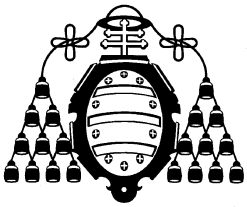
UNIVERSIDAD DE OVIEDO



DEPARTAMENTO DE INFORMÁTICA

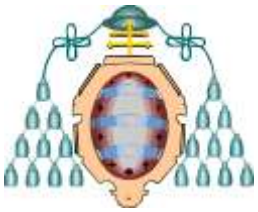
Mejora de Algoritmos de Búsqueda Heurística mediante Poda por Dominancia. Aplicación a Problemas de Scheduling

Tesis Doctoral

Autor: María Rita Sierra Sánchez

Director: José Ramiro Varela Arias

Noviembre, 2009



Edita: Universidad de Oviedo,
Biblioteca Universitaria, 2010
Colección Tesis Doctoral-TDR nº 72
ISBN 978-84-693-4917-5
D.L.: AS.00968-2010

Agradecimientos

A mi director de Tesis, José Ramiro Varela Arias, por su entusiasmo, dedicación y ánimo. A los miembros del Grupo de Tecnologías de la Computación, por su ayuda y apoyo, por todo lo que he aprendido, y sobre todo por su amistad.

Gracias a mi familia y amigos, especialmente a Dani, a mis padres Ismael y Maita, y a mis hermanos Ismael y Verónica. Su cariño y apoyo hacen que no abandone mis sueños. *Os quiero.*

Resumen

Los problemas de scheduling aparecen con profusión en la vida real en numerosos entornos productivos y de servicios. Se trata de problemas que requieren organizar en el tiempo la ejecución de tareas que compiten por el uso de un conjunto finito de recursos y que están sujetas a un conjunto de restricciones impuestas por factores como las características físicas del entorno, relaciones temporales o la normativa laboral. Además se trata de optimizar uno o varios criterios que se representan mediante funciones objetivo y que están relacionados normalmente con el coste, el beneficio o el tiempo de ejecución. Algunos ejemplos de problemas de esta naturaleza son los siguientes:

- Fabricación de obleas para circuitos semiconductores, donde cada oblea precisa de una serie de tareas como limpieza, oxidación, metalización, etc. El objetivo puede maximizar la utilización de algunas máquinas que son cuello de botella o minimizar el tiempo de ejecución.

- Planificar el aterrizaje de un conjunto de aviones sujetos a restricciones temporales que dependen de las características de los aviones. Los objetivos pueden ser minimizar la penalización por desvío con respecto al tiempo preferente de los aviones o maximizar las condiciones de seguridad.

- Planificar las rutas de flotas de autobuses, donde se trata de optimizar la ocupación de los vehículos y de ajustar los turnos de los conductores de acuerdo con la normativa laboral.

- Enrutamiento de paquetes de datos a través líneas de comunicación, donde se trata de maximizar el uso de la red y de minimizar los tiempos de llegada de los mensajes.

Dado que estos problemas son de naturaleza combinatoria, es decir que hay que elegir una entre un conjunto exponencialmente grande de combinaciones posibles, los problemas de scheduling precisan de algoritmos de búsqueda inteligentes para encontrar soluciones aceptables en un tiempo razonable. Así, en la literatura se pueden encontrar aproximaciones a los problemas de scheduling basadas en prácticamente todas las metaheurísticas conocidas y en particular en los algoritmos de búsqueda heurística propios de áreas como la Investigación Operativa y la Inteligencia Artificial.

En esta tesis nos centramos en el problema Job Shop Scheduling y en la técnica de búsqueda heurística en espacios de estados. Nuestro objetivo es diseñar estrategias que resulten eficaces y eficientes para diferentes funciones objetivo, tanto para encontrar soluciones exactas, cuando el tamaño del problema lo permita, como para obtener soluciones aproximadas para instancias mayores. La función objetivo a la que los investigadores han prestado mayor atención es sin duda el makespan, o tiempo de finalización de la última tarea. Las propiedades de esta versión del problema son muy bien conocidas y han permitido desarrollar métodos exactos y aproximados muy eficientes que se basan en el concepto de camino crítico. El inconveniente de estos métodos es que no se generalizan de forma eficiente para otras funciones objetivo como el tiempo de flujo total o el tardiness.

La aportación principal de esta tesis es la formalización de un método de poda basado en relaciones de dominancia entre los estados del espacio de búsqueda que se puede aplicar en principio a todas las funciones objetivo convencionales. Aunque el método no resulta competitivo con los métodos basados en el camino crítico cuando se trata de minimizar el makespan, sí lo es con los métodos que no están basados en el camino crítico y que son generalizables a otras funciones objetivo. Para funciones objetivo como el tiempo de flujo total, los resultados experimentales que hemos realizado sobre bancos de ejemplos estándar demuestran que el método es competitivo con otros métodos del estado del arte tanto para obtener soluciones óptimas como sub-óptimas.

Índice de Contenidos

Lista de Figuras

Índice de Tablas

Índice de Algoritmos

Capítulo 1

INTRODUCCIÓN

1.1. Introducción

El problema Job Shop Scheduling (JSS) es un paradigma de la familia de problemas de optimización combinatoria y de satisfacción de restricciones. Esta familia de problemas ha interesado a los investigadores durante las últimas décadas debido a su complejidad y a la variedad de situaciones reales que modelan. Los problemas de scheduling implican, en general, asignar una serie de recursos a una serie de tareas, satisfaciendo unas restricciones y uno o varios criterios de optimización [10, 66]. El problema JSS es un problema clásico, que debido a su presencia en el mundo real, principalmente en los sectores industrial y de servicios, ha despertado mucho interés entre los investigadores.

Los recursos y las tareas toman en las organizaciones diferentes formas. Así pues, por ejemplo, los recursos pueden ser máquinas en un taller, pistas de aterrizaje en un aeropuerto, el equipo de trabajo en una obra en construcción, las unidades de procesamiento en un entorno de computación, y así sucesivamente. Las tareas pueden ser unidades de trabajo en un proceso de producción, los despegues y aterrizajes en un aeropuerto, las fases de un proyecto de construcción, ejecución de programas de ordenador, etc. Cada tarea puede tener asignada una prioridad, y un tiempo de inicio más temprano y un tiempo de fin límite. Los objetivos también pueden tomar diversas formas, por ejemplo, la minimización del tiempo de completud de la última tarea (a este criterio de optimización se le conoce con el nombre de makespan), o la minimización del número de tareas terminadas tras su tiempo de fin límite (criterio conocido como minimización del tardiness).

Algunos ejemplos de problemas de esta naturaleza son los siguientes [66, 39].

Fábrica de Bolsas de Papel. Consideremos una fábrica dedicada a producir bolsas de papel. El material básico para cada una de las operaciones son rollos de papel. El proceso productivo consiste en tres fases: impresión del logo, pegado de los laterales de la bolsa, y la costura del final o de ambos finales de la bolsa. Cada fase consiste en una serie de máquinas no necesariamente idénticas. Las máquinas de una fase se pueden diferenciar en la velocidad a la que operan, el número de colores que pueden imprimir o el tamaño de la bolsa que producen. Cada orden de producción indica la

cantidad de bolsas de cada tipo a fabricar y a entregar en una fecha determinada. Los tiempos de procesamiento de las diferentes operaciones son proporcionales al tamaño de la orden, es decir al número de bolsas de la orden.

Un retraso en la entrega implica una sanción en forma de pérdida de confianza. La magnitud de la penalización depende de la importancia de la orden o del cliente y del retraso en la entrega. Uno de los objetivos del sistema de planificación es minimizar la suma de estas sanciones.

Cuando una máquina cambia de un tipo de bolsa a otro se requiere un tiempo de puesta en marcha o de configuración (*setup*). La longitud del tiempo de *setup* depende de las similitudes entre las dos órdenes consecutivas (el número de colores en común, las diferencias en tamaños, etc...). Un objetivo importante del sistema de planificación es la minimización del tiempo total de *setups*.

Fabricación de Semiconductores. Los semiconductores se fabrican en instalaciones altamente especializadas. Este es el caso de los chips de memoria y los microprocesadores. El proceso de producción en estas instalaciones generalmente consta de cuatro fases: fabricación de la oblea, sonda de la oblea, el montaje o embalaje, y la prueba final. La fabricación de las obleas es la fase más compleja. Para producir los circuitos, las capas de metal y el material de la oblea se construyen con patrones sobre las obleas de silicio o arseniuro de galio. Cada capa requiere una serie de operaciones que normalmente incluyen: limpieza, oxidación, deposición y metalización, la litografía, el grabado, la implantación de iones, extracción fotorresistente, y la inspección y medición. Como consiste en varias capas, cada oblea debe someterse a estas operaciones varias veces, por tanto hay una cantidad de recirculación en el proceso muy elevada. Las obleas se mueven a través de la instalación en lotes de un determinado tamaño (por ejemplo 24). Algunos equipos pueden requerir un tiempo de puesta en marcha o de configuración (*setup*) entre lotes. Este tiempo depende de la configuración del lote que acaba de terminar y el lote a punto de comenzar.

El número de pedidos en el proceso de producción es a menudo de cientos, teniendo cada uno su propia fecha de inicio y de envío o entrega. Uno de los objetivos de la planificación es cumplir las fechas de envío, en la medida en que sea posible, maximizando el rendimiento. El principal objetivo es maximizar la utilización del equipo, especialmente de las máquinas que son cuellos de botella, logrando también minimizar los tiempos de espera y de *setup*.

Asignación de Puertas de Embarque en un Aeropuerto. Las terminales de los aeropuertos importantes tienen multitud de puertas de embarque, y cientos de aviones que despegan y aterrizan cada día. Las puertas de embarque no son todas iguales, y tampoco lo son los aviones. Algunas puertas se encuentran en lugares espaciosos en los que los aviones grandes pueden acomodarse con facilidad. Sin embargo, otras puertas se encuentran en lugares de difícil acceso para los aviones, teniendo incluso que remolcar a muchos de ellos a sus puertas. Los aviones despegan y aterrizan con un calendario determinado. Sin embargo, dicha programación está sujeta a ciertas variaciones aleatorias debidas al clima, a acontecimientos imprevistos, etc. Durante el tiempo que un avión permanece en una puerta de embarque, los pasajeros que llegan se han de bajar del avión, el avión ha de ser preparado y los pasajeros que salen han de embarcar. La hora prevista de salida puede verse como una fecha de vencimiento, siendo este parámetro lo que mide el rendimiento de la línea

aérea. Sin embargo, si se sabe de antemano que el avión no puede aterrizar en su próximo aeropuerto de destino, debido a la congestión prevista en dicho aeropuerto a la hora de su aterrizaje, el avión no despega por política de ahorro de combustible. Si un avión no puede despegar, la política de funcionamiento es que los pasajeros permanezcan en la terminal y no en el avión. El aplazamiento de un embarque hace que al avión permanezca en la puerta un tiempo prolongado, por lo que otros aviones no pueden utilizarla. La planificación ha de asignar los aviones a las puertas de embarque de modo que dicha asignación sea físicamente posible (las puertas estén disponibles en los tiempos de aterrizaje previstos) y se optimicen una serie de objetivos.

El objetivo de la planificación es minimizar tanto el trabajo del personal de la aerolínea como los retrasos del avión. En este entorno las puertas son los recursos y el manejo y prestación de servicios de los aviones las tareas. La llegada de un avión a una puerta representa el tiempo de inicio de una tarea, y la salida su tiempo de fin.

Planificación de Tareas en una CPU. Una de las funciones de un sistema operativo multitarea en un ordenador es programar el tiempo que la CPU dedica a los diferentes programas que han de ser ejecutados. Generalmente, no se conoce de antemano el tiempo de ejecución exacto de cada uno de ellos. Sin embargo, una distribución aproximada de dichos tiempos de ejecución sí puede ser conocida de antemano, teniendo en cuenta sus medias y sus variaciones. Además, cada tarea, por lo general tiene un nivel de prioridad determinado (el sistema operativo permite a los usuarios establecer la prioridad de cada tarea). En este caso, el objetivo es minimizar la suma de los tiempos ponderados de fin de todas las tareas. Para evitar que una tarea breve permanezca mucho tiempo esperando que terminen las tareas más largas, que tienen mayor prioridad, el sistema operativo divide cada tarea en pequeños trozos. Una vez hecho esto, el sistema operativo rota los trozos en la CPU, de modo que en cualquier intervalo de tiempo la CPU pase cierta cantidad de tiempo con cada tarea. De esta manera, si el tiempo de procesamiento de una de las tareas es muy corto, la tarea será capaz de dejar el sistema rápidamente. La interrupción de una tarea en ejecución se conoce como expulsión. Es evidente que una política óptima en este entorno hace uso intensivo de expulsiones.

No siempre está claro el impacto que la planificación tiene sobre los objetivos deseados. ¿Tiene sentido invertir tiempo y esfuerzo en la búsqueda de una buena planificación, y no sólo elegir una una al azar?. En la práctica, con frecuencia sucede que la elección de una planificación tiene un impacto significativo en el rendimiento del sistema, por lo que tiene sentido dedicar tiempo y esfuerzo a buscar una planificación adecuada. La planificación puede ser difícil desde un punto de vista técnico y de aplicación. Las dificultades encontradas en el aspecto técnico son similares a las dificultades encontradas en otras formas de optimización combinatoria y modelización estocástica. Las dificultades en el lado de aplicación son de un tipo completamente diferente. Dependen de la exactitud del modelo utilizado para el análisis del problema real de planificación y de la fiabilidad de los datos de entrada que se necesitan.

Planificación de Horarios Ferroviarios. El problema de Optimizar y Programar horarios para Trenes (OPT) es un problema real que puede ser considerado como un problema de scheduling tipo

Job Shop. El sistema de tráfico ferroviario es muy complejo, por lo que el proceso de planificación es dividido en varios pasos, siendo la planificación de horarios para trenes uno de estos pasos. La formalización y resolución de este problema como un problema JSS puede verse con mayor nivel de detalle en [39].

En el problema OPT se considera un conjunto de trenes y una línea ferroviaria, la cual estará formada por una secuencia ordenada de dependencias (apeaderos, estaciones, bifurcaciones, ...etc). Cada tren tiene definido un recorrido sobre la misma, no el mismo para todos. El orden entre las dependencias determina el orden en el cual deben ser visitadas por los trenes que viajan en un determinado sentido, que se denomina *ida*. Cuando los trenes viajan en sentido contrario se denomina *vuelta*.

El problema consiste en asignar un horario factible a cada tren, cumpliendo una serie de restricciones y optimizando una determinada función objetivo. Las restricciones que ha de cumplir la planificación solución tienen en cuenta la capacidad de la infraestructura ferroviaria (señalización de cada tramo y número de vias en la dependencia y entre dependencias consecutivas), el tráfico de la línea (gestión de varios trenes por línea y evitar colisiones), su mantenimiento (horario de cierre de estaciones, tiempos de mantenimiento), el servicio al cliente (frecuencia de salidas entre trenes, paradas en apeaderos) y las condiciones impuestas a la planificación (intervalos de salida y llegada de los trenes, y máximo retraso permitido sobre un horario para el tren).

Vemos ahora la correspondencia entre la definición del problema y su formulación como un problema JSS. Un *trabajo* es el recorrido completo que lleva a cabo un tren. Las *tareas* del trabajo son el paso de un tren por las estaciones o por los tramos entre estaciones. Para realizar cada tarea es necesario una serie de *recursos*, en este problema los recursos son las vías en la estación o las vías en el tramo, dependiendo del tipo de tarea. El *orden* de las tareas en un trabajo viene determinado por el recorrido de cada tren. La *duración* de una tarea viene determinada por el tiempo que permanece un tren en una estación o por el tiempo que un tren emplea en recorrer un tramo de su recorrido.

Los trenes de *ida* emplean los recursos en un determinado orden, los de *vuelta* emplean parte o los mismos recursos en el orden inverso.

Las restricciones del problema determinan las condiciones en que es factible la utilización de los recursos, teniendo en cuenta todas las tareas del problema. Por ejemplo, la restricciones de cruce regulan el uso de recursos utilizados por tareas pertenecientes a trabajos que siguen un orden inverso; así pues estas tareas no podrá utilizar a la vez el mismo recurso debiendo existir un margen de seguridad (tiempo de *setup*) entre operaciones que emplean el mismo recurso, así como entre el intercambio de recursos entre operaciones. Otras restricciones regulan la utilización de recursos por operaciones que siguen el mismo orden, algunos permiten ser utilizados por más de una tarea siempre que exista un intervalo de tiempo entre sus tiempos de inicio y fin; otros no permiten que dos tareas lo utilicen simultáneamente. Las estaciones son centros de trabajo que no pueden llevar a cabo más tareas que el número de máquinas (vías) disponibles. El periodo de tiempo que una tarea puede utilizar un recurso, o que un recurso puede ser utilizado teniendo en cuenta su capacidad total, está también limitado por las operaciones de mantenimiento o el horario de cierre de una estación.

