqueda local, tal y como veremos a continuación en la sección 5.1.

5 Algunos Tópicos Actuales

Sin ánimo de ser exhaustivos, en esta sección consideramos algunas variantes del problema JSSP que tienen un gran interés actual. En primer lugar consideramos el JSSP con minimización del flujo total como función objetivo. A continuación el JSSP con minimización del makespan y tiempos de setup dependientes de la secuencia de operaciones. Y, finalmente, el JSSP con operaciones de duraciones imprecisas.

5.1 JSSP con minimización del tiempo de flujo total

El tiempo de flujo total es la suma de los tiempos de finalización de todos los trabajos. Cuando se considera esta función objetivo, el problema JSSP se denota como $J//\sum C_i$. Aunque se trata de una versión muy parecida al clásico problema $J//C_{max}$, en realidad el problema $J//\sum C_i$ es más difícil de resolver. Probablemente esto sea debido a que, en general, una instancia del problema $J//\sum C_i$ tiene menos soluciones óptimas que la correspondiente instancia del problema $J//\sum C_i$, por lo que para un AG resulta más difícil encontrar una de estas soluciones. Además, los métodos polinomiales de cálculo de cotas inferiores también son menos precisos para el $J//\sum C_i$, con lo que los algoritmos de búsqueda en espacios de estados tienen que visitar un mayor número de estados para encontrar una solución óptima. Para ilustrar la conjetura anterior, vamos a considerar un pequeño banco de ejemplos: las 5 instancias denominadas $LA01..LA05$. Se trata de problemas de tamaño 10×5 y se pueden descargar del repositorio $OR-library$ (http://people.brunel.ac.uk/ mastjjb/jeb/info.html). Cuando se considera el makespan como función de optimización, estas instancias son muy fáciles de resolver. Tanto el AG descrito en la sección 4.2, como los métodos de búsqueda comentados en la sección 4.4 encuentran la solución óptima en un tiempo muy pequeño. Sin embargo, cuando se trata de minimizar el tiempo de flujo total, estos problemas ya no son triviales. La Tabla 1 muestra los resultados del AG anterior sobre estos problemas. Para cada instancia se muestra la mejor solución encontrada en 30 ejecuciones, el valor medio de las 30 soluciones, el tiempo de cada ejecución y el número de veces que se encuentra la solución óptima en las 30 ejecuciones. Para el makespan se encuentra la solución óptima las 30 veces, mientras que esto no ocurre cuando la función objetivo es el tiempo de flujo total.

Table 1. Resultados del AG sobre las instancias $LA01 - LA05$.

	Makespan			Tiempo de Flujo Total			
Instancia	Mejor	Media	Tiempo(s)	Mejor	Media	Tiempo(s)	Nro. Op.
$LA01$	666	666	8	4832	4860	8	1
$LA02$	655	655	8	4459	4498	8	16
$LA03$	597	597	8	4151	4187	8	8
$LA04$	590	590	8	4259	4357	8	4
$LA05$	593	593	8	4072	4108	8	1

Para el problema $J//\sum C_i$ no es fácil adaptar los métodos basados en el camino crítico, como el algoritmo de ramificación y poda propuesto en [1,2] o los métodos de búsqueda local comentados en la sección 4.3, que resultan tan eficientes en el caso del problema $J//C_{max}$. Esto se debe a que para el problema $J//\sum C_i$ hay que considerar no uno sino varios caminos críticos, tantos como el número de trabajos N, con lo que el factor de ramificación de los algoritmos de búsqueda, o el número de vecinos de los esquemas de vecindad es excesivamente grande. Para el problema $J//\sum C_i$, en [17] se propone un algoritmo de búsqueda tipo A^* [16,18] que es capaz de resolver las instancias anteriores de forma más eficiente que el AG. Se trata de un algoritmo que utiliza un heurístico basado relajar el problema a instancias de problemas con una sola máquina y minimización de tiempo de retardo total (tardiness). Estas instancias se resuelven a su vez de forma aproximada mediante relajaciones de la restricción de no interrupción de las máquinas tal y como se describe en [19]. El algoritmo incorpora además un método de poda por dominancia que permite reducir el número de nodos expandidos casi en un orden de magnitud. Este método consiste en comparar cada nodo seleccionado para expansión con un subconjunto de los nodos ya expandidos para verificar la relación de dominancia. En muchas ocasiones, se puede determinar que la mejor solución alcanzable desde el nodo que se va a expandir no es mejor que aquella que se puede alcanzar desde otro nodo ya expandido, con lo cual el nodo a expandir se puede eliminar. En [17] se da una condición suficiente de dominancia que permite aplicar este método de manera eficiente. La Tabla 2 muestra los resultados de este método sobre los problemas $LA01..LA05$. Como se puede observar en este caso, si no se utiliza el método de poda, solamente se resuelven 2 instancias y en las otras 3 la memoria resulta insuficiente. Sin embargo, al utilizar el método de poda por dominancia, se resuelven las 5 instancias y en las 2 que se resuelven sin la poda, tanto el tiempo como el número de nodos expandidos y generados disminuyen de forma notable.

