*IX Congreso de Ingeniería de Organización*
*Gijón, 8 y 9 de septiembre de 2005*

# Búsqueda Heurística para Problemas de Scheduling*

**María Sierra Sánchez[1], Ramiro Varela Arias[2]**

[1] Dpto. de Informática. Campus de Viesques, 33204 Gijón (Asturias). mariasierra@aic.uniovi.es
[2] Dpto. de Informática. Campus de Viesques, 33204 Gijón (Asturias). ramiro@aic.uniovi.es

**Resumen**

*En este artículo presentamos una aplicación del algoritmo A* al problema Job Shop Scheduling (JSS). Explotamos el espacio de búsqueda de las planificaciones activas y un heurístico basado en relajaciones del problema. Estudiamos experimentalmente el límite hasta el cual se pueden obtener soluciones óptimas de esta forma y también algunas extensiones para obtener soluciones semi-óptimas. Y comparamos los resultados con los obtenidos por otros sistemas similares bien conocidos. Finalmente proponemos algunas mejoras para realizar en un trabajo futuro.*

**Palabras clave:** Scheduling, búsqueda heurística, optimización, satisfacción de restricciones

## 1. Introducción

El algoritmo A* propuesto en Nilsson (1980) y Pearl (1983) es un algoritmo de búsqueda heurística en espacios de estados. En este trabajo presentamos una aplicación del algoritmo A* para resolver el problema *Job Shop Scheduling (JSS)*. El *JSS* es un paradigma de la familia de problemas de Scheduling que ha interesado a muchos investigadores en las últimas tres décadas; en consecuencia, se han aplicado un gran número de técnicas a este problema. En Jain y Meeran (1999) se realiza una extensa revisión de las aproximaciones más importantes.

A la hora de resolver un problema empleando el algoritmo A*, es necesario tomar dos decisiones fundamentales: la primera es definir un espacio de búsqueda completo, y después considerar una buena función heurística. En este trabajo hemos elegido como espacio de búsqueda el espacio de planificaciones activas y hemos considerado como estrategia heurística una, obtenida de relajaciones del problema, basada en el cálculo de la planificación denominada *Jackson´s Preemptive Schedule (JPS),* planificación propuesta en Brucker, Jurisch y Sievers (1994) y Carlier (1994). Con esta aproximación hemos estudiado el límite del tamaño de problemas que se pueden resolver de forma óptima y proponemos un método para obtener, en un tiempo limitado, soluciones no óptimas. Además hemos comparado el algoritmo A* con otros conocidos paradigmas como son el método de *Backtracking* propuesto por Sadeh y Fox (1996) y el método de *Branch and Bound* (*B&B*) propuesto en Brucker, Jurisch y Sievers (1994).

En lo referente a la búsqueda de soluciones óptimas, los resultados experimentales muestran que la versión de A* propuesta es mucho mejor que el algoritmo de *Backtracking* propuesto en Sadeh y Fox (1996), pero peor que el algoritmo de *B&B* propuesto en Brucker, Jurisch y

* Este trabajo ha sido financiado por el proyecto FEDER-MCYT TIC2003-04153 y por la beca FICYT BP04-21.

Sievers (1994). Sin embargo el algoritmo A* se comporta mucho mejor en la famosa instancia FT20 de este problema. En este caso el algoritmo A* alcanza una solución óptima en pocos segundos, mientras que el *B&B* no es capaz de obtener una solución óptima después de varios días. Aquí es importante resaltar que el algoritmo de *B&B* considerado es hoy por hoy la aproximación más eficiente para obtener soluciones óptimas para el problema JSS. Puede resolver casi siempre instancias hasta un tamaño de 15 trabajos y 15 máquinas. Este algoritmo realiza una búsqueda dirigida también por la *JPS* y, a diferencia de la versión de A*, explota un método de propagación de restricciones que permite reducir el espacio de búsqueda sin perder la completud. Este método de propagación de restricciones podría utilizarse también en combinación con el algoritmo A*, con lo que cabría esperar que ambos métodos fuesen realmente comparables, ya que el algoritmo A* puede resolver instancias que resultan difíciles para el algoritmo *B&B*.

Con respecto a la búsqueda de soluciones semi-óptimas, hemos utilizado un método de reducción del espacio de búsqueda que no garantiza la completud, junto con un método de ponderación de la función heurística. Además hemos combinado el algoritmo A* con un algoritmo voraz, guiado también por los valores de la *JPS*. Este algoritmo voraz se aplica a partir de la situación que representa un estado intermedio y proporciona una solución heurística en un tiempo polinomial. Como tiene un coste no despreciable, no se aplica a todos los nodos sino a unos pocos con una probabilidad que depende de la profundidad del nodo.

El resto del trabajo está organizado como sigue. En la sección 2, se formula el problema *JSS*. La sección 3 resume las principales características y propiedades del algoritmo A*. En la sección 4, describimos la aplicación propuesta del algoritmo A* al problema *JSS*, en particular el espacio de búsqueda de planificaciones activas. La sección 5 describe la estrategia heurística empleada que se basa en dos relajaciones del problema. En la sección 6, se muestran los resultados y conclusiones del estudio experimental. Por último, en la sección 7 proponemos algunas las ideas para futuros trabajos.