El objetivo de este problema es minimizar el retraso promedio de cada trabajo con respecto a

un tiempo de referencia (horario) y la diferencia entre el retraso promedio correspondiente a los trabajos que siguen un orden de utilización de recursos y el retraso correspondiente a los trabajos que siguen el orden inverso. A cada uno de estos criterios se le asigna un peso.

1.2. Marco de Trabajo

Como hemos indicado anteriormente, el problema JSS es un problema clásico que consiste en la asignación de unos recursos a unas tareas, cumpliendo una serie de restricciones y optimizando una serie de objetivos. Tradicionalmente, el criterio de optimización es el makespan; para esta versión del problema ($J||C_{max}$) se han desarrollado numerosos métodos exactos y aproximados, la mayoría 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 [12, 14, 11], desarrollado a partir de conceptos y técnicas propuestas también por otros investigadores como Carlier y Pinson [16, 17]. Entre los métodos no exactos destacan las técnicas de búsqueda local que utilizan técnicas de vecindad definidas en principio por Dell' Amico y Trubian [24] y desarrolladas por otros investigadores. Normalmente estas técnicas se combinan con otras metaheurísticas como los algoritmos genéticos [51, 94] o la búsqueda tabu [95]. Las técnicas basadas en el camino crítico normalmente no son muy eficientes para criterios de optimización diferentes del makespan. Por ejemplo, en [47] se define un esquema de vecindad para el problema JSS con minimización del tiempo de flujo total ($J||\sum C_i$) que requiere calcular múltiples caminos críticos, dando lugar a un gran número de vecinos para cada solución.

Los métodos basados en el camino crítico son muy eficientes para resolver el problema $J||C_{max}$, sin embargo no son fácilmente generalizables a otras funciones objetivo. Por ejemplo, el algoritmo de Brucker es capaz de resolver de forma exacta problemas de tamaño 10×10 (10 trabajos y 10 máquinas), pero tiene el inconveniente de que no se puede generalizar, de forma simple, para resolver el problema JSS con funciones objetivo distintas a la minimización del makespan, ya que el espacio de búsqueda que recorre solamente es dominante para esta función objetivo. Una alternativa al esquema de ramificación del algoritmo de Brucker es el que se deriva del algoritmo voraz $G\&T$ propuesto por Giffler y Thomson en [31]. 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^* propuesto por Nilsson en ([56]). Este algoritmo es claramente menos eficiente 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, es competitivo con cualquier otro método exacto, o incluso con algunos métodos aproximados como por ejemplo los algoritmos genéticos y la búsqueda local.

En esta tesis resolvemos en primer lugar la versión del problema $J||C_{max}$ y a continuación introducimos una de las variantes con gran interés actual como es el problema $J||\sum C_i$. Ambas versiones se resolverán con el algoritmo A^*, recorriendo el espacio de planificaciones activas.

1.3. Motivaciones

La resolución del problema JSS mediante el algoritmo A^* supone un reto si tenemos en cuenta la cantidad de recursos computacionales que este algoritmo consume. El recurso crítico que maneja este algoritmo es principalmente la memoria. Cuando el espacio de búsqueda a recorrer es demasiado grande, la memoria se agota rápidamente, por lo que el algoritmo en muchas ocasiones agota los recursos disponibles sin alcanzar una solución.

En esta tesis, consideraremos el espacio de búsqueda de las planificaciones activas. Este espacio de búsqueda es el más pequeño que garantiza contener al menos una solución óptima. A pesar de ello, continúa siendo un espacio demasiado grande para muchos problemas. El algoritmo A^* puede emplear información heurística para guiar la búsqueda, de modo que si el heurístico empleado es bueno se reduce mucho el número de estados a visitar para alcanzar una solución. Sin embargo, aún disponiendo de buenos heurísticos, en muchas ocasiones el número de nodos a visitar continúa siendo demasiado elevado. La solución a este problema pasa, o bien por relajar las restricciones de optimalidad, en cuyo caso puede que la solución alcanzada no sea la óptima, o bien por aplicar alguna técnica de poda que permita reducir el espacio de búsqueda sin dejar fuera a todas las soluciones óptimas.

Teniendo en cuenta los aspectos anteriores, la motivación principal de esta tesis es lograr que el algoritmo A^* encuentre una solución óptima para los problemas con un tamaño cercano al umbral que comienza a ser considerado interesante (10×10), en un tiempo razonable. Se trata de evitar que el algoritmo A^* agote la memoria sin alcanzar solución, diseñando heurísticos bien informados para cada versión del problema y una técnica de poda que se adapte a diferentes versiones del problema JSS y que garantice alcanzar la solución óptima.

1.4. Objetivos

Los principales objetivos de la tesis son los siguientes:

1 Resolver óptimamente problemas $J||C_{max}$ de tamaño cercano al umbral de 10×10 mediante el algoritmo A^*. Para ello se estudiarán diversas estrategias heurísticas admisibles.

2 Formalizar y diseñar una técnica de poda que permita reducir el espacio de búsqueda garantizando que se alcance una solución óptima. El objetivo es desarrollar un método de poda que sea aplicable a las distintas versiones del problema JSS, y cuya idea sea aplicable a otros problemas de la familia de Scheduling.

3 Formalizar y adaptar la técnica de poda propuesta a la versión del problema $J||C_{max}$.

4 Estudiar la versión del problema $J||\sum C_i$ y su resolución de forma óptima mediante el algoritmo A^*. Se diseñarán una serie de heurísticos admisibles que persiguen guiar la búsqueda, reduciendo el número de estados a visitar para alcanzar una solución.

5 Estudiar y adaptar la técnica de poda propuesta a la versión del problema $J||\sum C_i$.

6 Estudiar el comportamiento de los algoritmos relajando las condiciones de optimalidad. Cuando empleando buenos heurísticos y la técnica de poda se agoten los recursos disponibles sin alcanzar una solución, se relajará la restricción de optimalidad. Se estudiarán diversas técnicas que permitan alcanzar una solución, aunque no sea la óptima, y se aplicarán a ambas versiones del problema $J||C_{max}$ y $J||\sum C_i$.

7 Evaluar las técnicas propuestas. Se emplearán problemas seleccionados del repositorio *OR-library*, originalmente descrito por J.E.Beasley en [7] y que se puede encontrar en [6]. Este repositorio contiene instancias de problemas JSS de diferente tamaño y dificultad, utilizados ampliamente en las investigaciones con problemas de Scheduling.

1.5. Esquema

A la hora de abordar los objetivos planteados en esta tesis hemos comenzado estudiando cómo se formula y representa el problema JSS, así como los diferentes espacios de soluciones factibles que se tienen para él. Además, hemos estudiado una serie de técnicas publicados en la literatura que nos permiten obtener cotas inferiores (LB) y superiores (UB) de la solución óptima, así como soluciones óptimas. El capítulo 2 recoge todos estos conceptos.

En el capítulo 3 se enmarca el algoritmo A^* dentro de los métodos de búsqueda heurística en espacios de estados, describiendo al algoritmo, sus principales características y algunas de sus variantes. Por otro lado, se muestra el diseño sistemático de heurístico y algunas de las principales propiedades de los heurísticos.

El capítulo 4 se centra en la resolución de la versión del problema $J||C_{max}$ con el algoritmo A^*. Tras la formulación del problema, este capítulo se centra en el estudio de diferentes heurísticos y en el diseño de una técnica de poda por dominancia que constituye una de las aportaciones principales de esta tesis. La técnica de poda por dominancia se basa en la idea de que si se puede comprobar que la mejor solución alcanzable desde un estado n_2 no es mejor que la mejor solución alcanzable desde otro estado n_1, entonces el estado n_1 domina al estado n_2, estado que por tanto puede ser podado. En este capítulo se realiza un estudio experimental que permite comparar los distintos heurísticos diseñados y probar la eficiencia de la técnica de poda por dominancia propuesta.

La resolución del problema $J||\sum C_i$ con el algoritmo A^* es tratada en el capítulo 5. En él se formula esta versión del problema, se diseñan diferentes heurísticos y se adapta la técnica de poda por dominancia para su aplicación a esta versión del problema. Mediante un estudio experimental se testa la eficiencia tanto de los heurísticos como de la técnica de poda por dominancia.

Los capítulos 4 y 5 ponen de manifiesto que el límite de tamaño de problemas que somos capaces de resolver de forma óptima con el algoritmo A^* está próximo al tamaño 10×10. En ese tamaño umbral hay problemas para los que la memoria disponible se agota sin alcanzar una solución. Para poder ofrecer una solución aunque no sea la óptima hemos estudiado para ambas versiones del problema diversas técnicas, entre ellas el cálculo de cotas superiores que permiten podar el espacio de búsqueda y ofrecer una solución en cualquier momento, así como otras técnicas basadas en la relajación de la restricción de optimalidad, como por ejemplo la ponderación estática y la reducción

del espacio de búsqueda de planificaciones activas. El capítulo 6 recoge el estudio experimental que prueba todas estas técnicas en ambas versiones del problema.

Por último, el capítulo 7 recoge las principales conclusiones y las líneas de trabajo futuro que esta tesis deja abiertas.

Capítulo 2

PROBLEMAS DE SCHEDULING

2.1. Introducción

Dentro de la teoría de scheduling se pueden distinguir un gran número de diferentes tipos de problemas (ver, [4, 10, 19, 20, 30, 64, 70, 86]). En los problemas de scheduling se tienen un conjunto de N trabajos $\{J_1, \ldots, J_N\}$ que han de ser procesados sobre un conjunto de M recursos o máquinas físicas $\{R_1, \ldots, R_M\}$. Una planificación consiste en encontrar para cada trabajo un tiempo o un intervalo de tiempos en los que este puede procesarse en una o varias máquinas. El objetivo es encontrar una planificación sujeta a una serie de restricciones y que optimice una o varias funciones objetivo.

Los problemas de scheduling pueden emplearse para modelar procesos de producción industrial. Debido a ello estos problemas tienen gran interés, existiendo una serie de problemas, casos especiales del problema de planificación general (General Shop Scheduling Problem, $GSSP$), muy referenciados, entre ellos se encuentran: el Open Shop Scheduling Problem ($OSSP$), el Flow Shop Scheduling Problem ($FSSP$) y el Job Shop Scheduling Problem ($JSSP$).

El $GSSP$ se puede describir del siguiente modo: se tienen N trabajos $\{J_1, \ldots, J_N\}$ y M máquinas físicas $\{R_1, \ldots, R_M\}$. Cada trabajo J_i consta de un conjunto de tareas u operaciones $\theta_{ij} \in \{\theta_{i1}, \ldots, \theta_{iM}\}$ $(j = 1, \ldots, n_i)$, con unos tiempos de procesamiento $p_{\theta_{ij}}$. Cada operación θ_{ij} debe ser procesada sobre una máquina $\mu_{ij} \in \{R_1, \ldots, R_M\}$. Existen además relaciones de precedencia entre las operaciones de todos los trabajos. Cada trabajo sólo puede ser procesado sobre una máquina y cada máquina sólo puede procesar un trabajo en un instante. El objetivo es encontrar una planificación factible que optimice una función objetivo que depende del tiempo de fin de los trabajos.

Los problemas que derivan del $GSSP$, son casos especiales de este que añaden alguna restricción, en [49] podemos encontrar una clasificación de problemas de scheduling más extendida, siendo entre ellos los problemas más referenciados los siguientes:

- ***One Machine Sequence Problem (OMS):*** es un GPS en el que cada trabajo J_i consiste en 1 única operación con tiempo de procesamiento p_i donde θ_{i1} debe ser procesado sobre la

máquina R, es decir, cada trabajo tiene una única tarea que ha de ser procesada, según su prioridad, en la máquina R. El problema consiste en encontrar el mejor orden de procesamiento de los trabajos J_i, sobre la máquina R.

- ***Open Shop Scheduling Problem (OSSP):*** es un $GSSP$ en el que cada trabajo J_i consiste en M operaciones θ_{ij} que deben ser procesadas sobre la máquina R_j, no existiendo relaciones de precedencia entre las operaciones. El problema consiste en encontrar órdenes de los trabajos (ordenaciones de las operaciones pertenecientes al mismo trabajo) y órdenes de las máquinas (ordenaciones de las operaciones que han de ser procesadas en las misma máquina).

- ***Flow Shop Scheduling Problem (FSSP):*** es un $GSSP$ en el que cada trabajo J_i consiste en M operaciones $\theta_{ij} \in (j = 1, \ldots, M)$ con tiempos de procesamiento $p_{\theta_{ij}}$ donde todas las tareas θ_{ij} $(j = 1, \ldots N)$ debe ser procesadas en la misma máquina R_j. Hay restricciones de precedencia del tipo θ_{ij} antes que $\theta_{i(j+1)} (j = 1, \ldots M-1)$ para cada $i = 1, \ldots, N$, es decir, cada trabajo es procesado primero en la máquina 1, luego en la 2, luego la 3, etc.... El problema consiste en encontrar un orden de trabajos π_j, para cada máquina j.

- ***Job Shop Scheduling Problem (JSSP):*** es un GSP en el que cada trabajo J_i consiste en M operaciones $\theta_{ij} \in (j = 1, \ldots, M)$ con tiempos de procesamiento $p_{\theta_{ij}}$ donde cada tarea θ_{ij} $(j = 1, \ldots N)$ debe ser procesadas en una máquina distinta. Tiene tres restricciones: las tareas de los trabajo se han de ejecutar de forma secuencial, dos tareas que utilizan la misma máquina no pueden solapar sus tiempos y por último, una vez asignado el tiempo de inicio a una tarea, esta debe ser procesada en su totalidad. El problema consiste en encontrar un orden de tareas para cada máquina. Este tipo de problemas es el elegido para esta tesis, su descripción detallada puede encontrarse en el siguiente apartado.

2.2. Formulación del Problema Job Shop Scheduling Problem

El problema Job Shop Scheduling, conocido como JSS, consiste en planificar un conjunto de N trabajos $\{J_1, \ldots, J_N\}$ sobre un conjunto de M recursos o máquinas físicas $\{R_1, \ldots, R_M\}$. Cada trabajo J_i consta de un conjunto de tareas u operaciones $\{\theta_{i1}, \ldots, \theta_{iM}\}$ que han de ser ejecutadas de forma secuencial. Cada tarea θ_{il} tiene un tiempo de procesamiento de $p_{\theta_{il}}$ unidades de tiempo durante el cual requiere el uso exclusivo de un único recurso, $R_{\theta_{il}}$, a partir de un tiempo de inicio $st_{\theta_{il}}$ que se ha de determinar. 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 problema JSS tiene tres restricciones: restricciones secuenciales o de precedencia, restricciones de capacidad y restricciones de no-expulsión de las máquinas. Las restricciones secuenciales están definidas por la secuencia de tareas de un trabajo y se pueden describir mediante desigualdades lineales del tipo: $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$ (es decir la tarea θ_{il} ha de ejecutarse antes que la tarea $\theta_{i(l+1)}$). Las restricciones de capacidad restringen el uso de cada recurso a una única tarea en cada instante de tiempo y se pueden describir como una restricción disyuntiva de la forma: $st_v + p_v \leq$

$st_w \vee st_w + p_w \leq st_v$ si $R_v = R_w$ (dos tareas que emplean el mismo recurso no pueden solaparse). Por último las restricciones de no-expulsión, expresan que una vez determinado el tiempo de inicio $st_{\theta_{il}}$ para una tarea θ_{il}, esta tarea debe ser procesada durante el intervalo de tiempo $[st_{\theta_{il}}, st_{\theta_{il}} + p_{\theta_{il}}[$ sin interrupción.

El objetivo es encontrar una planificación óptima para un determinado criterio, siendo los más habituales los tres siguientes:

- ***Makespan:*** es el tiempo de finalización de la última tarea, tiempo denotado como C_{max}. Esta versión del problema es conocida, en la literatura, como $J||C_{max}$.
- ***Flujo Total:*** es la suma de los tiempos de fin de todos los trabajos. Esta versión se conoce como $J||\sum C_i$, donde C_i es el tiempo de finalización del trabajo J_i.
- ***Tardiness:***o tiempo de retardo de los trabajos. Esta versión se conoce como $J||\sum T_i$ donde T_i es el tardiness, $T_i = max\{0, C_i - d_i\}$ (d_i es el due date de la tarea i).