Es algoritmo de búsqueda no es capaz de resolver de forma exacta problemas de mayor tamaño, por ejemplo problemas de tamaños 15×5 o 10×10. Para poder hacerlo, serán precisos mejores métodos polinomiales de estimación de cotas inferiores o métodos de poda más efectivos.

5.2 JSSP con tiempos de setup

En esta versión del problema es necesario un tiempo de puesta en marcha de la máquina k cuando ésta conmuta entre dos trabajos i y j. Este tiempo se denota como S_{ij}^k, y el problema se denota como $J/S_{ij}^k/C_{max}$. En [20] y en [21] se proponen sendos algoritmos

Table 2. Resumen de resultados con búsqueda tipo A^* para los problemas $LA01 - LA05$. El tiempo límite es de 3600s.

	Sin Poda			Poda por Dominancia		
Instancia	Generated	Expanded	T.(s)	Generated	Expanded	T.(s)
$LA01$	**1100057**	**467735**	**284**	248935	107955	116
$LA02$	**1030746**	**398422**	**266**	520356	217751	255
$LA03$	392860	154791	102	79883	32118	34
$LA04$	855500	345005	218	136928	57558	60
$LA05$	**1058615**	**412658**	**276**	412277	174320	210

*Los valores en **negrita** indican que la memoria o el tiempo límite se han agotado.*

de ramificación y poda para este problema. Estos algoritmos se diferencian en el esquema de ramificación y en el método de cálculo de cotas inferiores. El primero utiliza una extensión de los métodos propuestos en [1], mientras que el segundo utiliza un método de ramificación que genera planificaciones semi-activas y un método de cálculo de cotas inferiores que se basa en relajaciones a instancias del problema TSP (Travelling Salesman Problem). Estas instancias se resuelven de forma exacta y, a pesar de que no son de gran tamaño, hacen que el algoritmo sea muy costoso computacionalmente. Entre los métodos no exactos, podemos citar el método basado en el conocido heurístico Shifting Bottleneck propuesto en [22] y el algoritmo genético combinado con búsqueda local propuesto en [23]. En el caso del método propuesto en [23], tanto el algoritmo genético como el método de búsqueda local son extensiones de los métodos descritos en las secciones 4.2 y 4.3 respectivamente. El algoritmo genético solamente se diferencia en el esquema de decodificación, en lugar de utilizar el algoritmo $G\&T$, se utiliza el método $EG\&T$ propuesto en [24] que, a pesar de no ser dominante, resulta muy eficiente. El método de búsqueda local se diferencia del anterior principalmente en el primero de los esquemas de vecindad, que ahora se define del siguiente modo, ya que a diferencia del problema $J//C_{max}$, en el problema $J/S_{ij}^k/C_{max}$, los intercambios en el interior de un camino crítico también pueden producir mejoras.

Definition 4 (N_1^S). *Sean v y w dos operaciones que se encuentran un bloque crítico, entonces se genera un vecino intercambiando el orden de estas operaciones.*

En [23] se muestra un resumen de los resultados obtenidos con los cuatro métodos comentados sobre el banco de ejemplos propuesto en [20], en que se puede observar que el método propuesto en [21] proporciona las mejores cotas inferiores, mientras que el método propuesto en [23] es, en general, el que proporciona las mejores cotas superiores.

El problema $J/S_{ij}^k/C_{max}$ tiene un gran interés actual y en consecuencia debe ser objeto de nuevas investigaciones. Algunos de los métodos que se han aplicado con éxito al problema $J//C_{max}$ aun no han sido extendidos para este problema, como por ejemplo el método de búsqueda tabu propuesto en [8]. Asimismo, si se consideran nuevas funciones objetivo, como el tiempo de flujo total o el tiempo de retardo, el problema es todavía más completo y en consecuencia supone un nuevo reto para los investigadores.

5.3 JSSP con duraciones imprecisas

Otra variante de gran interés actual es el problema JSSP con duraciones imprecisas. Un modelo habitual para representar esta imprecisión son los números borrosos (fuzzy numbers) en particular los números borrosos triangulares o TFNs. Una ventaja de los TFNs es que las operaciones de suma y máximo son fáciles de realizar. Un TNF viene dado por una terna de números reales (p^1, p^2, p^3) que expresan respectivamente los valores mínimo, más probable y máximo, que la duración de una tarea puede tener, y que solamente se conocerá con exactitud una vez que la tarea haya terminado. Los algoritmos que hemos visto anteriormente se pueden extender para tratar este problema y en este caso las soluciones que ofrecen son planificaciones borrosas (fuzzy schedules), cuyo makespan es incierto y viene dado también por un número triangular. La Figura 2 muestra un ejemplo de solución, en este caso en forma de gráfico de Gantt adaptado a TFNs siguiendo a [25], para una instancia de tamaño 3×2.