## 2. El Problema Job Shop Scheduling

El problema Job Shop Scheduling consiste en planificar un conjunto de $N$ trabajos $\{J_1,...,J_N\}$ sobre un conjunto de $M$ recursos o maquinas físicas $\{R_1,...,R_M\}$. Cada trabajo $Ji$ consta de un conjunto de tareas $\{\theta_{i1},...,\theta_{iM}\}$ que han de ser ejecutadas de forma secuencial. Cada tarea $\theta_{ij}$ tiene un tiempo de procesamiento sin interrupción de $du_{ij}$ unidades de tiempo durante el cual requiere el uso exclusivo de un único recurso. Cada trabajo tiene un tiempo de inicio más temprano y en ocasiones se considera también un tiempo de finalización más tardío, lo que obliga a que los tiempos de inicio de las tareas tomen valores en dominios finitos. El objetivo es encontrar una planificación factible, es decir una asignación de tiempos de inicio $st_{ij}$ a cada una de las tareas, que minimice el makespan, es decir el tiempo de finalización de la última tarea. Este problema se conoce como *JSS//Cmax* en la literatura.

## 3. El Algoritmo A*

El algoritmo A* es un método heurístico de búsqueda BF (*Best First, o Primero el Mejor*) en grafos propuesto en Nilsson (1980) y Pearl (1983), aunque cuando el espacio de búsqueda es un árbol, el algoritmo es mucho más simple. El algoritmo parte de un estado inicial *s*, tiene una serie de estados objetivo y un operador de transición *SCS* tal que para cada estado *n* del espacio de búsqueda, *SCS(n)* proporciona el conjunto de estados sucesores de *n*. La transición de un estado *n* a su sucesor *n'* tiene asociado un coste positivo *c(n,n')*. El objetivo de la

búsqueda es encontrar un camino solución, desde el estado inicial *s* a uno de los estados objetivo. El algoritmo mantiene una lista, ABIERTA, con los estados candidatos a ser expandidos. En cada iteración se expande siempre el estado de ABIERTA con menor valor de la función de evaluación *f*, definida como $f(n) = g(n) + h(n)$; donde $g(n)$ es el coste mínimo, conocido hasta el momento, desde el estado inicial *s* hasta el estado actual *n*, (por supuesto si el espacio de búsqueda es un árbol, el valor de $g(n)$ no cambia una vez que el nodo es visitado); y $h(n)$ es la función heurística y proporciona una estimación positiva de la distancia mínima desde el estado actual hasta el estado objetivo más cercano.

El algoritmo A* tiene una serie de propiedades interesantes, la mayoría de ellas dependen de la función heurística *h*. La primera de todas es que el algoritmo es completo. Por otro lado, si la función heurística subestima el coste mínimo, $h^*(n)$, desde *n* a los objetivos, es decir $h(n) \leq h^*(n)$, para todos los estados, el algoritmo es admisible, es decir siempre encuentra una solución óptima. La función heurística $h(n)$ representa el conocimiento disponible sobre el dominio del problema, por lo tanto si se dispone del conocimiento completo con un coste computacional asumible (polinomial en el tamaño del problema), es decir $h(n) = h^*(n)$, se puede encontrar la mejor solución expandiendo el menor número de estados posible. Desafortunadamente esto no suele ocurrir, y en los casos prácticos hemos de buscar la mejor subestimación cuyo coste computacional sea razonable. Esto se debe a otra interesante propiedad que dice que si tenemos dos heurísticos admisibles *h1* y *h2*, tales que $h1(n) < h2(n)$ para todos los estados no terminales *n*, *h2* se dice que está mejor informado que *h1* y se puede demostrar que, en este caso, todo estado expandido por *h2* es también expandido por *h1*. Cuando ocurre esto podemos decir también que el algoritmo que emplea *h2* domina al que emplea *h1*. Finalmente si $h(n) \leq h(n') + c(n,n')$ para todo par de estados *n* y *n'*, el heurístico es monótono. La monotonía tiene también consecuencias relevantes. La primera de ellas es que todo heurístico monótono es admisible. Además si tenemos dos heurísticos monótonos *h1* y *h2*, tales que $h1(n) \leq h2(n)$ para todo estado *n*, *h2* puede ser considerado mejor que *h1* debido a que todo estado *n* expandido por *h2* pero no por *h1* debe cumplirse que $h1(n) = h2(n) = C^* - h^*(n)$, siendo $C^*$ el coste de la solución óptima, y puede considerarse poco probable que estas igualdades se cumplan para un número elevado de estados.

Para instancias de problemas largos la búsqueda de soluciones óptimas es inabordable incluso si disponemos del heurístico mejor informado. En estos casos el algoritmo A* puede ser modificado para obtener soluciones semi-óptimas con un coste razonable. En esta línea podemos encontrar en la literatura varias aproximaciones que se basan, normalmente, en el uso de heurísticos no admisibles, por ejemplo en Nilsson (1980), Nilsson (1998), Russell y Norving (1995).