2.2.1. Representación del Problema

Los problemas de scheduling pueden representarse mediante un grafo dirigido $G = (V, A \cup E)$ definido del siguiente modo [71]:

- V: Conjunto de nodos que representan a las tareas del problema, con la excepción de los nodos ficticios *inicio*(O) y *fin*$(*)$, los cuales representan tareas con tiempo de procesamiento 0.
- A: Conjunto de arcos *conjuntivos*. Reflejan las relaciones de precedencia entre tareas, es decir, representan las restricciones secuenciales.
- E: Conjunto de arcos *disyuntivos*. Reflejan las relaciones entre las tareas que emplean el mismo recurso, es decir, las restricciones de capacidad. Este conjunto E se descompone en M subconjuntos E_i, uno por cada recurso R_i, de forma que $E = \cup_{\{i=1,\ldots,M\}} E_i$. El subconjunto E_i incluye un arco (v, w) por cada par de tareas v y w que requieren el recurso R_i.

Todos los arcos de los conjuntos A y E van etiquetados con unos pesos que representan el tiempo de procesamiento de la tarea de la que parte el arco. El nodo ficticio *inicio* está conectado con la primera tarea de cada trabajo y las últimas tareas de cada trabajo están conectadas con el nodo ficticio *fin*.

La Figura 2.1 muestra la representación de una instancia de un problema JSS con tres trabajos $\{J_1, J_2, J_3\}$ y tres recursos $\{R_1, R_2, R_3\}$ donde cada nodo representa una tarea y el recurso que requiere. El tiempo de inicio es 0 y el de fin 15 para todos los trabajos. Los arcos en línea continua representan los elementos del conjunto A y los arcos en línea discontinua representan los elementos del conjunto E.

2.2.2. Representación de las Soluciones

La representación de una planificación factible, se suele realizar mediante un subgrafo del grafo anterior o bien mediante un diagrama de Gantt.

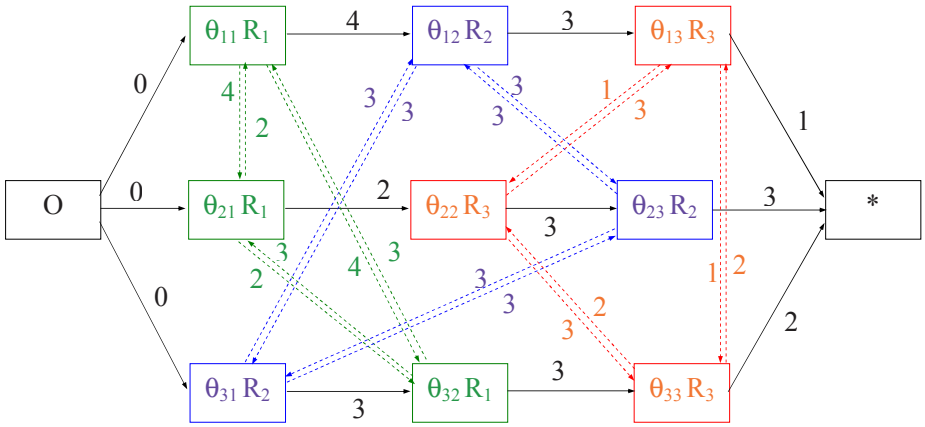


Figura 2.1: Representación en forma de grafo dirigido de un problema JSS con tres trabajos y tres recursos

Si se elige la representación mediante un grafo, la planificación factible se representaría como un subgrafo sin ciclos G_s de G, $G_s = (V, A \cup H)$, donde $H = \cup_{i=1,\ldots,M} H_i$, siendo H_i una selección hamiltoniana de E_i. H_i es una selección hamiltoniana de E_i, si para cada par de tareas u y v que requieren el recurso R_i, uno y sólo uno de los pares (u, v) o (v, u) pertenece a H_i. Si $(u, v) \in H_i$, entonces la tarea u se procesa antes que v en la solución que representa G_s. Dicho de otra forma H_i representa un orden total para las tareas que requieren R_i. Por lo tanto, encontrar una solución se puede reducir a encontrar una selección hamiltoniana o planificación parcial, que se traducirá en un grafo solución sin ciclos G_s.

Dependiendo de cual sea el criterio a optimizar el coste de la solución del problema se representará en el grafo de distinta forma, por ejemplo cuando el criterio de optimización es el flujo total una solución será la suma de los tiempos de completud de cada trabajo, siendo el tiempo de completud de un trabajo el coste del camino más largo del nodo inicial al final restringido a pasar por la última tarea del trabajo justo antes del nodo final. Cuando el objetivo es la minimización del makespan, el coste del *camino crítico* será el makespan de la planificación. Un camino crítico es la ruta más larga desde el nodo *inicial* hasta el nodo *final*. Un subconjunto maximal de tareas consecutivas en el camino crítico que se ejecutan sobre la misma máquina se denomina *bloque crítico*. Como veremos más adelante los bloques críticos tienen una importancia crucial en algunos esquemas de ramificación y poda, así como en los algoritmos de búsqueda local.

La Figura 2.2 muestra el grafo que representa una planificación factible para el problema de la Figura 2.1, en donde los arcos en negrita representan un camino crítico cuyo coste es 12, es decir, el makespan de esta planificación será 12. Por claridad, para cada H_i se indican únicamente los arcos (u, v), tales que u es la predecesora inmediata de v en el orden que representa H_i. El tiempo de flujo total es la suma de los tiempos de fin de las tareas de cada trabajo, calculados según las restricciones de la solución, en este caso 27.

Otra forma de representar planificaciones factibles es mediante el empleo de diagramas de Gantt, orientados a la máquina o al trabajo. La Figura 2.3 muestra la misma planificación factible que el

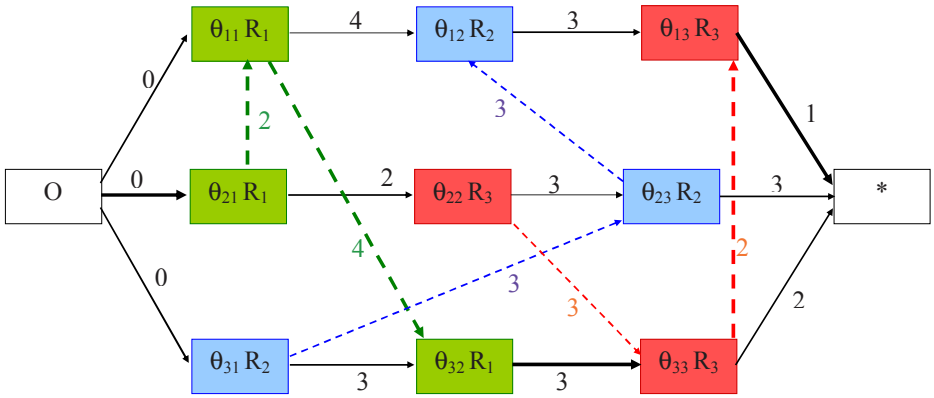


Figura 2.2: Planificación factible para el problema de la Figura 2.1

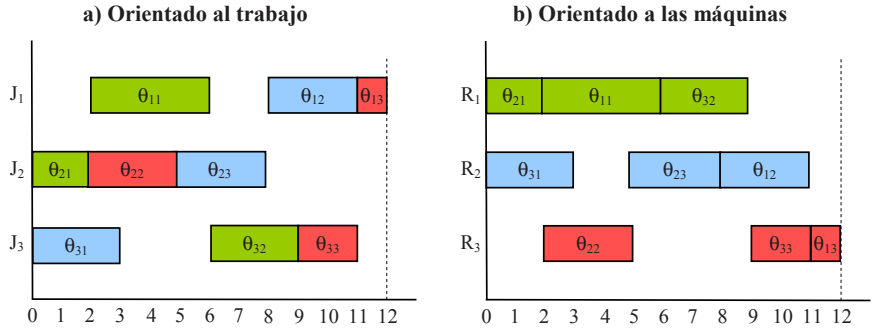


Figura 2.3: Planificación factible en forma de diagrama de Gantt para el problema de la Figura 2.1

grafo de la Figura 2.2, en forma de diagrama de Gantt orientado al trabajo y a la máquina.

2.2.3. Espacios de Soluciones

El espacio de planificaciones factibles completo es muy grande y contiene además muchas soluciones que no son realmente interesantes, ya que a partir de ellas se pueden obtener de forma inmediata otras que son mejores o iguales. En la práctica, se suelen considerar espacios de búsqueda más reducidos como por ejemplo los espacios de planificaciones *semiactivas*, *activas* y *densas*.

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 densa 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 densas son un subconjunto de las planificaciones activas, las cuales son a su vez son un subconjunto de las planificaciones semiactivas. La Figura 2.5 muestra, para un problema con 2 trabajos y 3 máquinas, una planificación semiactiva que no es activa, y una planificación activa que no es densa.

Para funciones objetivo como el makespan o el flujo total, la experiencia muestra que el valor

medio de la función de evaluación 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 densas. Al mismo tiempo sólo los espacios de planificaciones semiactivas y activas son dominantes, es decir, existe la garantía de que contienen al menos una solución óptima. La Figura 2.4 muestra la relación de inclusión existente entre los diversos tipos de planificaciones. Las planificaciones densas pueden contener o no alguna planificación óptima dependiendo de la instancia del problema.

De los espacios anteriores, el espacio de planificaciones activas es el más reducido entre los que contienen al menos una solución óptima. Además, la estrategia del conocido algoritmo *G&T* sugiere una forma de generar un árbol de búsqueda que representa a todos las planificaciones activas. Por estas razones en muchos algoritmos de búsqueda se elige este espacio de soluciones.

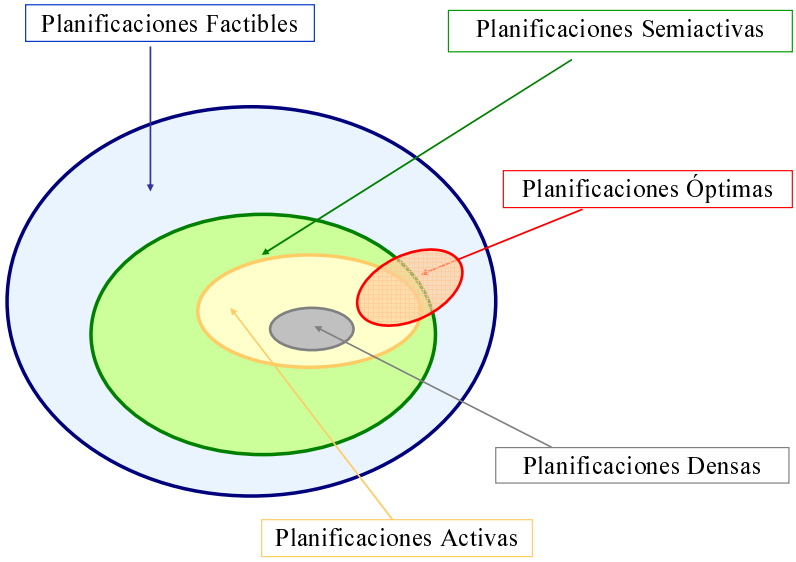


Figura 2.4: Relación entre los espacios de planificaciones

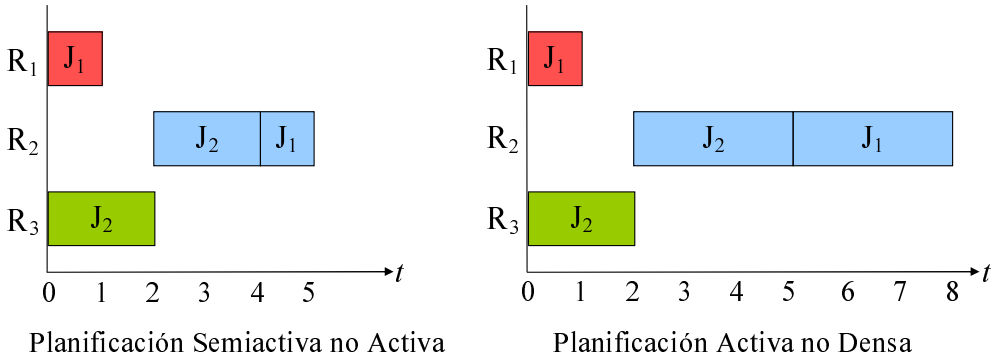


Figura 2.5: Ejemplos de planificaciones semiactiva y activa

2.3. Cálculo de Cotas Inferiores

La obtención de buenas cotas inferiores es fundamental para la eficiencia de muchos algoritmos de búsqueda. En esta sección vamos a revisar algunos de los métodos habituales de cálculo de cotas inferiores. Para ello, comenzaremos viendo cómo la relajación de restricciones en un problema JSS permite obtener un problema de secuenciamiento de una máquina OMS y cómo a partir de ese problema se puede obtener una cota inferior de la solución óptima del problema de partida JSS.

2.3.1. Relajación a Problemas de Secuenciamiento de una Máquina

Este problema tiene mucho interés en sí mismo y también porque se obtiene cuando se realizan relajaciones de las restricciones de capacidad en problemas JSS; se elige una máquina y se relajan las restricciones de capacidad del resto de las máquinas.

El problema de secuenciaciamiento de una máquina puede formularse de la siguiente manera: se tiene un conjunto de trabajos $\{J_1, \ldots, J_N\}$ y cada trabajo J_i debe ser procesado sin interrupción sobre una máquina m durante p_i unidades de tiempo. La máquina m no puede procesar más de una tarea al mismo tiempo.

El objetivo es planificar todos los trabajos de forma que se optimice la función objetivo, por ejemplo que el makespan sea mínimo.

Cada trabajo J_i, además de su propio tiempo de procesamiento p_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. Estos tiempos de procesamiento previo y posterior suelen denominarse *cabezas* y *colas*; tienen además un comportamiento simétrico, lo que hace que el problema dual OMS (problema de secuenciamiento de una máquina en el que r_i y q_i son intercambiados) también resulte de interés. Cuando se trata de una relajación de un problema JSS, la manera más simple de calcular estos valores es propagando las restricciones secuenciales en el grafo que representa al problema. Así, r_i será la suma de las duraciones de las tareas anteriores a i en su trabajo y q_i la suma de las duraciones de las tareas posteriores a i en su trabajo.

El problema OMS se representa también mediante un grafo en el que los nodos representan a las tareas; los arcos que parten del nodo inicial representan los tiempos r_i, los arcos que llegan al nodo final los tiempos q_i y los arcos discontinuos que unen las tareas representan las restricciones disyuntivas entre ellas (van etiquetados con el tiempo de procesamiento de la tarea de la que parten). La Figura 2.6 muestra un problema JSS y los problemas OMS obtenidos a partir del mismo tras una relajación.

El problema OMS así formulado es un problema $NP-duro$, por lo que no es posible calcular de forma eficiente una solución óptima del problema. Debido a esto se suele considerar una simplificación: se relajan las restricciones de no-expulsión de la máquina m. La relajación de estas restricciones permite que los trabajos puedan ser interrumpidos si se considera oportuno y restaurados más tarde. Si un trabajo ha sido interrumpido cuando se vuelve a planificar continuaría su ejecución desde el mismo punto en que fue interrumpido. El problema así relajado deja de ser $NP-duro$, siendo posible encontrar una solución óptima con un algoritmo de complejidad polinomial.

Para resolver óptimamente el problema OMS relajado se obtiene la Planificación con Prioridad

de Jackson (JPS, Jackson's Preemptive Schedule) [15]. Cuando se trata de minimizar el makespan, la planificación JPS es un orden de planificación en cuyo cálculo se utiliza una regla de prioridad que asigna mayor prioridad a la tarea con la cola más larga y que puede ser calculada mediante el algoritmo de Schrage, modificado ligeramente para que permita la expulsión de una tarea que se está planificando. Este algoritmo se verá en el siguiente apartado.