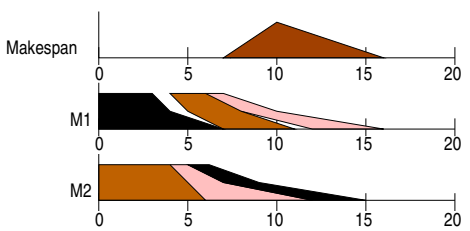

Fig. 2. Gráfico de Gantt de una planificación borrosa.

Para este tipo de soluciones, una de las cuestiones abiertas es cómo evaluar su calidad. En ocasiones se toma como medida el valor esperado del makespan que se obtiene a partir del makespan borroso. Otra posibilidad puede ser aplicar la solución borrosa (en realidad el orden que expresa para las tareas que es nítido) a instancias reales del problema en las que las duraciones de las tareas son compatibles con los correspondientes TFNs, como se hace por ejemplo en [26]. Las soluciones a estos problemas se pueden comparar con aquellas que se obtienen aplicando el orden de las tareas que se obtiene para el problema con las duraciones más probables. A partir de esta comparación se podrán determinar características de la solución borrosa, como por ejemplo la calidad y la robustez, considerando distintos patrones de desviaciones de la duración con respecto a la más probable.

References

1. Brucker, P., Jurisch, B., Sievers, B.: A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics **49** (1994) 107–127
2. Brucker, P.: Scheduling Algorithms. 4th edn. Springer (2004)
3. Carlier, J., Pinson, E.: An algorithm for solving the job-shop problem. Management Science **35(2)** (1989) 164–176
4. Carlier, J., Pinson, E.: Adjustment of heads and tails for the job-shop problem. European Journal of Operational Research **78** (1994) 146–161

5. Dell' Amico, M., Trubian, M.: Applying tabu search to the job-shop scheduling problem. Annals of Operational Research **41** (1993) 231–252
6. Mattfeld, D.C.: Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling. Springer-Verlag (1995)
7. Yamada, T., Nakano, R.: Scheduling by genetic local search with multi-step crossover. In: Proceedings of Fourth International Conference On Parallel Problem Solving from Nature (PPSN IV 1996). (1996) 960–969
8. Zhang, C.Y., Li, P., Rao, Y., Guan, Z.: A very fast ts/sa algorithm for the job shop scheduling problem. Computers and Operations Research **35** (2008) 282–294
9. Kreipl, S.: A large step random walk for minimizing total weighted tardiness in a job shop. Journal of Scheduling **3** (2000) 125–138
10. El-Bouri, A., Shah, P.: A neural network for dispatching rule selection in a job shop. Int. Journal of Advanced Manufacturing Technology **31** (2006) 342–349
11. Giffler, B., Thomson, G.L.: Algorithms for solving production scheduling problems. Operations Research **8** (1960) 487–503
12. Bierwirth, C.: A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum **17** (1995) 87–92
13. Bierwirth, C., Mattfeld, D.C.: Production scheduling and rescheduling with genetic algoritms. Evolutionary Computation **7** (1999) 1–17
14. Varela, R., Vela, C.R., Puente, J., Gómez, A.: A knowledge-based evolutionary strategy for scheduling problems with bottlenecks. European Journal of Operational Research **145** (2003) 57–71
15. Varela, R., Serrano, D., Sierra, M.: New codification schemas for scheduling with genetic algorithms. Lecture Notes in Computer Science **3562** (2005) 11–20
16. Nilsson, N.: Principles of Artificial Intelligence. Tioga, Palo Alto, CA (1980)
17. Sierra, M., Varela, R.: Pruning by dominance in best-first search. In: Proceedings of CAEPIA'2007. Volume 2. (2007) 289–298
18. Pearl, J.: Parallel machine scheduling models with fuzzy processing times. Information Sciences **166** (1984) 49–66
19. Baptiste, Ph. Carlier, J., Jouglet, A.: A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary relesase dates. European Journal of Operational Research **158** (2004) 595–608
20. Brucker, P., Thiele, O.: A branch and bound method for the general-job shop problem with sequence-dependent setup times. Operations Research Spektrum **18** (1996) 145–161
21. Artigues, C., Feillet, D.: A branch and bound method for the job-shop problem with sequence-dependent setup times. Annals of Operations Research **159**(1) (2008) 135–159
22. Balas, E., Simonetti, N., Vazacopoulos, A.: Job shop scheduling with set-up times, deadlines and precedence constraints. In: Proceedings of MISTA'2005. (2005)
23. Vela, C.R., Varela, R., González, M.A.: Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. Journal of Heuristics **Accepted with minor revisions** (2008)
24. Artigues, C., Lopez, P., Ayache, P.: Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. Annals of Operations Research **138** (2005) 21–52
25. Fortemps, P.: Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions of Fuzzy Systems **7** (1997) 557–569
26. González Rodríguez, I., Puente, J., Vela, C.R., Varela, R.: Semantics of schedules for the fuzzy job shop problem. IEEE Transactions on Systems, Man and Cybernetics, Part A **38(3)** (2008) 655–666