## 4. Aplicación del algoritmo A* al problema JSS

El espacio completo de planificaciones factibles es muy grande y además contiene muchas soluciones que no son realmente interesantes. Por a ello se han buscado espacios de búsqueda alternativos con un tamaño mucho más pequeño, pero con la condición de que sean completos. Bajo estas condiciones existen tres espacios de búsqueda de interés: espacios de planificaciones *semiactivas*, *activas* y de *no-retraso*. Una planificación es semiactiva si para adelantar el tiempo de inicio de una tarea en una máquina, es necesario modificar el orden de al menos dos tareas. Una planificación es activa si para adelantar el tiempo de inicio de una tarea, al menos otra debe ser retrasada. Finalmente una planificación es de no-retraso si nunca se da la situación de que una máquina esté desocupada en el mismo instante en el que una tarea puede ser procesada en dicha máquina. Las planificaciones de no-retraso son un

subconjunto de las planificaciones activas las cuales son a su vez son un subconjunto de las planificaciones semiactivas. La experiencia muestra que el valor medio del makespan es mucho mayor para las planificaciones semiactivas que para las planificaciones activas; y que también, es mucho mayor para las planificaciones activas que para las planificaciones de no-retraso. Pero al mismo tiempo sólo los espacios de planificaciones semiactivas y activas garantizan contener, al menos, una planificación óptima.

Por lo tanto parece que el espacio más interesante es el de planificaciones *activas*. Este hecho se ve reforzado por la existencia del conocido algoritmo G&T propuesto por Giffler y Thomson (1960). Este algoritmo es un algoritmo voraz que produce planificaciones activas en $N*M$ pasos. En cada paso el algoritmo G&T realiza una elección no determinista sobre un conjunto B de tareas que pueden ser planificadas a continuación. Cada planificación activa puede ser alcanzada tomando la secuencia apropiada de elecciones. Por otra parte si en cada paso consideramos todas las elecciones posibles tenemos un grafo de búsqueda completo, apropiado para estrategias de búsqueda heurística en espacios de estados tales como *Branch and Bound (B&B)*, *Backtracking* o *A**. Esta es la aproximación adoptada por Storer y Talbot (1992), Hatzikonstantis y Besant (1992), por Varela y Soto (2002) y por Sierra (2003).

Basándonos en estos trabajos, como algoritmo de expansión hemos empleado un método basado en el algoritmo G&T en su versión híbrida, propuesto por Bierwirth y Mattfeld (1999). El algoritmo G&T híbrido introduce un mecanismo que permite reducir el número de tareas del conjunto B en cada iteración. Esta reducción viene controlada por el parámetro de reducción $\delta \in [0,1]$; cuando $\delta=1$ la reducción tiene un efecto nulo con lo que se obtendría el mismo espacio que con el algoritmo G&T original. En esta situación, es posible encontrar una secuencia de elecciones de tareas, que de lugar a una solución óptima, ya que se genera el espacio de planificaciones activas completo. En otro caso, si $\delta<1$, el espacio de búsqueda se restringe a un subconjunto de las planificaciones activas; lo que hace imposible garantizar tal secuencia y por lo tanto garantizar que se pueda alcanzar una solución óptima. Tal y como se indica en Bierwirth y Mattfelf (1999), el algoritmo produce planificaciones de no-retraso en el extremo $\delta=0$.

**Algoritmo** *SCS(estado n)*

1. *A = Conjunto con las primeras tareas sin planificar de cada trabajo en el estado n;*
2. *Determinar la tarea* $\theta' \in A$ *que si se planificase en el estado n, tiene el menor tiempo de fin, es decir* $st\theta'+du\theta' \leq st\theta + du\theta, \ \forall\theta \in A;$
3. *Seleccionar la máquina M' que requiere la tarea* $\theta'$*, y construir el conjunto B con las tareas de A que requieren la máquina M';*
4. *Eliminar de B las tareas que no puedan comenzar antes del tiempo de fin de la tarea* $\theta'$*,* $st\theta'+du\theta'$*;*
5. *Reducir el conjunto B a B'*

   $B' = \{ \theta \in B / t\theta < t\theta'' + \delta((t\theta'+du\theta') - t\theta'') \}, \ \delta \in [0,1]\};$
6. ***Mientras*** $(B' \neq \varnothing)$ ***hacer***
   7. *Seleccionar una tarea* $\theta \in B'$ *y planificarla en su menor tiempo de inicio posible para construir la planificación parcial correspondiente al próximo estado n';*
   8. *Añadir n' a los sucesores;*