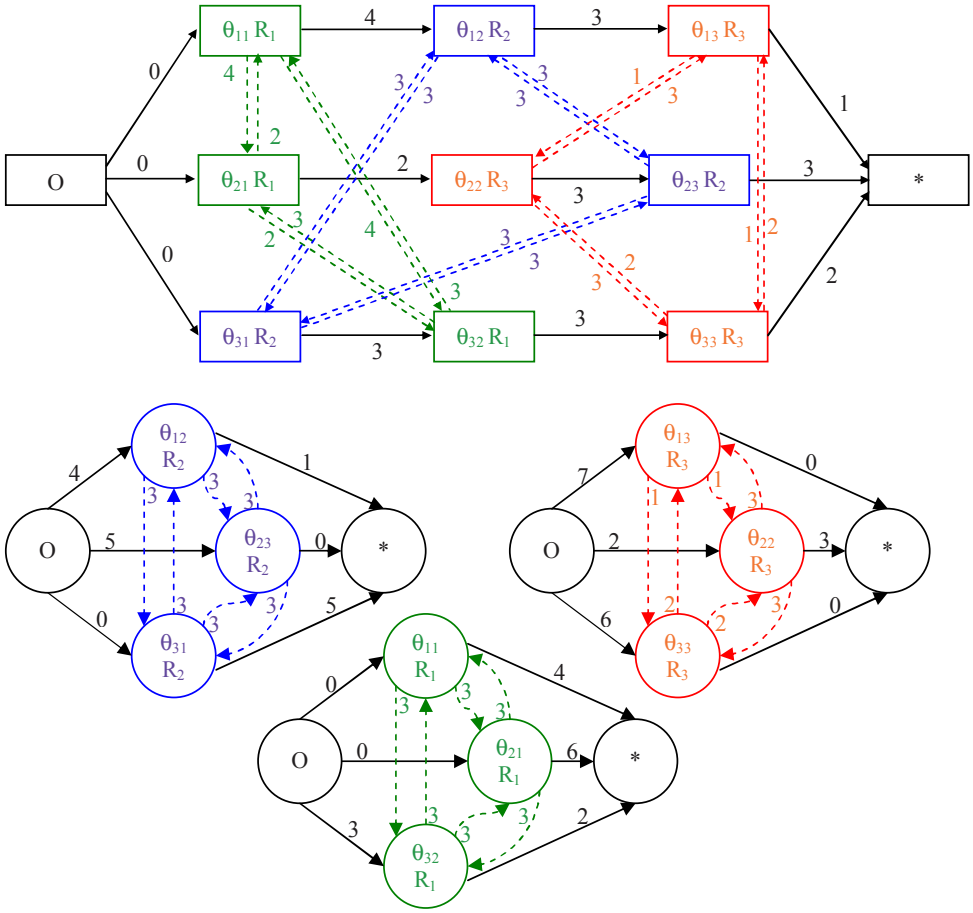


Figura 2.6: Problema JSS y problemas OMS correspondientes a cada una de las máquinas

Algoritmo de Schrage Modificado

Como se ha indicado anteriormente, el problema OMS es un problema $NP-duro$, por lo que para calcular una cota inferior o solución es necesario relajar sus restricciones de no-expulsión. En [15] Carlier propone el algoritmo de Schrage que construye planificaciones solución (cota superior) para problemas OMS, así como una modificación del mismo que permite construir planificaciones con expulsión. Además, también enuncia y prueba la proposición 2.3.1. Esta proposición nos demuestra, que la planificación construida mediante el algoritmo de Schrage modificado, aplicado a la relajación del problema OMS produce una planificación óptima del problema OMS relajado. El coste de esta planificación será, obviamente, una cota inferior de la solución del problema OMS sin

relajar, y por lo tanto cota inferior del problema JSS de partida.

Proposición 2.3.1. *El makespan de una planificación con expulsión JPS, para el problema OMS relajado, es óptimo.*

La forma en que este algoritmo construye la planificación JPS es relativamente sencilla. El algoritmo de Schrage modificado comienza asignando a cada trabajo J_i una prioridad proporcional a su q_i. Después, en cada instante de tiempo t, desde 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 $[t, t+p_i[$ o bien hasta que otro trabajo con mayor prioridad (mayor q_i) pueda comenzar, es decir, si un trabajo con mayor prioridad puede comenzar expulsa al que se estaba procesando. Un trabajo que ha sido expulsado, cuando vuelve a ser planificado continúa desde el punto en el que se había parado su planificación, por lo que es necesario guardar, en cada instante, el tiempo de proceso que le falta a un trabajo. En el caso de que la función objetivo sea el makespan, su valor será el máximo, de entre todos los trabajos, del tiempo de fin sobre m más el correspondiente q_i.

Como se ha indicado en la descripción del problema OMS, los tiempos r_i y q_i tienen un comportamiento simétrico, por lo que el problema OMS *Dual* resulta también de interés. Debido a esta simetría el algoritmo de Schrage modificado permite calcular también la planificación JPS Dual con solo intercambiar cabezas por colas. 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 ($PJPS$). En la Figura 2.7 se puede observar un problema OMS y sus correspondientes $PJPS$ y $DJPS$.

La implementación del algoritmo de Schrage modificado, utilizada en este trabajo, se basa en la propuesta por Carlier en [15]. Esta implementación permite construir tanto la planificación $PJPS$ como la $DJPS$ en $O(N * log_2 N)$ pasos, complejidad similar a una ordenación.

El algoritmo 2.1, muestra la implementación del algoritmo de Schrage modificado. En esta implementación se emplean los siguientes términos:

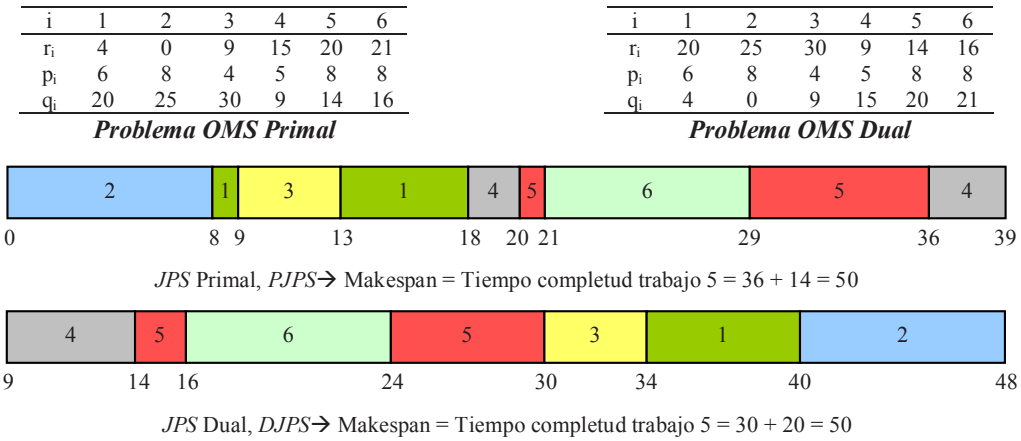


Figura 2.7: Problema OMS y sus correspondientes planificaciones JPS modificadas para los problemas Primal y Dual

Algoritmo 2.1 Algoritmo de Schrage

Requiere: Problema OMS

Produce: Planificación JPS para el problema OMS

1. Inicializaciones:

Ordenar todos los trabajos en orden creciente de cabezas (r_i);

$t = \text{mín}_{i \in U'} r_i$;

$m = 0$;

$p = [lof_2 N]$;

$x_0 = x_1 = \ldots = x_p = 0$;

$\alpha_0 = \alpha_1 = \ldots = \alpha_p = 0$;

$falta_1 = falta_2 = \ldots = falta_N = p_j$, ($p_j$, tiempo de procesamiento trabajo j) ;

2. El trabajo $m+1$ pertenece a S?

si $r_{m+1} > t$ **entonces**

Ir al paso 4;

si no

Ir al paso 3;

fin si

3. Inserción de un trabajo en S

Asignar a la partición vacía de menor número la unión del trabajo m y del resto de particiones de menor número, es decir $S_{k_0} = m \cup S_0 \cup S_1 \cup \ldots \cup S_{k_0 - 1}$;

Actualizar el trabajo ($m++$);

Volver al paso 2;

4. Selección de un trabajo j

si Todos los trabajos han sido ya planificados **entonces**

Terminar;

fin si

si Todos los trabajos de S han sido planificados **entonces**

Actualizar $t = r_{m+1}$ e ir al paso 2;

si no

De entre todos los conjuntos S_k no vacíos y con trabajos sin planificar elegir el trabajo j con el mayor q_j posible, de entre los no planificados;

fin si

5. Planificación del trabajo j

Ordenar todos los trabajos en orden creciente de cabezas (r_i);

$t_j = t$;

si $\neg \exists k \in U' / r_k < t_j + falta_j$ y $q_k > q_j$ **entonces**

$t = \text{máx}(t_j + falta_j, \text{mín}_{i \in U'} r_i)$;

si no

$t = r_k$;

fin si

Ir al paso 2;

- S: Conjunto que contiene a todos los trabajos con tiempo $r_i \leq t$, t es el instante actual.
- m: Cardinalidad del conjunto S.
- $x_p \dots x_1 x_0$: Expresión binaria de m, indica qué particiones se considerarán vacías y qué particiones se considerarán ocupadas.
- p: Parte entera del logaritmo en base 2 del número total de trabajos N ($p = \lceil log_2 N \rceil$). S se dividirá en $p+1$ particiones.
- $S_0, S_1, \dots, S_p$: $p+1$ conjuntos en los que se particiona S. Las cardinalidades de estos conjuntos serán $2^p x_p, \dots, 2^1 x_1, x_0$. Una partición S_k estará vacía cuando $x_k = 0$. Cada conjunto S_k estará ordenado del siguiente modo: primero estarán los trabajos que hayan terminado su ejecución y después el resto de trabajos ordenados de mayor a menor tiempo posterior.
- $\alpha_0 \alpha_1 \dots \alpha_p$: α_k será nulo si la partición k es vacía o si todos los trabajos de esa partición han terminado. En otro caso contendrá la posición del primer trabajo no terminado dentro de la partición. Estos valores permitirán saber cuál será el siguiente trabajo a planificar de ese conjunto, ya que los de número inferior ya habrán terminado y los de número superior tendrán menor prioridad.

Como se observa en el paso 3 del algoritmo, cada vez que un nuevo trabajo se introduce en S, será colocado en la partición vacía de menor número posible, asignando a esa partición la unión del nuevo trabajo y de todas las particiones con número inferior a ella. Las principal modificación que incluye el algoritmo de Schrage para ser aplicado a un problema OMS sin restricciones de no-expulsión se encuentra en el paso 5: un trabajo se ejecutará hasta que termine o hasta que otro trabajo con mayor prioridad pueda entrar en el conjunto S. En el segundo caso se volverá al punto 2 del algoritmo y no se marcará como finalizado el trabajo que se estaba planificando. Un trabajo que ha sido expulsado de su procesamiento, continúa su planificación cuando el nuevo trabajo que se puede empezar a procesar tiene una prioridad (q_i) menor que la suya. Un trabajo que ha sido expulsado continuará cuando sea el de mayor prioridad de los que pueden comenzar en un instante posterior. En ese instante continúa en el punto en el que fue interrumpido por lo que es necesario guardar, en cada instante, el tiempo de procesamiento que le falta a un trabajo. La Figura 2.8 muestra la traza del algoritmo que construye la planificación para el problema primal relajado de la Figura 2.7. Se partirá de un vector J con los trabajos ordenados.

El algoritmo de Schrage modificado tiene una complejidad $O(N * log_2 N)$:

- Inicializaciones. La operación más costosa de este apartado es la ordenación (por menor tiempo anterior r_i y ante igualdad por mayor tiempo posterior q_i). Esta operación puede realizarse mediante el algoritmo de ordenación Quicksort que tiene en el peor de los casos una complejidad $O(N log_2 N)$.
- El trabajo $m+1$ pertenece a S?. Este test se hará como máximo N veces, luego el coste será $O(N)$.

- Inserción de un trabajo en S. La inserción de un trabajo en S implica realizar una unión de conjuntos o de un elemento y un conjunto, además la unión de los conjuntos se ha de realizar siguiendo un orden determinado (primero los trabajos que hayan terminado su ejecución y después el resto de trabajos, ordenados de mayor a menor q_i). Cada trabajo se ordenará un máximo de p veces, luego el coste total será $O(Np) = O(Nlog_2N)$.
- Selección de un trabajo j. La selección del trabajo j se hará determinando el mínimo entre los $p+1$ elementos, luego el coste total es $O(Nlog_2N)$.
- Planificación del trabajo j. Sólo se actualizan variables, por lo tanto su coste es constante.

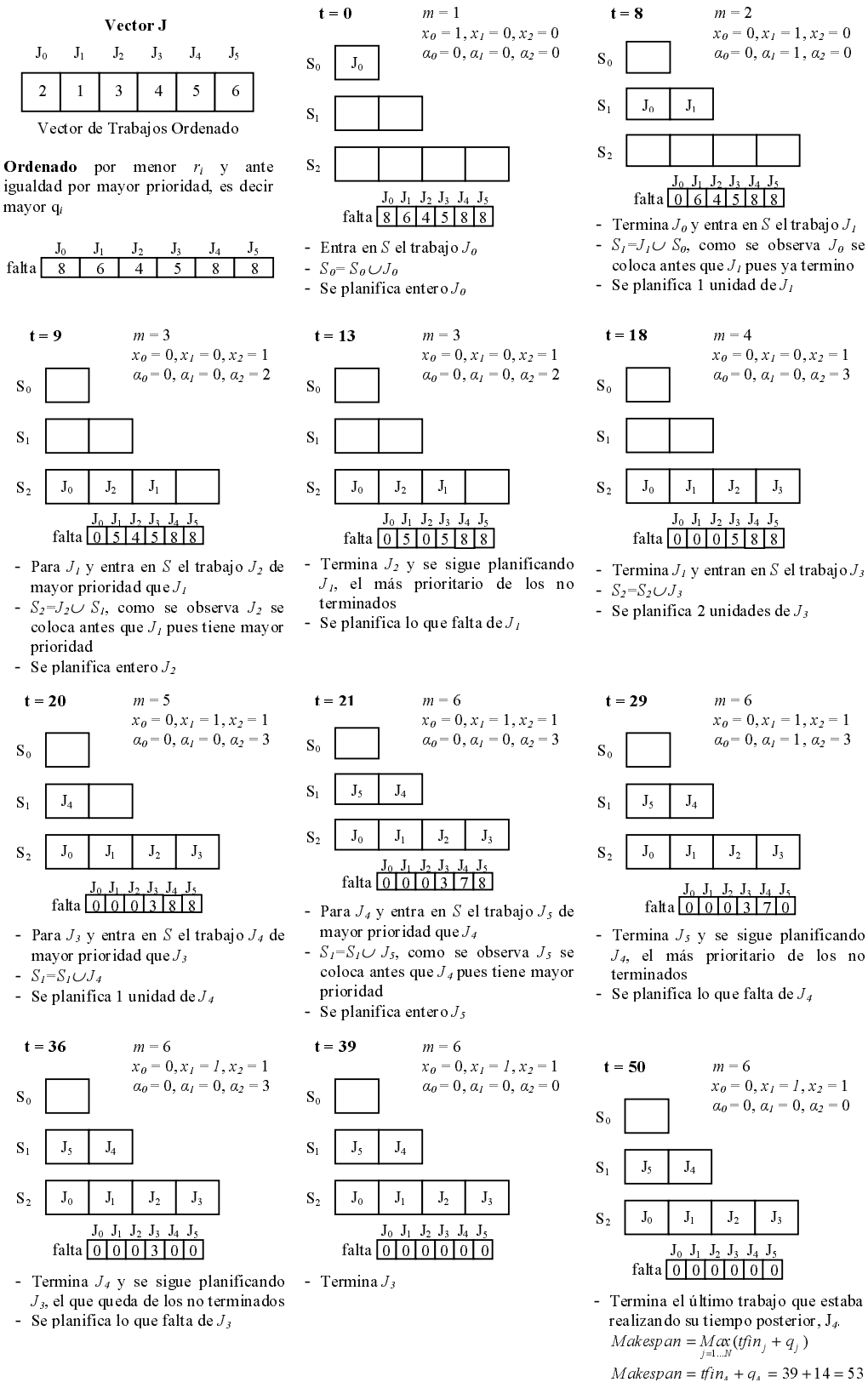


Figura 2.8: Traza del Algoritmo de Schrage modificado

2.4. Cálculo de Cotas Superiores

Las cotas superiores están asociadas a soluciones reales de un problema completo. Normalmente para el cálculo de cotas superiores se pueden emplear técnicas que no garantizan alcanzar la solución óptima pero sí buenas soluciones en poco tiempo. En está sección veremos algunos de los métodos más habituales de cálculo de cotas superiores para el problema $J||C_{max}$, concretamente: un algoritmo voraz, un algoritmo genético, un algoritmo de búsqueda local, otros métodos heurísticos como son los shifting bottleneck y las reglas de prioridad.

2.4.1. El Algoritmo *G&T*

El algoritmo *G&T* fue propuesto por B. Giffler y G. L. Thomson en el año 1960 [31]. Se trata de un algoritmo voraz que produce planificaciones activas en $N * M$ pasos. Su estrategia se muestra en el Algoritmo 2.2, como se puede observar es un algoritmo de tipo no determinista que puede ser particularizado de muchas formas.