   ***finmientras;***
9. *retornar sucesores;*

**end.**

**Figura 1.** Algoritmo de expansión de un estado *n*

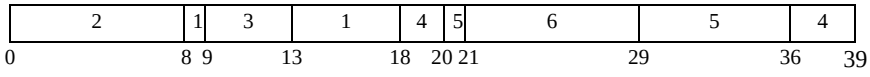
La (Figura 1) muestra el algoritmo de expansión que hemos empleado. Este algoritmo es en realidad una extensión del algoritmo G&T híbrido en el que para cada estado intermedio se consideran todas las posibilidades de expansión, en lugar de realizar una única elección no determinista. Este algoritmo genera el espacio de planificaciones activas, completo ($\delta=1$) o no ($\delta<1$) cuando es aplicado sucesivamente desde el estado inicial. El estado inicial se corresponde con la situación en la que ninguna tarea del problema esta planificada. Mientras que en el resto de estados ya están planificadas un subconjunto de tareas. El coste *c(n,n')* se obtiene de la diferencia $C_{max}(PS(n')) - C_{max}(PS(n))$, donde $C_{max}(PS(n))$ es el tiempo máximo de fin de una tarea en la planificación parcial, *PS(n)*, correspondiente al estado *n*. Es sencillo probar que el espacio de estados generado de este modo es un árbol.

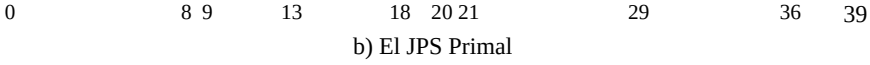
## 5. Estrategias heurísticas para el problema JSS

La técnica más común para obtener heurísticos monótonos es la relajación del problema, Pearl (1983). Esta técnica consiste en relajar algunas de las restricciones con el fin de obtener un problema simplificado que pueda ser resuelto óptimamente en un tiempo polinomial. Luego, el coste de la solución para el problema relajado se toma como estimación del coste del problema real. En el caso del problema JSS, las restricciones que pueden ser relajadas son las secuenciales, de capacidad y de no-expulsión. En este trabajo se ha empleado un heurístico basado en la relajación del problema en dos pasos. Primero se relajan todas las restricciones de capacidad, excepto las restricciones de capacidad correspondientes a una máquina *m*, obteniendo el problema de Secuenciación de una Máquina (*OMS, One Machine Sequencing*) para la máquina *m*. La segunda relajación consiste en relajar la restricción de no-expulsión en el problema *OMS* para esa máquina *m*.

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $r_i$ | 4 | 0 | 9 | 15 | 20 | 21 |
| $p_i$ | 6 | 8 | 4 | 5 | 8 | 8 |
| $q_i$ | 20 | 25 | 30 | 9 | 14 | 16 |

a) Instancia de un problema OMS




b) El JPS Primal

El makespan es 50, se obtiene del tiempo de completud para el trabajo 5(36 + 14)

**Figura 2.** Ejemplo de una instancia OMS y de su correspondiente JPS Primal

El problema de secuenciación de una máquina puede formularse de la siguiente forma: tenemos un conjunto de trabajos $\{J_1,\ldots,J_N\}$; cada trabajo $J_i$ debe ser procesado sin interrupción sobre una máquina *m* durante $p_i$ unidades de tiempo. La máquina *m* solo puede procesar una tarea a la vez. Además cada trabajo $J_i$ tiene un tiempo de procesamiento previo $r_i$ y un tiempo de procesamiento posterior $q_i$ en otras máquinas que no tienen restricciones de capacidad. El objetivo es planificar todos los trabajos de forma que el makespan sea mínimo. El problema *OMS* es un problema NP-completo, por ello se hace una nueva simplificación: se relajan las restricciones de no-expulsión de la máquina *m*. De este modo la solución óptima del problema resultante, tras esta última relajación, se obtiene de la Planificación con Expulsión de Jackson (*JPS, Jackson's Preemptive Schedule"*) propuesta en Brucker, Jurisch y Sievers (1994). El método para calcular la *JPS* comienza asignando a cada trabajo $J_i$ una prioridad proporcional a su $q_i$. Después en cada instante de tiempo *t*, desde el trabajo con menor $r_i$ hasta que todos los trabajos son procesados en la máquina *m*, se planifica el trabajo con mayor prioridad, sobre la

máquina *m*, durante el intervalo *[t,t+1[*. El makespan es el máximo, de entre todos los trabajos, del tiempo de fin sobre *m* más el correspondiente $q_i$. La (Figura 2) muestra un problema de *OMS* para 6 trabajos y su correspondiente *JPS*.

Debido a la simetría entre los tiempos previos y posteriores de procesamiento $r_i$ y $q_i$, el problema dual es también interesante. Este es el problema en el que $r_i$ y $q_i$ son intercambiados. Es fácil ver que la solución del problema primal es también solución del problema dual. Sin embargo el coste del *JPS Dual* (*DJPS*) puede ser diferente del coste del problema *JPS Primal* (*PJPS*). Ambas *JPSs* pueden ser calculadas en $O(N*log_2N)$ pasos.

Del modelo relajado anterior para el problema *JSS,* obtenemos un nuevo heurístico que implica los siguientes cálculos. Para cada estado *n* tomamos el problema *JSS* definido por las tareas no planificadas en *n*, con los tiempos de inicio de las tareas no planificadas del trabajo *J* tomadas como el máximo de $C^J_{\max}(PS(n))$ y el tiempo de fin de la última tarea planificada en el estado *n* en la máquina correspondiente. El heurístico, denominado $h_{JPS}$, se calcula de la siguiente forma:

$$h_{JPS}(n) = MAX_{\,m \in Maquinas} \left\{ MAX\left(PJPS(m), DJPS(m)\right) \right\} - C_{\max}\left(PS(n)\right) \qquad (1)$$

Donde *PJPS(m)* y *DJPS(m)* representan las planificaciones *JPS Primal* y *Dual*, respectivamente, para el problema correspondiente al estado *n* relajado de modo que la única restricción de capacidad que se considera es la correspondiente a la máquina *m*. *Máquinas* se refiere a las máquinas con al menos una tarea sin planificar en el estado *n*.

Aunque el heurístico $h_{JPS}$ es claramente una subestimación del coste óptimo, no es de esperar que la diferencia $h^*(n) - h_{JPS}(n)$ sea uniforme para todos los estados a lo largo del proceso de búsqueda. Sin embargo, se espera que esta diferencia sea grande al principio de la búsqueda, disminuya a medida que la búsqueda avanza y que tienda a ser nula al final de ésta. Esto nos induce a emplear una técnica heurística de ponderación dinámica: se introduce un factor de ponderación *P(n)* que se aplica a la componente *h* de la función *f*, de tal manera que esta función pasa a ser $f(n)=g(n)+P(n)h(n)$. El valor *P(n)* depende de la profundidad del nodo *n*. En este trabajo proponemos el siguiente valor:

$$P(n) = K - \left( K * \frac{profundida\ d(n)}{N * M} \right) + 1 \qquad (2)$$

Donde $K \geq 0$ es el parámetro de ponderación que permite controlar la función de ponderación *P(n)* tal como se describe en Nilsson (1980), *profundidad(n)* es el número de tareas planificadas en el estado *n*, *N* es el número de trabajos y *M* el número de máquinas. Se ve claramente que para valores de $K>0$, no se puede garantizar que la función *f*, mencionada anteriormente, sea admisible ya que la componente *P(n)h(n)* puede sobreestimar el valor de $h^*(n)$. Sin embargo, para algunos valores de *K*, se espera que este componente sea una buena aproximación de $h^*(n)$. Tal y como se prueba en Polh (1977) este método de ponderación garantiza que la solución que se encuentra tiene un coste no superior a $C^*(1+K)$. En Bonet y Geffner (2001), se propone un método estático que emplea un factor de ponderación constante *W* para problemas de planificación de actuaciones; este método garantiza una solución con coste no superior a $C^*(1+W)$. En Hatzikonstantis y Besant (1992) se emplea un método dinámico similar al que empleamos en este trabajo para el problema JSS combinado con un

esquema de memoria limitado y un heurístico no admisible obtenido de una regla de prioridad (*dispatching rule*) clásica que sobreestima el coste óptimo.

En este trabajo para obtener soluciones semi-óptimas hemos empleado valores de $K>0$, combinados con valores para el parámetro de reducción del espacio de búsqueda $\delta<1$.

## 6. Estudio Experimental

En esta sección mostramos los resultados del estudio experimental realizado sobre varios conjuntos de problemas *JSS*. Hemos empleado un prototipo de A* implementado en lenguaje C++ y desarrollado en Builder C++ 6.0 para Windows.

En el primer conjunto de experimentos hemos considerado los problemas propuestos por Sadeh y Fox (1996). Se trata de un conjunto de problemas de tamaño 10×5 que está organizado en 6 grupos de 10 problemas cada uno. Cada grupo está caracterizado por 2 parámetros: *BK* y *RG*. *BK* es el parámetro de *cuello de botella*, este parámetro controla el número de recursos que son cuello de botella. Un recurso es un *cuello de botella* si aparece en la misma posición en la secuencia de tareas de todos los trabajos y además las tareas que requieren el recurso, tienen en promedio, un tiempo de procesamiento más grande que el resto. *RG* es el parámetro de rango que controla la distribución de los tiempos más tempranos de inicio y los tiempos más tardíos de fin de los trabajos.