En cada paso el algoritmo *G&T* realiza una elección no determinista sobre el 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 se consideran todas las elecciones posibles se tiene 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**. Este es el espacio de búsqueda utilizado en [83, 37, 91, 77, 76, 78].

La Figura 2.9 muestra un ejemplo para un problema de 4 trabajos y 4 máquinas, en el que para una situación intermedia hay 2 opciones posibles. La tarea θ_{32} es la que tiene menor tiempo de fin entre las que pueden ser planificadas. La tarea θ_{11} requiere la misma máquina que θ_{32} y puede empezar antes del tiempo de fin de la tarea θ_{32}. Por tanto el algoritmo *G&T* contemplará en el conjunto B las tareas que pueden empezar en el intervalo $[T, C[$ y que requieren la misma máquina que la tarea θ_{32}, es decir la propia tarea θ_{32} y la θ_{11}. Estas son las dos elecciones posibles que puede hacer el algoritmo *G&T* a continuación.

Algoritmo 2.2 Algoritmo *G&T*

Requiere: Problema *JSS*
Produce: Solución problema *JSS*

1: A = Conjunto con las primeras tareas sin planificar de cada trabajo;
2: **mientras** $(A \neq \oslash)$ **hacer**
3: Determinar la tarea $\theta' \in A$ que si se planifica en el estado n, tiene el menor tiempo de fin, es decir $st_{\theta'} + p_{\theta'} \leq st_\theta + p_\theta, \forall \theta \in A$;
4: Construir el conjunto B con las tareas de A que requieren la máquina $R_{\theta'}$ y que pueden comenzar antes que el tiempo de fin de la tarea θ';
$B = \{\theta \in A, R_\theta = R_{\theta'} \wedge st_\theta < st_{\theta'} + p_{\theta'}\}$;
{De este modo sea cual sea la tarea de B elegida para ser planificada, se obtiene finalmente una planificación activa}
5: Seleccionar θ^* de B con *algún criterio* y planificarla;
6: Borrar θ^* de A y añadir a A la sucesora de θ^* en caso de que exista, es decir, si no es la última tarea de su trabajo;
7: **fin mientras**

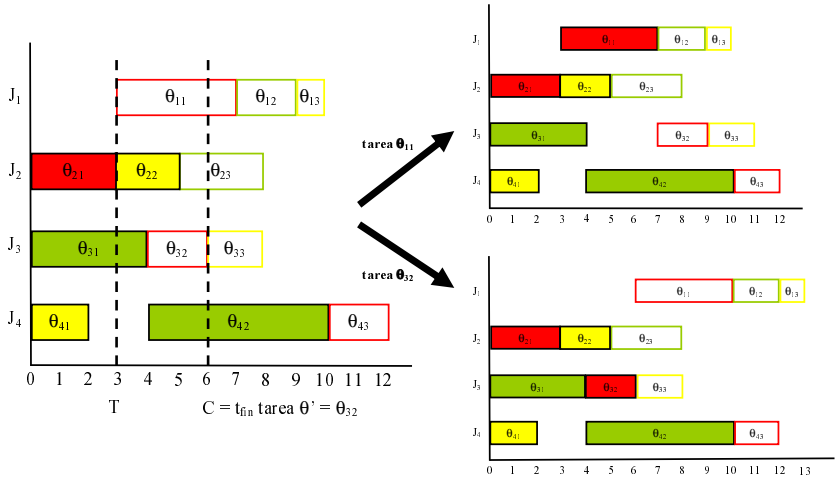


Figura 2.9: Elección de las tareas a planificar por el Algoritmo *G&T*

Algoritmo 2.3 Algoritmo *G&T* Híbrido

Requiere: Problema *JSS*

Produce: Solución problema *JSS*

1: A = Conjunto con las primeras tareas sin planificar de cada trabajo en el estado n;
2: **mientras** $(A \neq \oslash)$ **hacer**
3: Determinar la tarea $\theta' \in A$ que si se planifica en el estado n, tiene el menor tiempo de fin, es decir $st_{\theta'} + p_{\theta'} \leq st_\theta + p_\theta, \forall \theta \in A$;
4: Construir el conjunto B con las tareas de A que requieren la máquina $R_{\theta'}$ y que pueden comenzar antes que el tiempo de fin de la tarea θ';
$B = \{\theta \in A, R_\theta = R_{\theta'} \wedge st_\theta < st_{\theta'} + p_{\theta'}\}$;
{De este modo sea cual sea la tarea de B elegida para ser planificada, se obtiene finalmente una planificación activa}
5: Reducir el conjunto B a B'
$B' = \{\theta \in B / st_\theta < st_{\theta''} + \delta((st_{\theta'} + p_{\theta'}) - st_{\theta''})\}, \delta \in [0, 1]$;
6: Seleccionar θ^* de B con *algún criterio* y planificarla;
7: Borrar θ^* de A y añadir a A la sucesora de θ^* en caso de que exista, es decir, si no es la última tarea de su trabajo;
8: **fin mientras**

A partir de esta forma de proceder es simple ver que el espacio de búsqueda así generado es un árbol. La Figura 2.10 muestra el árbol de búsqueda completo que generaría el algoritmo *G&T* para un problema con 2 trabajos y 3 máquinas. Para cada nodo del árbol se muestra en color claro y con línea discontinua las siguientes tareas a planificar en cada trabajo (conjunto A), así como las tareas que darán lugar a sucesores (conjunto B). Como se ve, para este ejemplo, se llega a la solución óptima por la rama izquierda del árbol.

En problemas de gran tamaño el espacio de búsqueda formado por todas las planificaciones activas es excesivamente amplio, por lo que la búsqueda se debe restringir a una parte de este espacio por razones de eficiencia, con la consiguiente pérdida de la completud. En este sentido, una posibilidad es emplear una variante del algoritmo *G&T* denominada *G&T Híbrido*. ([9, 83]).

El algoritmo *G&T* híbrido, Algoritmo 2.3, 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 $\delta \in [0, 1]$, parámetro que permite restringir los intervalos de inactividad de las máquinas cuando hay tareas disponibles.

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 otro caso, si $\delta < 1$, el espacio de búsqueda se restringe a un subconjunto de las planificaciones activas; lo que hace imposible garantizar que se pueda alcanzar una solución óptima. Tal y como indica Bierwirth en [9], el algoritmo produce planificaciones densas en el extremo $\delta = 0$.

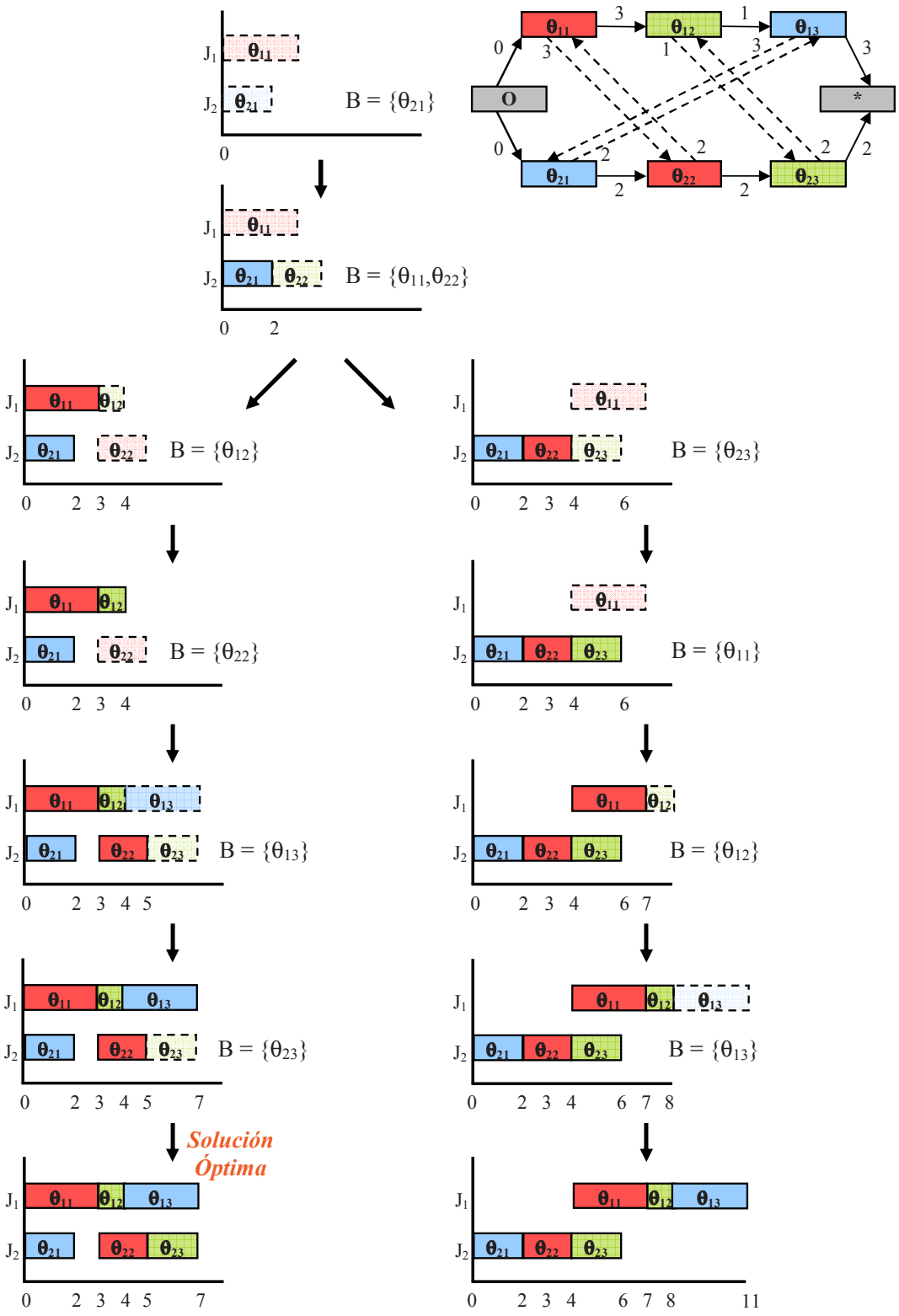


Figura 2.10: Funcionamiento del Algoritmo *G&T*

2.4.2. Algoritmos Evolutivos

Los Algoritmos Evolutivos (AEs) son algoritmos de búsqueda y optimización cuyo mecanismo de funcionamiento está basado en el proceso de evolución natural de las especies, en particular en la selección natural y en la herencia genética. Los AEs fueron introducidos por John Holland y algunos de sus colaboradores de la Universidad de Michigan en los años 70. El objetivo que se plantearon fue por una parte abstraer y explicar el proceso adaptativo de los sistemas naturales, y por otra tratar de diseñar sistemas artificiales que emulasen los mecanismos esenciales de los sistemas naturales. De este modo consiguieron crear sistemas de cómputo artificiales con características propias de los sistemas naturales, tales como la robustez, flexibilidad, auto-organización y reproducción. La estructura de un AE se puede ver en el Algoritmo 2.4. Un AE es un algoritmo de carácter probabilista que mantiene una población de individuos o cromosomas, $P(t)$, en cada iteración t. Cada uno de los individuos constituye una solución potencial del problema, representada normalmente mediante una cadena de símbolos. Los individuos se evalúan mediante la función fitness que determina la calidad de la solución o grado de adaptación del individuo al entorno. Posteriormente, se construye una nueva población (iteración $t+1$) comenzando con un proceso de selección a partir de la población $P(t)$, de manera que los mejores individuos tienen mayor probabilidad de ser seleccionados. A continuación se altera esta nueva población mediante la aplicación sobre algunos individuos de los operadores genéticos. Estos operadores pueden ser unarios (mutaciones), y en este caso producen un nuevo individuo mediante pequeños cambios en el individuo sobre el que se aplican, o de aridad superior (cruces) de manera que dos o más individuos se combinan para producir nuevos individuos. Así se obtiene finalmente una nueva población $P(t+1)$. Mediante este proceso de evolución simulada se espera que las poblaciones vayan mejorando en media a lo largo de las sucesivas generaciones, con lo que también parece razonable esperar que se vayan obteniendo cada vez mejores soluciones del problema. Las principales componentes de un AE son las siguientes: un método de codificación de los individuos o soluciones potenciales del problema, una función de evaluación (fitness) o una forma de generar la población inicial, operadores genéticos de selección, cruce y mutación; así como una serie de parámetros: probabilidad de cruce y mutación, tamaño de la población, número de generaciones, pruebas, etc...

Algoritmo 2.4 Algoritmo Evolutivo

Requiere: Problema JSS
Produce: Solución problema JSS

1: Leer Parámetros (P_c, P_m, $NroGen$,...);
2: $t = 0$;
3: Iniciar($P(t)$);
4: Evaluar($P(t)$);
5: **mientras** no última generación **hacer**
6: $t = t + 1$;
7: $P'(t)$ = Selección($P(t-1)$); *{Op. Selección}*
8: $P''(t)$ = Alterar($P'(t)$); *{Ops. Cruce y Mutación}*
9: Evaluar($P''(t)$); *{Función Fitness}*
10: $P(t)$ = Aceptar($P'(t)$, $P''(t)$); *{Op.de aceptación}*
11: **fin mientras**

Un caso particular de los AEs son los Algoritmos Genéticos (AGs). La característica esencial de un AG es el uso de operadores de cruce de dos o más cromosomas para obtener descendientes. En líneas generales la estrategia operativa de un AG consiste en partir de una población inicial de cromosomas, generada aleatoriamente, cada uno de los cuales representa una posible solución del problema. Estos cromosomas se evalúan mediante una función ($fitness$) que indica la calidad de la solución o grado de adaptación del individuo al entorno. A partir de esta situación inicial se realizan una serie de iteraciones en cada una de las cuales se simula la creación de una nueva generación de individuos a partir de la generación anterior. Este proceso consiste en aplicar los operadores genéticos de selección, cruce, mutación y aceptación sobre los individuos. Hasta llegar finalmente a una población que, si el algoritmo converge adecuadamente, estará compuesta por buenos individuos. En principio el mejor de todos los cromosomas evaluados es la solución que proporciona el AG. Un AG realiza una búsqueda multidireccional. Esta búsqueda es más intensiva en las direcciones más prometedoras, pero se dedica también una parte del esfuerzo a considerar regiones en principio de menor interés. En cada generación los individuos buenos se reproducen mientras que los malos se mueren; en este proceso se generan individuos nuevos que heredan algunas características de sus progenitores mientras que otras son propias de cada individuo. Así pues podemos decir que los algoritmos genéticos tratan de alcanzar un equilibrio entre dos objetivos aparentemente en conflicto: explotar las buenas cualidades de las potenciales soluciones y explorar el espacio de búsqueda.

Los Algoritmos Genéticos (AGs) [8, 9, 31, 51, 92] constituyen un método muy prometedor ya que se pueden combinar fácilmente con otras técnicas tales como la búsqueda tabú o el enfriamiento simulado. Además, los AGs permiten explotar cualquier clase de conocimiento heurístico sobre el dominio del problema. De este modo son realmente competitivos con los mejores métodos para resolver el problema JSS.

Veamos ahora un AG convencional para resolver el problema JSS, concretamente el que se propone en [32]. La codificación de cromosomas se hace con el método de permutaciones con repetición propuesto por Bierwirth en [8]: un cromosoma es una permutación del conjunto de tareas, en la que cada tarea viene representada por el número de su trabajo. De este modo el número de un trabajo aparecerá en el cromosoma tantas veces como tareas tenga el trabajo. Por ejemplo, el cromosoma $(2, 1, 1, 3, 2, 3, 1, 2, 3)$ representa la permutación de operaciones (θ_{21}, θ_{11}, θ_{12}, θ_{31}, θ_{22}, θ_{32}, θ_{13}, θ_{23}, θ_{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 [90] 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.

La decodificación del cromosoma en una planificación depende del algoritmo de decodificación. Para ello, como algoritmo de evaluación elegimos el algoritmo $G\&T$ propuesto en [31] y descrito en la sección 2.4.1 de este capítulo. Así, el proceso de decodificación consiste en ejecutar este algoritmo

(Algoritmo 2.5) y elegir la tarea θ^* 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. Cabe destacar aquí que el AG empleando este algoritmo de decodificación se mueve en el espacio de las planificaciones activas, siempre que el parámetro δ tenga valor 1. Es fácil diseñar un algoritmo de decodificación, con menor complejidad que el algoritmo $G\&T$, sobre el espacio de planificaciones semi-activas, sin embargo, una planificación semi-activa es peor o igual que una planificación activa para el mismo cromosoma.