**Tabla 1.** Resultados experimentales de A* con el heurístico $h_{JPS}$ sobre los problemas de Sadeh (tamaño 10×5)

| Subconjunto de Problemas | Número de problemas resueltos óptimamente | Numero de problemas resueltos expandiendo 51 estados | Número medio de estados expandidos | Tiempo Medio (s.) |
|---|---|---|---|---|
| BK = 1, RG = 0,0 | 10 | 8 | 159.5 | 1.5 |
| BK = 2, RG = 0,0 | 10 | 0 | 32120.2 | 191 |
| BK = 1, RG = 0,1 | 10 | 6 | 280.5 | 0.6 |
| BK = 2, RG = 0,1 | 10 | 2 | 45265.5 | 118.9 |
| BK = 1, RG = 0,2 | 10 | 6 | 115.2 | 0.7 |
| BK = 2, RG = 0,2 | 10 | 4 | 715.7 | 4.6 |

**Tabla 2.** Resultados experimentales sobre los problemas LA01-15 y FT20

| Instancias | Tamaño | A*( $h_{JPS}$.) | | B&B | |
|---|---|---|---|---|---|
| | | Problemas Resueltos | Tiempo Medio (s.) | Problemas Resueltos | Tiempo Medio (s.) |
| La01-05 | 10×5 | 5 of 5 | 46 | 5 of 5 | < 1 |
| La06-10 | 15×5 | 4 of 5 | 13.3 | 5 of 5 | < 1 |
| La11-15 | 20×5 | 4 of 5 | 285.3 | 5 of 5 | < 1 |
| FT20 | 20×5 | 1 of 1 | 30 | 0 of 1 | > 3 days |

La (Tabla 1) resume los resultados obtenidos con el heurístico $h_{JPS}$ para los 60 problemas. Como podemos observar todas las instancias se resuelven óptimamente. De estos resultados se puede concluir que los problemas con $BK=2$ son más difíciles de resolver que los que sólo tienen 1. Sin embargo parece que el parámetro *RG* no tiene relevancia desde el punto de vista de la dificultad del problema. Sadeh y Fox (1996) muestran los resultados de aplicar un algoritmo de *Backtracking* dirigido por un heurístico de ordenación de valores y variables diseñado para hacer frente a los problemas con recursos que son *cuellos de botella*. En este estudio no se trataba de alcanzar soluciones óptimas, sino solamente soluciones no peores en

un 20% a la solución óptima. Incluso dentro de este límite la búsqueda del *Backtracking* sólo en 52 de los 60 problemas alcanzaba una solución factible; aunque el tiempo medio requerido para los casos resueltos es de aproximadamente 2 segundos en un DECstation5000/200.

La (Tabla 2) resume los resultados del algoritmo A* junto con el heurístico $h_{JPS}$ y del algoritmo *B&B* sobre un conjunto de problemas tomados del repositorio OR-library. Como podemos observar A* resuelve todas las instancias de tamaño 10×5, sin embargo para las instancias de tamaños 15×5 y 20×5 sólo se resolvieron 4 de 5 (las 2 no resueltas son el LA10 y LA11). En general A* requiere mucho más tiempo que *B&B*; sin embargo hay una excepción, el problema FT20. En este caso el A* alcanza la solución óptima en 30 segundos, mientras que el B&B no es capaz de encontrar una solución óptima después de 3 días de procesamiento. En la (Tabla 3) resumimos los resultados de 3 conjuntos de 10 problemas cada uno, generados de forma aleatoria por nosotros. Aquí podemos concluir que el prototipo de A* es capaz de resolver, con un coste razonable, problemas cuadrados de hasta un tamaño de 9×9.

**Tabla 3.** Resultados experimentales del A* y *B&B*, sobre 3 conjuntos de 10 problemas cuadrados

| Tamaño de los Problemas | A* ($h_{JPS}$) | | | B&B | | |
|---|---|---|---|---|---|---|
| | Prob. Resueltos | Nro. Medio de Nodos Exp. | Tiempo Medio (s.) | Prob. Resueltos | Nro. Medio de Nodos Exp. | Tiempo Medio (s.) |
| 7×7 | 10 | 7210.8 | 10 | 10 | 37 | < 1 |
| 8×8 | 10 | 21647.5 | 40.6 | 10 | 23 | < 1 |
| 9×9 | 8 | 155383.75 | 643.6 | 10 | 11 | < 1 |

Una vez alcanzado el límite de tamaño de problemas resolubles óptimamente, hemos intentado resolver de forma semi-óptima, tanto los problemas no resueltos óptimamente en los experimentos anteriores, como una batería de problemas de tamaño 10×10 tomados del repositorio OR-library (problemas ORB). Para ello hemos empleado la técnica de reducción del espacio de búsqueda combinada con la técnica de ponderación de la estimación heurística. Nuestro objetivo es encontrar soluciones buenas, óptimas o no, en un tiempo razonable (1200 segundos); para ello se emplea una técnica de construcción de soluciones heurísticas que proporciona cotas superiores de la solución óptima, y en consecuencia permite podar el espacio de búsqueda del algoritmo A* evitando expandir nodos que superen el coste de la mejor solución calculada hasta el momento.