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 (P_c) y mutación (P_m). De acuerdo con nuestra experiencia, el mejor operador de cruce es el JOX (Job Order Crossover) descrito en [8]: 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. El operador de mutación consiste simplemente 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).

Algoritmo 2.5 Algoritmo $G\&T$ híbrido para la de Decodificación de Cromosomas

Requiere: Cromosoma para un problema JSS

Produce: Planificación que representa el cromosoma de entrada

Sea $A = \{\theta_{j0}, 0 \leq j < n\}$; *{n número de trabajos}*

mientras $A \neq \emptyset$ **hacer**

$\forall\theta \in A$ sea $st\theta$ el tiempo de inicio más temprano de θ si se planifica a continuación;

Sea $\theta_1 \in A$ tal que $st\theta_1 + p\theta_1 \leq st\theta + p\theta, \forall\theta \in A$;

Sea $M = MR(\theta_1)$; *{$MR(\theta)$ máquina requerida por la tarea θ}*

Sea $B = \{\theta \in A : MR(\theta) = M, st\theta < st\theta_1 + du\theta_1\}$;

Sea $\theta_2 \in B$ tal que $st\theta_2 \leq st\theta, \forall\theta \in B$; *{tiempo de inicio más temprano para cada tarea B, si se selecciona la siguiente, valor del intervalo $[st\theta_2, st\theta_1 + du\theta_1]$}*

Reducir el conjunto B: $B = \{\theta \in B : st\theta \leq st\theta_2 + \delta((st\theta_1 + du\theta_1) - st\theta_2), \delta \in [0,1]\}$;*{se reduce el intervalo a $[st\theta_2, st\theta_2 + \delta((st\theta_1 + du\theta_1) - st\theta_2)]$}*

Seleccionar la tarea $\theta^\star \in B$ situada más a la izquierda en el cromosoma y planificarla en el instante $st\theta^\star$;

Sea $A = A \setminus \{\theta^\star\} \cup \{SUC(\theta^\star)\}$; *{$SUC(\theta)$ es la operación siguiente a θ en su trabajo, si existe}*

fin mientras

2.4.3. Métodos de Mejora Iterativa o Búsqueda Local

La estructura básica de un método de búsqueda local es la que se muestra en el Algoritmo 2.6. El algoritmo parte de una solución inicial y en cada iteración calcula un conjunto de soluciones vecinas mediante una regla de vecindad. Cada una de estas soluciones debe ser evaluada, siendo ésta una de las acciones más críticas del algoritmo por el elevado tiempo de ejecución que puede suponer, especialmente si el número de soluciones vecinas es muy grande, o bien si el procedimiento de evaluación de cada solución es muy costoso. A continuación se selecciona una de las soluciones vecinas con un determinado criterio, normalmente la que tiene menor coste. Si esta solución cumple

el criterio de aceptación, que puede ser simplemente que sea mejor que la solución actual S, la solución seleccionada reemplaza a la solución S, y el proceso continúa hasta que se cumpla el criterio de finalización. Este criterio suele ser que se agote un determinado número de iteraciones, o bien que no se produzcan mejoras en los últimos intentos.

Algoritmo 2.6 Esquema de un algoritmo de Búsqueda Local

Requiere: Solución inicial para un Problema JSS
Produce: Mejor Solución encontrada para el problema JSS

```
S = SolucionInicial;
mientras (no CriterioDeTerminación) hacer
  V = SolucionesVecinas(S);
  EvaluarSoluciones(V);
  S1 = Seleccion(V);
  si (CriterioDeAceptación) entonces
    S = S1;
  fin si
fin mientras
devuelve S;
```

Los algoritmos de búsqueda local realizan una búsqueda en un espacio de soluciones potenciales del problema tratando de encontrar aquella que maximice, o minimice, una determinada función objetivo. Esta función objetivo tiene normalmente una forma o relieve (que se suele denominar *landscape*) muy irregular, con numerosos máximos y mínimos locales, por lo que la búsqueda del óptimo global es muy costosa. Normalmente la búsqueda local no mantiene traza de todo el proceso realizado hasta el momento, sino que solamente utiliza información sobre el estado actual y sus vecinos. A pesar de esto, la búsqueda local es una estrategia muy eficaz en muchos casos. Además se puede combinar con otras estrategias, como por ejemplo los algoritmos evolutivos descritos en el apartado anterior (Sección 2.4.2).

Las tres variantes de búsqueda local más extendidas son:

- Los algoritmos de escalada o máximo gradiente: en éstos el criterio de aceptación es que la solución vecina S_1 sea mejor o igual que la solución actual S. Previamente en la selección se intenta elegir una solución que mejore a S, o bien la mejor de todas las soluciones de la vecindad. Esta búsqueda local tiene tres problemas fundamentales: óptimos locales (vecinos peores, se termina la búsqueda), regiones planas (vecinos con valor igual a solución actual, búsqueda aleatoria) y crestas (con pendientes marcadas se puede llegar a la cima, con cimas de pendientes suaves en la dirección del óptimo global, resulta difícil guiar la búsqueda sin desviarse a menos que existan operadores específicos). En los tres casos la búsqueda se queda estancada en un óptimo local, que puede ser una solución razonable o no serlo. Lo que se puede hacer en estos casos es reiniciar la búsqueda a partir de otra situación de partida, lo que se suele denominar *búsqueda multiarranque*, con lo que se alcanzaría posiblemente otro óptimo local distinto. Así, después de un número de intentos se podría llegar a una solución aceptable, sobre todo si el número de óptimos locales es pequeño. Sin embargo, para los problemas NP-duros, el número de óptimos locales suele ser un número exponencial, con lo

que las posibilidades de alcanzar un óptimo global son muy escasas.

- El temple simulado (o *simulated annealing*): este método consiste en admitir, con una cierta probabilidad, algunas transiciones en las que la solución actual empeore; de este modo se puede salir de óptimos locales. La principal diferencia de este algoritmo con respecto al algoritmo de escalada es la siguiente: el temple simulado realiza una selección aleatoria entre los vecinos del estado actual. Si esta solución es mejor que la actual, se realiza la transición de forma incondicional, como en el caso del algoritmo de escalada. Pero si la solución vecina es peor que la actual, entonces la nueva solución se acepta con una determinada probabilidad que depende de dos parámetros: la temperatura T y el incremento de energía ΔE. Al principio de la búsqueda, la temperatura tiene un valor alto, inicialmente T_0, de modo que la probabilidad de aceptar una solución peor que la actual es alta. A medida que la búsqueda progresa, el valor de T se va actualizando de forma $T = \alpha(t, T)$, siendo t la iteración actual y α una función que decrece al aumentar t. De este modo, la probabilidad de aceptar una solución que empeora a la actual va disminuyendo a medida que avanza la búsqueda, hasta que al final prácticamente solo se admiten soluciones que mejoren o igualen a la actual. El principal inconveniente que presenta este método es que requiere un ajuste de parámetros adecuado. Normalmente, este ajuste depende fuertemente del problema y hay que realizarlo de forma experimental. No obstante, el temple simulado es un método que resulta eficiente en muchos problemas.

- La búsqueda tabú: la diferencia esencial de esta búsqueda respecto a los métodos anteriores es que dispone de un mecanismo de memoria, con ello evita la generación de algunos vecinos dependiendo de la historia reciente, o de la frecuencia con la que se realizaron algunas transformaciones para llegar al estado actual (ejemplo: si la regla de vecindad es simétrica, y se genera x como vecino de y, suele ser buena idea recordarlo y no generar y inmediatamente a partir de x). En el caso más simple, el mecanismo de memoria se realiza mediante una estructura H que registra las transformaciones que dieron lugar a las últimas soluciones, de modo que estas transformaciones no se consideran en la generación de los vecinos de la solución actual, es decir se consideran tabú. El tamaño de la lista tabú debe ajustarse en función de las características del problema. En algunos casos, es posible que un movimiento de la lista tabú produzca en la solución actual una mejora, por este motivo se introduce lo que se denomina *criterio de aspiración* y que consiste en establecer excepciones a lo que indica la lista tabú, concretamente lo que se suele hacer es que si un movimiento tabú produce una solución que mejora a la mejor solución encontrada hasta el momento, este movimiento se aplica como si no estuviese en la lista. Al igual que el método anterior, el principal inconveniente de la búsqueda tabú es el ajuste de parámetros, como el tamaño de la lista tabú, la elección de los movimientos que se deben registrar y la definición del criterio de aspiración. También como en el caso del temple simulado, la búsqueda tabú se ha utilizado con éxito en muchos problemas, en particular cuando se combina con otras estrategias como por ejemplo los algoritmos evolutivos.

Veamos ahora como emplear una búsqueda local para la resolución de un problema JSS. En

muchas ocasiones, la eficiencia de un *AG* se puede mejorar si se combina con un método de búsqueda local, en este caso al algoritmo genético se le llama algoritmo Memético, esta es la aproximación que describiremos aquí y que es la que se utiliza en [32]. 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 este 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. Como estrategias de búsqueda local consideramos los esquemas de vecindad denominados N_1, N_2 y N_3, N_4, descritos por Mattfeld en [51]. Todas estas estrategias se basan en los conceptos de *camino crítico* y *bloque crítico*. Consideran cada *bloque crítico* de un *camino crítico* y hacen una serie de movimientos con las tareas de cada bloque crítico. Tras un movimiento dentro de un bloque crítico se debe testar la factibilidad. Estimar la factibilidad empleando un método exacto es computacionalmente prohibitivo, por ello la factibilidad se estima mediante un algoritmo aproximado propuesto por Dell' Amico y Trubian en [24]. Los esquemas de vecindad N_1, N_2, N_3 y N_4 se definen del siguiente modo:

Definición 2.1 (N_1). *Si la tarea n pertenece a un bloque crítico b de la forma $b = (wvb')$. En una solución vecina v se mueve delante de w, salvo que sea el primer bloque del camino crítico. Análogamente, si el bloque b es de la forma $b = (b'vw)$, v se mueve detrás de w salvo que b sea el último bloque crítico.*

El esquema de vecindad N_1 es una simplificación de la estrategia propuesta por Van Laarhoven y sus colaboradores en [89]. Con esta estrategia se invierten todos los pares de operaciones consecutivas en cualquier bloque crítico, con lo que el número total de vecinos es mucho mayor.

Definición 2.2 (N_2). *Si la tarea v pertenece a un bloque crítico b de la forma $b = (b'vb'')$. En una solución vecina, v se mueve al inicio de b o al final b siempre que se conserve la factibilidad, en otro caso v se mueve a la posición más próxima a la primera o la última tarea de b en la que se conserve la factibilidad.*

Definición 2.3 (N_3). *Sean v y w dos tareas seguidas en el camino crítico, y sean PM_v y SM_w los predecesores en la máquina de v y los sucesores en la máquina de w respectivamente. Si PM_v y SM_w no pertenecen al mismo bloque que v y w, entonces la única permutación es (w, v), en otro caso todas las posibles permutaciones de PM_v, v, w y v, w, SM_w son consideradas como vecinos si v y w son también invertidas.*

Teniendo en cuenta que ni $N_2 \subset N_3$ ni $N_3 \subset N_2$ es razonable definir la vecindad N_4 como:

Definición 2.4 (N_4). $N_4 = N_3 \cup N_2$.

El criterio de aceptación se basa en estimaciones del makespan. Esta estimación se puede calcular en tiempo constante, en lugar de calcular el makespan exacto de cada vecino, mediante el método que se describe en el párrafo siguiente. Esta estimación proporciona una cota inferior del makespan. El vecino seleccionado es aquel con menor estimación siempre que este valor sea menor que el makespan del cromosoma actual.

Estimación del Makespan. La estimación del makespan se basa en las cabezas y las colas. Consideremos en primer lugar la situación en la que el arco *(v,w)* es invertido. Debemos destacar que PM_v o SM_w no son críticos, en otro caso invertir el arco *(v,w)* no produce mejora. Los valores de r'_w, r'_v, q'_v, q'_w tras invertir el arco *(v,w)* son

$$r'_w = max(r_{PM_v} + p_{PM_v}, r_{PJ_w} + p_{PJ_w})$$

$$r'_v = max(r'_w + p_w, r_{PJ_v} + p_{PJ_v})$$

$$q'_v = max(q_{SM_w} + p_{SM_w}, q_{SJ_v} + p_{SJ_v})$$

$$q'_w = max(q'_v + p_v, q_{SJ_w} + p_{SJ_w})$$

Por tanto el makespan puede ser estimado del siguiente modo

$$C'_{max} = max(r'_w + p_w + q'_w, r'_v + p_v + q'_v)$$

Consideremos ahora el caso general de N_2 donde no se mueven únicamente tareas que van seguidas. Sea $CB = (O_1, ..., O_i, ..., O_j, ..., On)$ un bloque crítico de la máquina m y supongamos que la tarea O_j se mueve antes que O_i. Por tanto en la solución vecina tendremos la secuencia $(O_1, ..., O_{i-1}, O_j, O_i, ..., O_{j-1}, O_{j+1}..., On)$ sobre la máquina m. La cabeza de las tareas $(O_1, ..., O_{i-1})$ y la cola de las tareas $(O_{j+1}..., On)$ no cambian con respecto a la solución original. Sin embargo, la cola de las tareas $(O_1, ..., O_{i-1})$ y la cabeza de las tareas $(O_{j+1}..., On)$ en general cambia. Pero es este caso, ni los valores exactos ni las cotas inferiores de esas nuevas cabezas y colas pueden ser estimados sin calcular las nuevas cabezas y colas para el conjunto completo de todas las tareas, lo cual consume mucho tiempo. Por el contrario, los valores exactos de las nuevas cabezas y colas de las tareas $(O_j, O_i, ..., O_{j-1})$ puedes ser eficientemente calculadas como se indicará más adelante. Por lo tanto, estos nuevos valores pueden ser empleados para la estimación del makespan de las soluciones vecinas. Un razonamiento análogo se puede hacer si movemos O_j hacia el final del bloque crítico CB.

Ahora, el conjunto $(O_j, O_i, ..., O_{j-1})$ se denota $(L_1, ..., L_l)$, y O_{i-1} y O_{j+1} se denotan *primera* y *ultima* respectivamente. Como $r_{primera}$ y q_{ultima} no cambiarán de la solución original a un vecino, la estimación del makespan puede ser calculada del siguiente modo:

$$r'_{L_1} = max(r_{primera} + p_{primera}, r_{PJ_{L_1}} + p_{PJ_{L_1}})$$

$$r'_{L_2} = max(r'_{L_1} + p_{L_1}, r_{PJ_{L_2}} + p_{PJ_{L_2}})$$

$$\dots$$

$$r'_{L_l} = max(r'_{L_{l-1}} + p_{L_{l-1}}, r_{PJ_{L_l}} + p_{PJ_{L_l}})$$

$$q'_{L_l} = max(q_{ultima} + p_{ultima}, q_{SJ_{L_l}} + p_{SJ_{L_l}})$$

$$q'_{L_{l-1}} = max(q'_{L_l} + p_{L_l}, q_{SJ_{L_{l-1}}} + p_{SJ_{L_{l-1}}})$$

$$\ldots$$

$$q'_{L_1} = max(q_{L_2} + p_{L_2}, q_{SJ_{L_1}} + p_{SJ_{L_1}})$$

$$C'_{max} = r'_{L_1} + p_{L_1} + q'_{L_1}$$

$$C'_{max} = max(C'_{max}, r'_{L_2} + p_{L_2} + q'_{L_2})$$

$$\ldots$$

$$C'_{max} = max(C'_{max}, r'_{L_l} + p_{L_l} + q'_{L_l})$$

Este método es también apropiado para N_3. En este caso la subsecuencia $(L_1, ..., L_l)$ es dada por las primeras o últimas tareas del bloque crítico.

Test de Factibilidad. Para testar la factibilidad de una solución resultante de un movimiento en un bloque, se puede emplear un algoritmo de etiquetado para detectar ciclos en el grafo solución. Este procedimiento es exacto pero también computacionalmente caro. Por tanto, es mejor para testar la factibilidad una estimación eficiente como el algoritmo propuesto en [24] para N_2 dado por el siguiente lema.

Lema 2.1. *Para un movimiento de una tarea v dentro de un bloque $b = (b', b'', v, b''')$ que produce la secuencia (b', v, b'', b'''), puede existir un ciclo en el grafo de la solución resultante si y solo si existe una tarea w de b'' tal que hay un camino desde SJ_w a PJ_v.*

Por lo tanto, para comprobar la desigualdad es suficiente

$$r_{SJ_w} + p_{SJ_w} > r_{PJ_v}$$

para toda w de b'' que garantice que la planificación resultante es factible. De forma análoga, para un movimiento de v dentro del bloque $b = (b', v, b'', b''')$ que produce (b', b'', v, b''') es también suficiente testar la desigualdad

$$r_{SJ_v} + p_{SJ_v} > r_{PJ_w}$$

para toda w de b''.