La técnica de construcción de soluciones heurísticas consiste en emplear el algoritmo *G&T híbrido*, aplicando a la lista de sucesores de cada nodo *n* tres reducciones, la primera tomando un factor de reducción $\delta$=0,5; la segunda dejando en el conjunto de sucesores sólo aquellos nodos *n'* tales que tienen el mejor valor de $c(n,n') + JPS(n',M')$; donde $JPS(n',M')$ es el valor del *JPS* para las tareas que requieren la máquina *M'* y no están planificadas en el estado *n'*; y *M'* es la máquina requerida por la tarea planificada para pasar de *n* a *n'*. Y por último se eliminan aquellos sucesores con una estimación de coste superior a la cota superior actual. Cuando el conjunto de sucesores resultante tiene una tamaño mayor que 1 se elije de forma aleatoria el siguiente sucesor; en caso de que tenga tamaño 0 se para el proceso de construcción de la solución heurística, manteniéndose la mejor encontrada hasta el momento. El primer paso de esta técnica consiste en obtener una solución inicial (se calcula para el estado inicial), por lo que no se aplica la última reducción. Por tratarse de una técnica costosa sólo se aplicará con una probabilidad 0,3 a los nodos que superen una profundidad del 80% de la profundidad máxima del espacio de búsqueda ($N$*$M$+1), ya que cuanto más cerca nos

encontramos de obtener una solución, es de esperar que las soluciones calculadas con esta técnica sean mejores. El tiempo máximo de ejecución se fija en 1200 s y $\delta$=0.5 en el algoritmo de cálculo de soluciones heurísticas.

**Tabla 4.** Resultados experimentales sobre los problemas ORB01-10. δ=0 en A*( $h_{JPS}$)

| Problema | Solución Óptima | K=0,1 | | K=0,2 | | K=0,3 | |
|---|---|---|---|---|---|---|---|
| | | Solución | Tiempo (s.) | Solución | Tiempo (s.) | Solución | Tiempo (s.) |
| ORB01 | 1059 | 1085 | 224 | 1099 | 15 | 1186 | 12 |
| ORB02 | 888 | 905 | 565 | 909 | 99 | 951 | 9 |
| ORB03 | 1005 | 1269 | 1200 | 1074 | 533 | 1074 | 21 |
| ORB04 | 1005 | 1180 | 1200 | 1061 | 317 | 1113 | 124 |
| ORB05 | 887 | 911 | 298 | 928 | 31 | 954 | 4 |
| ORB06 | 1010 | 1046 | 1106 | 1089 | 42 | 1152 | 5 |
| ORB07 | 397 | 406 | 178 | 419 | 3 | 439 | 11 |
| ORB08 | 899 | 965 | 6 | 1014 | 37 | 1120 | 0 |
| ORB09 | 934 | 974 | 770 | 1001 | 48 | 1086 | 0 |
| ORB10 | 944 | 1012 | 326 | 1067 | 6 | 1131 | 1 |
| LA10 | 958 | 969 | 1 | 1004 | 0 | 989 | 1 |
| LA11 | 1222 | 1248 | 0 | 1222 | 1 | 1311 | 1 |
| p09x09_08 | 774 | 808 | 33 | 804 | 1 | 850 | 1 |
| p09x09_09 | 840 | 854 | 18 | 854 | 8 | 862 | 0 |

**Tabla 5.** Resultados experimentales sobre los problemas ORB01-10, δ=0,5 en A*( $h_{JPS}$)

| Problema | Solución Óptima | K=0,1 | | K=0,2 | | K=0,3 | |
|---|---|---|---|---|---|---|---|
| | | Solución | Tiempo (s.) | Solución | Tiempo (s.) | Solución | Tiempo (s.) |
| ORB01 | 1059 | 1078 | 319 | 1110 | 33 | 1173 | 11 |
| ORB02 | 888 | 996 | 1200 | 902 | 122 | 985 | 1 |
| ORB03 | 1005 | 1201 | 1200 | 1044 | 64 | 1079 | 11 |
| ORB04 | 1005 | 1135 | 1200 | 1068 | 667 | 1073 | 161 |
| ORB05 | 887 | 889 | 1200 | 924 | 6 | 949 | 7 |
| ORB06 | 1010 | 1034 | 677 | 1080 | 24 | 1092 | 11 |
| ORB07 | 397 | 401 | 578 | 412 | 4 | 417 | 8 |
| ORB08 | 899 | 935 | 4 | 1040 | 0 | 1000 | 11 |
| ORB09 | 934 | 943 | 389 | 990 | 46 | 1039 | 6 |
| ORB10 | 944 | 985 | 266 | 1039 | 11 | 1078 | 8 |
| LA10 | 958 | 958 | 0 | 999 | 0 | 958 | 0 |
| LA11 | 1222 | 1234 | 1 | 1272 | 1 | 1222 | 1 |
| p09x09_08 | 774 | 774 | 147 | 806 | 10 | 838 | 0 |
| p09x09_09 | 840 | 853 | 48 | 937 | 1 | 1031 | 0 |