En el caso de N_3 el chequeo de la factibilidad es incluso más simple. Debido a que la inversión de (v, w) siempre produce una planificación factible y al siguiente lema, que es probado en [24], es suficiente considerar como único vecino la permutación con menor estimación del makespan.

Lema 2.2. *La estimación de la inversión de (v, w) es menor o igual que cualquier estimación del makespan para el resultado de una inversión de dos arcos si tal inversión produce una solución no factible.*

2.4.4. Otros Métodos

Además de los métodos que hemos visto hasta ahora, existen otras técnicas aproximadas, o heurísticas, que aunque tampoco garantizan la solución óptima, son capaces de encontrar soluciones

para problemas JSS razonablemente buenas en poco tiempo. Entre estas técnicas cabe destacar las *Reglas de Prioridad* y el heurístico *Shifting Bottleneck*. Las reglas de prioridad permiten elegir el siguiente trabajo a ser planificado en el proceso de construir una planificación. Estas reglas son muy rápidas, pero la calidad de las soluciones que producen, normalmente, no es muy alta. El *Shifting Bottleneck* es un método mucho más elaborado, con un coste computacional mayor, que produce mejores aproximaciones. Se basa en optimizar repetidamente la secuencia de tareas de cada máquina individual, mientras que la secuencia de todas las máquinas no esté fijada y se produzcan mejoras. Veamos ahora con más detalle cada una de estas técnicas.

Las *Reglas de Prioridad* son probablemente las estrategias más aplicadas para resolver problemas de scheduling, debido a su fácil implementación y a su baja complejidad de tiempo. Una regla de prioridad asigna prioridades a todos los trabajos que están esperando ser procesados en una máquina. Cada vez que una máquina queda libre, una regla selecciona la siguiente tarea, aquella con mayor prioridad, a ser planificada. Las reglas de prioridad pueden ser empleadas para modificar el criterio de elección, en distintos algoritmos, a la hora de construir soluciones heurísticas, por ejemplo podríamos emplearlas en el algoritmo $G\&T$ de B. Giffler y G. L. Thomson [31] o en la construcción de una planificación con Prioridad de Jackson (JPS) [15]. Las reglas de prioridad se pueden clasificar en estáticas (no dependen del tiempo, sino de los trabajos y las máquinas) o dinámicas (depende del tiempo, por ejemplo *Mínima Holgura*). También se clasifican según la información que manejan. Por ejemplo las reglas locales sólo manejan información de la máquina, mientras que las reglas globales también manejan información relacionada con otras máquinas. Existen muchas reglas de prioridad (ver [50, 38]), entre ellas podemos destacar las siguientes: $SIRO$ (servicio en orden aleatorio), ERD (instante de inicio más temprano), EDD (instante de finalización más temprano), MS (mínima holgura), $WSPT$ (procesamiento ponderado más corto), LPT (tiempo de procesamiento más largo), SST (tiempo de setup más corto), LFJ (trabajo menos flexible), CP (camino crítico, mayor tiempo de procesamiento total), LNS (número mayor de sucesores), $SQNO$ (cola más corta en la siguiente tarea), $SRPT$ (tiempo remanente más corto), etc. Todas estas reglas de procesamiento básico son útiles para encontrar buenas soluciones a problemas con un único objetivo (makespan, flujo total o tardiness), sin embargo para problemas con objetivos más complejos son necesarias reglas de prioridad compuestas. Estas reglas combinan varias reglas básicas asignándoles un peso a cada una de ellas, ejemplos de estas reglas son la ATC (costo de retraso aparente) que trata de minimizar la suma de los retrasos ponderados de cada trabajo [65] y $COVERT$ (costo en el tiempo) que combina $WSPT$ y MS [93]).

El heurístico *Shifting Bottleneck* fue propuesto por Adams y sus colaboradores en [1], y consiste en planificar secuencias de máquinas de una en una. Para cada máquina aún no planificada, resuelve óptimamente el problema OMS resultante de la relajación del problema original (se relajan las restricciones de las máquinas cuyas tareas aún no han sido planificadas). Esta solución del problema OMS, calculada construyendo lo que se denomina una Planificación con Prioridad de Jackson [15], es empleada para clasificar las máquinas, siendo la máquina con la solución más costosa la que se denomina cuello de botella. Cada vez que se planifica la secuencia de una nueva máquina, el proceso reoptimiza la secuencia de las máquinas ya planificadas resolviendo de nuevo el problema de secuenciamiento de una máquina teniendo en cuenta los valores que proporciona la secuencia

de la nueva máquina planificada. Esta reoptimización se repite hasta que no se produce ninguna mejora. Una de las ventajas del heurístico *Shifting Bottleneck* es que se puede adaptar fácilmente a problemas con diferentes características y funciones objetivo.

2.5. Cálculo de Soluciones Exactas

En la literatura se encuentran diversos métodos para la resolución problemas de optimización de forma exacta, por ejemplo los métodos de programación dinámica y los de ramificación y poda. De entre todos ellos vamos a destacar el que sin duda es el mejor método exacto conocido para la resolución del problema JSS con minimización del makexpan, el algoritmo de ramificación y poda propuesto por P. Brucker y sus colaboradores en [12].

2.5.1. El Algoritmo de Ramificación y Poda de Brucker

Los algoritmos de ramificación y poda son posiblemente los métodos más empleados en la resolución exacta de problemas de optimización combinatoria. Tienen muchos elementos en común con otros algoritmos, como por ejemplo el algoritmo A*, pero tienen también algunas características propias. En primer lugar, con los otros algoritmos, la búsqueda se plantea como un proceso en el que la solución se construye paso a paso, por tanto el camino desde el estado inicial (estado en el que aún no se ha hecho nada) a otro estado representa parte de la solución del problema, representando el estado un subproblema que se debe resolver para llegar a una solución del problema original. Sin embargo, en un algoritmo de ramificación y poda cada estado se interpreta como un subconjunto de soluciones del problema original, en el estado inicial se tienen todas las soluciones y a partir de aquí, mediante el proceso de ramificación, un conjunto de soluciones se descompone en la unión disjunta de varios conjuntos, hasta llegar a conjuntos unitarios que representan soluciones del problema (este planteamiento hace que el espacio de búsqueda tenga estructura de árbol).

Las componentes fundamentales de los algoritmos de ramificación y poda son: un esquema de ramificación, un método de cálculo de cotas inferiores (Lower Bounds, LB), un método de cálculo de cotas superiores (Upper Bounds, UB) y una estrategia de control.

En la literatura existen multitud de propuestas de algoritmos de ramificación y poda para resolver problemas de scheduling, además del algoritmo propuesto por P. Brucker en ([11]) podemos citar el algoritmo propuesto por C. Artigues y D. Feillet en [3] para el problema JSS con tiempos de setup.

El algoritmo de ramificación y poda propuesto por P. Brucker en [12] es el mejor algoritmo exacto para la resolución del problema $J//C_{max}$. El algoritmo parte del grafo de restricciones del problema a resolver y va fijando arcos disyuntivos de forma sucesiva. Su principal característica es que emplea un elegante esquema de ramificación que permite fijar arcos en la misma u opuesta dirección a la que tienen en el bloque crítico de una solución factible. Además, el algoritmo utiliza potentes métodos para la obtención de buenas cotas inferiores y superiores. El cálculo de las cotas inferiores se basa en el problema se secuenciamiento de una máquina relajado. La solución óptima se este problema se obtiene de la planificación de Jackson (Jackson's Preemptive Schedule, JPS)

obtenida en tiempo polinomial por el algoritmo propuesto por J. Carlier en [15] y descrito en la sección 2.3.1. Las cotas superiores se obtienen empleando el algoritmo voraz $G\&T$ propuesto por B. Giffler y G. L. Thomson en [31] y que ha sido visto en la sección 2.4.1 de este capítulo. Por último el algoritmo de Brucker explota la propagación se restricciones debida a J. Carlier y E. Pinson [16] denominada *Selección Inmediata*. Este método permite fijar arcos disyuntivos adicionales con lo que se logra una drástica reducción del árbol de búsqueda. El algoritmo de Brucker resuelve fácilmente casi todas las instancias de tamaño 10×10 así como algunas instancias mayores. De hecho, el conjunto de instancias seleccionadas por D. Applegate y W. Cook en [2] son consideradas muy difíciles debido al hecho de que no son resueltas por este algoritmo (con la única excepción del $FT10$, que es famoso por otras razones).

Veamos ahora de forma más detallada cada uno de los componentes fundamentales del algoritmo de ramificación y poda propuesto por Brucker.

Esquema de Ramificación

La ramificación de un nodo es equivalente al cálculo de sucesores en los algoritmos de búsqueda en los que la solución se va construyendo poco a poco. Cada estado n es un grafo solución parcial $G_n = (V, A \cup FD_n)$, donde FD_n denota los arcos disyuntivos fijados en n. En el estado inicial $FD_n = \oslash$. La Figura 2.11 muestra un estado para el problema de la Figura 2.6 en el que hay dos arcos disyuntivos fijados, arco (θ_{21},θ_{11}) y arco (θ_{11},θ_{32}).

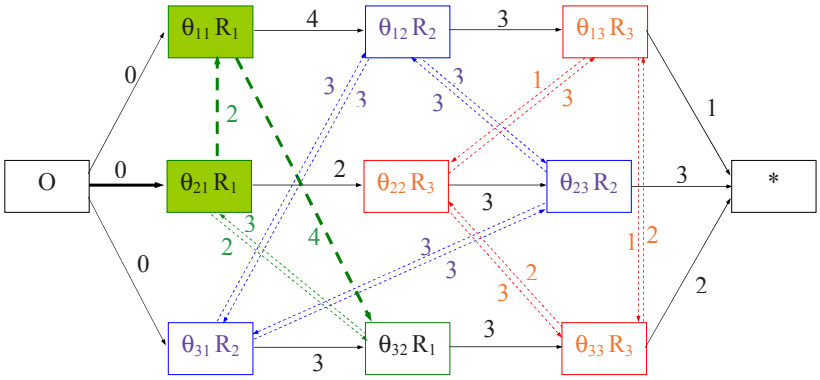


Figura 2.11: Estado intermedio de un problema JSS con dos arcos disyuntivos fijados

Las cabezas y colas del estado n se calculan a partir de los arcos disyuntivos fijados mediante las expresiones siguientes:

$$r_v = \text{máx}\{\max_{J \subseteq P(v)} \{\min_{j \in J} r_j + \sum_{j \in J} p_j\}, r_{PJ_v} + p_{PJ_v}\} \tag{2.1}$$

$$q_v = \text{máx}\{\max_{J \subseteq S(v)} \{\sum_{j \in J} p_j + \min_{j \in J} q_j\}, p_{SJ_v} + q_{SJ_v}\} \tag{2.2}$$

donde $P(v)$ denota los predecesores disyuntivos de la tarea v y $S(v)$ los sucesores disyuntivos. Y PJ_v y SJ_v representan al predecesor y sucesor conjuntivo de v respectivamente. El mecanismo

de ramificación se basa en el siguiente teorema ([12]), donde $L(S)$ denota el makespan de una planificación S.

Teorema 2.1. *Sean S y S' dos planificaciones. Si $L(S') < L(S)$, entonces se cumple una de las dos condiciones siguientes:*

1. *al menos una tarea v de un bloque crítico B en G_S, diferente de la primera tarea de B, es procesada en S' antes que todas las tareas de B.*
2. *al menos una tarea v de un bloque crítico B en G_S, diferente de la última tarea de B, es procesada en S' después que todas las tareas de B.*

Consideremos ahora una planificación S compatible con los arcos disyuntivos fijados en el estado n (por supuesto, S puede ser calculada por el algoritmo $G\&T$). El grafo solución G_S tiene un camino crítico con los bloques críticos $B_1, \ldots, B_k$. Para el bloque $B_j = (u_1^j, \ldots, u_{mj}^j)$ los conjuntos de tareas

$$E_j^B = B_j \setminus \{u_1^j\} \text{ y } E_j^A = B_j \setminus \{u_{mj}^j\}$$

son denominados *candidatos a ir antes* (*before-candidates*) y *candidatos a ir después* (*after-candidates*) respectivamente. Para cada before-candidate (after-candidate) se genera un sucesor s de n moviendo los candidatos antes (después) del correspondiente bloque. Una operación $l \in E_j^B$ se mueve antes del bloque B_j fijando los arcos $\{l \to i; i \in B_j \setminus \{l\}\}$. De forma similar, $l \in E_j^A$ se mueve tras el bloque B_j fijando los arcos $\{i \to l; i \in B_j \setminus \{l\}\}$.

Como consecuencia del teorema anterior, esta estrategia de expansión es completa por lo que garantiza que el espacio de la búsqueda contiene al menos una solución óptima. Sin embargo, esta estrategia puede ser mejorada fijando arcos adicionales que hagan que el espacio de búsqueda sea un árbol. Consideremos una permutación $(E_1, \ldots, E_{2k})$ de todos los conjuntos E_j^B y E_j^A. Esta permutación define un orden para la generación de sucesores. Cuando un sucesor se crea desde un candidato E_t, podemos tener en cuenta que todas las soluciones alcanzables desde n fijando los arcos correspondientes de los candidatos $E_1, \ldots, E_{t-1}$ serán exploradas desde los sucesores asociados a dichos candidatos. Por ello, para el estado sucesor s generado a partir de E_t se pueden fijar los siguientes arcos disyuntivos: $F_j = \{u_1^j \to i; i = u_2^j, \ldots, u_{mj}^j\}$, para cada $E_j^B < E_t$ y $L_j = \{i \to u_{mj}^j; i = u_1^j, \ldots, u_{mj-1}^j\}$, para cada $E_j^A < E_t$ en dicha permutación. Por tanto los sucesores de un nodo n en el árbol de búsqueda, generados a partir de la permutación $(E_1, \ldots, E_{2k})$ se definen del siguiente modo. Para cada tarea $l \in E_j^B$ se genera un nodo s del árbol de búsqueda fijando los arcos $FD_s = FD_n \cup S_j^B$, siempre que el grafo de la solución parcial no tenga ciclos, con

$$S_j^B = \bigcup_{E_i^B < E_j^B} F_i \cup \bigcup_{E_i^A < E_j^B} L_i \cup \{l \to i : i \in B_j \setminus \{l\}\}. \tag{2.3}$$

Y para cada tarea $l \in E_j^A$ se genera un nodo s del árbol de búsqueda fijando los arcos $FD_s = FD_n \cup S_j^A$, con

$$S_j^A = \bigcup_{E_i^B < E_j^A} F_i \cup \bigcup_{E_i^A < E_j^A} L_i \cup \{i \to l : i \in B_j \setminus \{l\}\}. \tag{2.4}$$

Cálculo de Cotas Inferiores

Como se ha visto en la sección 2.3 de este capítulo la obtención de buenas cotas inferiores es fundamental para la eficiencia de muchos algoritmos de búsqueda, y por tanto también para el algoritmo de ramificación y poda. En este algoritmo el cálculo de cotas inferiores se realiza de igual modo que en la sección 2.3. Se relajan las restricciones de capacidad en el problema JSS y se obtiene para cada máquina el problema de secuenciamiento de una máquina (OMS) (Ver sección 2.3.1). Para obtener el problema OMS de una máquina, se relajan las restricciones de capacidad del resto de máquinas. La solución óptima de este problema para un estado n es claramente una cota inferior de la solución óptima del problema JSS. Como ya hemos visto, el problema OMS es aún $NP-duro$, por lo que para poder resolverlo en tiempo polinomial es necesario realizar una segunda relajación, se relajan las restricciones de no expulsión. La solución óptima de este problema OMS relajado se obtiene mediante la Jackson's Preemptive Schedule (JPS) [16, 17], vista en la sección 2.3.1 de este capítulo. Recordemos cómo se calcula esta planificación: en cada instante t dado por la cabeza o el tiempo de fin de una tarea, desde la menor r_v hasta que todas las tareas están completamente planificadas, planificar la tarea que pueda empezar y que tenga mayor cola en la máquina m. Carlier y Pinson probaron que el cálculo de la JPS tiene una complejidad de $O(k \times \log k)$, donde k es el número de tareas. La cota inferior será el valor mayor de las JPS para todas las máquinas.