**Tabla 6.** Resultados experimentales sobre los problemas ORB01-10, δ=1 en A*( $h_{JPS}$)

| Problema | Solución Óptima | K=0,1 | | K=0,2 | | K=0,3 | |
|---|---|---|---|---|---|---|---|
| | | Solución | Tiempo (s.) | Solución | Tiempo (s.) | Solución | Tiempo (s.) |
| ORB01 | 1059 | 1059 | 239 | 1089 | 50 | 1143 | 3 |
| ORB02 | 888 | 1058 | 1200 | 909 | 400 | 931 | 62 |
| ORB03 | 1005 | 1256 | 1200 | 1052 | 173 | 1096 | 28 |
| ORB04 | 1005 | 1169 | 1200 | 1051 | 1200 | 1098 | 2 |
| ORB05 | 887 | 1035 | 1200 | 924 | 32 | 934 | 4 |
| ORB06 | 1010 | 1263 | 1200 | 1042 | 61 | 1076 | 11 |
| ORB07 | 397 | 409 | 1200 | 412 | 16 | 422 | 8 |
| ORB08 | 899 | 929 | 344 | 967 | 39 | 983 | 10 |
| ORB09 | 934 | 953 | 1086 | 1000 | 68 | 1098 | 1 |
| ORB10 | 944 | 986 | 1041 | 1019 | 19 | 1141 | 0 |
| LA10 | 958 | 963 | 1 | 1019 | 0 | 1014 | 0 |
| LA11 | 1222 | 1222 | 1 | 1280 | 0 | 1304 | 0 |
| p09x09_08 | 774 | 776 | 311 | 798 | 58 | 833 | 0 |
| p09x09_09 | 840 | 851 | 11 | 916 | 0 | 949 | 0 |

En estas condiciones hemos dado al parámetro de reducción $\delta$ los valores 0, 0,5 y 1, variando para cada uno de ellos el factor de ponderación de 0,1 a 0,3 con un incremento de 0,1. Las tablas (Tabla 4), (Tabla 5) y (Tabla 6) muestran los resultados. Lo primero que se observa para todos los valores de $\delta$ es que a medida que se incrementa $K$ las soluciones son en general peores pero el coste de la búsqueda disminuye, pudiendo considerar que la mejor relación entre el coste de la búsqueda y la calidad de la solución se obtiene para $K$=0,2. En lo que respecta al parámetro de reducción, se observa que el valor de $\delta$ con mejor comportamiento es $\delta$=0,5, valor para el que se resuelven de forma óptima los problemas LA10, LA11, p09x09_08. Este resultado es análogo al obtenido en Bierwirth y Mattfeld (1999) dónde se utiliza un algoritmo genético que realiza búsqueda en el espacio de planificaciones activas y se utiliza también la reducción del mismo a través del parámetro $\delta$.

## 7. Trabajo Futuro

Como principal idea proponemos incorporar en el prototipo de A* los métodos de propagación de restricciones que se utilizan, por ejemplo, en Brucker, Jurisch y Sievers (1994). De este modo se espera conseguir un algoritmo comparable a los métodos más eficientes de ramificación y poda.

## Referencias


Bierwirth, Ch.; Mattfeld, D. C. (1999). *Production Scheduling and Rescheduling with Genetic Algoritms*. Evolutionary Computation 7 (1), 1-17.

Bonet, B.; Geffner, H. (2001). *Planning as Heuristic Search*. Art. Intelligence 129, 5-33.

Brucker, P.; Jurisch, B.; Sievers, B. (1994). *A branch and bound algorithm for the job-shop scheduling problem*. Discrete Applied Mathematics 49, 107-127.

Carlier, J.; Pinson, E. (1994). *Adjustment of heads and tails for the job-shop problem*. European Journal of Operational Research 78, 146-161.

Giffler, B.; Thomson, G. L. (1960). *Algorithms for Solving Production Scheduling Problems*. Operations Reseach 8, 487-503.

Hatzikonstantis, L.; Besant, C. B. (1992). *Job-Shop Scheduling Using Certain Heuristic Search Algorithms*. Int. J. Adv. Manuf. Tecnol. 7, 251-261.

Jain, A. S.; Meeran, S. (1999). *Deterministic job-shop scheduling: Past, present and future*. European Journal of Operational Research 113, 390-434.

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Nilsson, N. (1998). *Artificial Intelligence. A New Synstesis*. Morgan Kaufmann Publishers.

Pearl, J. (1983). *Heuristics*. Morgan Kauffman, San Francisco, CA.

Pohl, I. (1977) *Practical and theoretical considerations in heuristic search algorithms*. Machine Intelligence 8. Ed. E. W. Elcock and D. Michie, Ellis H. Ltd., Chich., G.B.

Russell, S.; Norving, P. (1995). *Artificial Intelligence, A Modern Approach*. Prentice-Hall International.

Sadeh, N.; Fox, M.S. (1996). *Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem*. Artificial Intelligence 86, 1-41.

Sierra, M. (2003). *Resolución de Problemas de Scheduling con el Algoritmo A**. Proyecto Fin de Carrera, EPSIG, Universidad de Oviedo.

Storer, R.; Talbot, F. (1992). *New search spaces for sequencing problems with application to job shop scheduling*. Management Science 38, 1494-1509.

Varela, R.; Soto, E. (2002). *Scheduling as Heuristic Search with State Space Reduction*. Ed. Springer, LNAI 2527, pp. 815-824.