El algoritmo de Brucker, calcula cotas inferiores en tres lugares diferentes: durante el cálculo de los conjuntos E_i^B y E_j^A, durante el cálculo de las cabezas y colas, y tras el cálculo de las cabezas y colas (tras el cálculo de la JPS para cada máquina). Si tenemos en cuenta que el valor de la cota inferior LB puede incrementarse tras fijar arcos disyuntivos adicionales, es una buena política comprobar el valor de LB cada vez que se fije un arco disyuntivo. Si en todos o alguno de estos casos la cota LB es superior a la cota UB, no será necesario inspeccionar este nodo del árbol de búsqueda, con el consiguiente ahorro de tiempo.

Cálculo de Cotas Superiores

Las cotas superiores están asociadas a soluciones reales del problema completo. El algoritmo de ramificación y poda mantiene en una variable UB la menor de las cotas superiores obtenidas hasta el momento, y la solución correspondiente en la variable S. Por lo general, los algoritmos de ramificación y poda obtienen estas cotas cada vez que en la búsqueda se llega a un nodo hoja. Sin embargo se pueden calcular cotas superiores para cada nodo que se expande mediante un algoritmo heurístico eficiente, por ejemplo se utiliza un algoritmo voraz. Se trata de emplear un algoritmo que para un nodo n produzca una buena solución S.

El algoritmo de Brucker emplea el algoritmo $G\&T$ propuesto en [31] y explicado con detalle en la sección 2.4.1 de este capítulo. Cada vez que expande un estado n el algoritmo calcula una solución que incluye todos los arcos disyuntivos fijados hasta el momento en dicho estado n. El algoritmo $G\&T$ como se ha visto es un algoritmo voraz que produce una planificación en $N * M$ pasos. El Algoritmo 2.7 muestra la adaptación del algoritmo $G\&T$ para la obtención de cotas superiores para el estado n. Recordemos que $P(v)$ son las tareas predecesoras de la tarea v en el estado n. El

algoritmo considera, en cada iteración, el conjunto A formado por aquellas tareas v que pueden ser planificadas a continuación, es decir por aquellas tareas tales que $P(v)$ y PJ_v están planificadas (inicialmente sólo la tarea *inicio* está planificada). Se busca la tarea v^* perteneciente a A con menor tiempo de fin (en caso de ser planificada) y se construye el conjunto B con las tareas de A que requieren la misma máquina que v^* y que pueden comenzar antes que el tiempo de fin de v^*. Del conjunto B se elige la siguiente tarea a ser planificada, aquella tarea w^* que produce una JPS de menor coste para el resto de tareas sin planificar con las que comparte máquina (si hay empate, planifica la que pertenece al trabajo de menor identificador). Finalmente, el algoritmo retorna una planificación S y el valor se su makespan.

Propagación de Restricciones

El buen funcionamiento del algoritmo de Brucker se debe principalmente a la forma del espacio de búsqueda, y en parte al empleo de la propagación de restricciones *Selección Inmediata*, propuesta por Carlier y Pinson en [16] y cuya explicación detallada se puede ver en la sección 2.7.3 del anexo de este capítulo. La *Selección Inmediata* fija arcos disyuntivos adicionales entre tareas que requieren la misma máquina, para ello emplea una cota superior (UB) del makespan óptimo y sencillas cotas inferiores (LB).

La *Selección Inmediata* aplica tres test de consistencia, enunciados en forma de tres condiciones. Denotaremos por I el conjunto de tareas que requieren una máquina en el estado n. Y para cada tarea $j \in I$, r_j, q_j, d_j son la cabeza, cola y deadline respectivamente de la tarea j en el estado n.

Sea $J \subset I$ y $c \in I \backslash J$.

Condición 1. Si

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB, \tag{2.5}$$

entonces todas las tareas $j \in J$ deben ser procesadas antes que la tarea c si se desea mejorar el valor del actual UB.

Algoritmo 2.7 UB(estado n)

Requiere: Un estado n de un problema JSS
Produce: Solución del problema JSS;

1: $SC = \{inicio\}$; *{O conjunto de todas las tareas, SC conjunto de las tareas planificadas}*
2: **mientras** $(SC \neq O)$ **hacer**
3: $A = \{v \in O \setminus SC;\ P(v) \cup \{PJ_v\} \subset SC\}$;
4: $v^* = \arg\min\{r_u + p_u;\ u \in A\}$;
5: $B = \{v \in A;\ R_v = R_{v^*}$ y $r_v < r_{v^*} + p_{v^*}\}$;
6: $C = \{v \in O \setminus SC;\ R_v = R_{v^*}\}$;
7: $w^* = \arg\min\{$makespan de $JPS(C \setminus \{w\})$ tras planificar $w;\ w \in B\}$;
8: Planificar w^* en S en el instante r_{w^*};
9: Añadir w^* a SC y actualizar las cabezas y colas de las tareas no planificadas;
10: **fin mientras**
11: 8. Retornar S y su makespan;

El arco (J, c) se denomina un *par primal* si la condición 2.5 se cumple. Los arcos correspondientes $j \to c$, donde $j \in J$ se denominan *arcos primales*.

Condición 2. Si

$$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j > \max_{j \in J \cup \{c\}} d_j \tag{2.6}$$

entonces todas las tareas $j \in J$ deben ser procesadas tras la tarea c si se desea mejorar el valor del actual UB.

El arco (c, J) se denomina un *par dual* si la condición 2.6 se cumple. Los arcos $c \to j$, se denominan *arcos duales*.

Condición 3. Sea $J \subset I$, cuando J contiene una única tarea, entonces la condición 2.5 puede ser sustituida por una condición más débil:

$$r_c + p_c + p_j + q_j \geq UB \tag{2.7}$$

Cuando se cumple 2.7, se puede fijar la disyunción $j \to c$. El arco correspondiente se denomina *arco directo*.

La *Selección Inmediata* dispone del algoritmo $SELECT$ para fijar todos los arcos primales y duales con una complejidad $O(k^2)$, siendo k el número de tareas del conjunto I. Para aplicar este algoritmo previamente se ha de aplicar un método que permite mejorar las cabezas y las colas. Este método se basa en la idea de que si (J, c) es un par primal, la tarea c no puede comenzar antes que

$$r_J = \max_{J' \subseteq J} \{\min_{j \in J'} r_j + \sum_{j \in J'} p_j\}. \tag{2.8}$$

Por tanto, si $r_c < r_J$, podemos actualizar $r_c = r_J$. El cálculo de r_J se puede hacer también a partir de una planificación tipo JPS, en la que se incluye el par primal (J, c). Si en esta planificación $r_c < r_J$ entonces se aplica la mejora.

Análogamente, si (c, J) es un par dual, q_c no puede ser menor de

$$q_J = \max_{J' \subseteq J} \{\sum_{j \in J'} p_j + \min_{j \in J'} q_j\}. \tag{2.9}$$

Por tanto, si $q_c < q_J$, podemos actualizar $q_c = q_J$. Análogamente, para poder establecer esta mejora de la cola de c hemos de construir también una planificación tipo JPS que incluya el par dual (c, J).

En [16], Carlier y Pinson proponen un algoritmo, basado en el cálculo de planificaciones JPS, que construye planificaciones $BJPS$ (Backward Jackson's Preemptive Schedule), que permiten calcular la mejora de las cabezas y las colas con una complejidad de $O(klogk)$, con $k = |I|$. Para consultar los detalles del algoritmo o ver una traza detallada del mismo se puede consultar la sección 2.7.3 del anexo de este capítulo.

El algoritmo $SELECT$ que vemos a continuación, empleando las cabezas y colas mejoradas,

permite fijar todos los arcos primales con una complejidad $O(n^2)$. El algoritmo $SELECT$ para fijar todos los arcos duales sería análogo, se puede ver en el anexo, sección 2.7.3.

Algoritmo 2.8 PROCEDURE Select

Requiere: Conjunto de tareas I
Produce: Fija todos los arcos primales que puede para I
  for all $c, j \in I, c \neq j$ **hacer**
    si $r_c + p_c + p_j + q_j \geq UB$ **entonces**
      fix the arc $(j \rightarrow c)$
    fin si
  fin para

Finalmente, el algoritmo que permite fijar arcos disyuntivos adicionales quedaría así:

(1) Calcular todos los arcos primales para todas las máquinas,

(2) Calcular las nuevas cabezas y colas,

(3) Calcular todos los arcos duales para todas las máquinas

(4) Calcular las nuevas cabezas y colas.

Las nuevas cabezas y colas se calculan en los pasos 2 y 4, los arcos adicionales se fijan en los pasos 1 y 3. Estos pasos se repiten hasta que no se puedan fijar nuevos arcos adicionales.

Estrategia de Control

El Algoritmo 2.9, muestra el esquema del algoritmo de ramificación y poda de Brucker. Los algoritmos de ramificación y poda utilizan una estrategia de búsqueda en profundidad para la que es posible establecer un orden de generación de sucesores. El algoritmo de Brucker establece un orden para la elección de las tareas $i \in E_j^v$, es decir establece un orden en la generación de sucesores del árbol de búsqueda: ordena los candidatos en orden creciente de cabezas de los before-candidates y de colas de los after-candidates.

Por otro lado, el algoritmo mantiene la mejor cota superior calculada en todo momento y además calcula en diferentes instantes cotas inferiores. Esto le permite realizar podas del espacio de búsqueda, ya que si un sucesor s tiene una cota inferior $LB(s) \geq UB$, ya se conoce que en el conjunto $Y(s)$ no hay una solución mejor que la que ya se tiene en este momento, y en consecuencia el sucesor s no se tiene en cuenta. Cuanto mejores sean las estimaciones que se van haciendo mediante las cotas inferiores y superiores, más efectivas serán las podas realizadas.

El tratamiento de un nodo s termina si se da alguna de las siguientes condiciones: $LB(s) \geq UB$, el grafo disyuntivo contiene ciclos, o los conjuntos E_j^B y E_j^A están vacíos para todos los bloques B_j (es decir, que todos los B_j están formados por una única tarea).

Para asegurar la condición de admisibilidad, los algoritmos de ramificación y poda deben recorrer todo el espacio de búsqueda, salvo las partes que se podan, antes de terminar, ya que, en general, la primera solución que encuentran no es la óptima. No obstante, se pueden establecer criterios adicionales de parada, por ejemplo para instancias difíciles, después de un cierto número de nodos

expandidos, o después de un determinado tiempo de ejecución. Obviamente en este caso se pierde la admisibilidad, pero el algoritmo devolverá la mejor solución encontrada, con su coste UB, y la mejor cota inferior, LB, obtenida durante la búsqueda. Esta es una característica importante de este tipo de algoritmos: son admisibles si disponen de un tiempo de ejecución suficiente, pero producen una solución y una cota inferior si el tiempo es limitado.

Algoritmo 2.9 Branch_and_Bound (Problema)
Requiere: Problema JSS
Produce: Solución del problema JSS
1: Calcular una solución $S \in Y(n)$ mediante heurísticos;
2: **si** $(C_{max}(S) < UB)$ **entonces**
3: $UB = C_{max}(S)$;
4: **fin si**
5: Calcular un camino crítico P;
6: Calcular los bloques de P;
7: Calcular los conjuntos E_j^B y E_j^A con $j = 1, \ldots, k$;
8: **mientras** (exista una operación $i \in E_j^v$ con $j = 1, \ldots, k$ y $v = A, B$) **hacer**
9: Borrar i de E_j^v;
10: Fijar los arcos no dirigidos para el correspondiente sucesor s;
11: Fijar arcos adicionales para el sucesor s (Selección Inmediata);
12: Calcular una cota inferior $LB(s)$ para el nodo s;
13: **si** $(LB(s) < UB)$ **entonces**
14: Branch_and_Bound (s);
15: **fin si**
16: **fin mientras**

Conclusiones del Algoritmo. En este apartado hemos descrito el algoritmo de ramificación y poda propuesto por P. Brucker y sus colaboradores en [12]. Se trata de un complejo algoritmo que parte del grafo de restricciones del problema a resolver y va fijando arcos disyuntivos de forma sucesiva. Además, fija arcos disyuntivos adicionales empleando la propagación de restricciones *Selección Inmediata*, lo que consigue reducir considerablemente el espacio de búsqueda efectivo. Sin embargo, lo más destacado de este algoritmo es su elegante esquema de ramificación que permite fijar arcos en la misma u opuesta dirección a la que tienen en el bloque crítico de una solución factible. El espacio de búsqueda recorrido de este modo está contenido en el espacio de planificaciones *semiactivas* y contiene al menos una solución óptima. Todas estas características le han hecho convertirse en el mejor método exacto, conocido hasta el momento, para la resolución de problemas del problema $J||C_{max}$, aunque las ideas de este algoritmo no son fácilmente generalizables cuando la función objetivo es diferente del makespan.

2.6. Conclusiones

En este capítulo, tras una breve introducción a los problemas de Scheduling, nos hemos centrado en la formulación del problema JSS. Hemos visto la representación de una instancia del problema en

forma de grafo de restricciones y cómo se representa en este grafo una solución del mismo. Además, hemos identificado los diferentes espacios de soluciones factibles que se tienen para este problema.

Por otra parte, hemos realizado un recorrido por diversas técnicas que nos permiten obtener cotas inferiores (LB) y superiores (UB) de la solución óptima, así como soluciones óptimas, para el problema JSS. Para el calculo de cotas inferiores se ha empleado la relajación del problema JSS a problemas de secuenciamiento de una máquina (OMS). Para el cálculo de cotas superiores hemos visto distintos métodos, concretamente el Algoritmo $G\&T$, los Algoritmos Evolutivos y la Búsqueda Local, así como otros métodos heurísticos como el *Shifting Bottleneck* y las Reglas de Prioridad. Además, hemos descrito el mejor método exacto conocido hasta el momento para la resolución del problema JSS con minimización del makespan, el algoritmo de ramificación y poda propuesto por Brucker en [12]. Este método combina varias técnicas (un buen esquema de ramificación, cálculo de buenas cotas LB y UB y propagación de restricciones *Selección Inmediata*) que le hacen tener el mejor comportamiento para la resolución del problema JSS, llegando a resolver todas las instancias hasta un tamaño de 10×10, e incluso algunas mayores.

2.7. Anexo. Propagación de Restricciones

2.7.1. Introducción

Uno de los factores importantes a la hora de resolver problemas de Scheduling es el empleo de estrategias que permitan reducir el espacio de búsqueda. Para ello es común utilizar estrategias basadas en la propagación de restricciones.

La propagación de restricciones es un método potente que permite la reducción del espacio de búsqueda a través del análisis y evaluación de las variables, de sus dominios y de la relación entre ellas. Así pues, el objetivo perseguido por la propagación de restricciones es detectar y eliminar las asignaciones de variables inconsistentes, es decir, aquellas que no pueden formar parte de una solución, o bien que no pueden formar parte de soluciones mejores que la que tenemos en un momento dado. Hoy en día se puede encontrar en la literatura una completa teoría que define los diferentes tipos de consistencia, lo que proporciona un respaldo a las técnicas de propagación de restricciones, en este sentido son relevantes los trabajos de Kumar [48], Tsang [88], Dorndorf [25, 26] o Phan-Huy [63].

La eliminación de todas las asignaciones inconsistentes provoca que la complejidad computacional se incremente exponencialmente, esto hace que se tengan en cuenta únicamente aproximaciones. Por tanto, una importante tarea será describir estas aproximaciones, reglas o condiciones simples que permitan realizar reducciones eficientes del espacio de búsqueda y que al mismo tiempo puedan ser implementadas de forma eficiente. Estas reglas son denominadas test de consistencia.

Nos centraremos en el estudio e implementación de la propagación de restricciones disyuntivas utilizada por Brucker en [12] para el problema $J||C_{max}$ y denominada *Selección Inmediata*. Este método trata de fijar las restricciones disyuntivas que la solución del problema JSS debe mantener. Además de esta propagación de restricciones disyuntivas se aplicarán tests relacionados con la propagación de las restricciones secuenciales.

2.7.2. Test de Consistencia

Los *test de consistencia* se describen generalmente a través de reglas de condición y reducción del dominio; es decir, cuando la condición se satisface, la reducción del dominio puede ser aplicada.

Para que los tests de consistencia puedan obtener la máxima reducción posible del domino han de ser aplicados más de una vez. La razón de esto es que después de reducir algunos dominios la aplicación de otros tests que previamente fallaron, es decir, que no consiguieron ninguna reducción, pueden producir nuevas reducciones. Por ello el proceso de reducción de los dominios debe repetirse hasta que no se logre ninguna mejora, es decir, hasta que no sea posible realizar una nueva reducción del dominio. En general las reducciones conseguidas dependen del orden en que sean aplicados los test, sin embargo cuando los test empleados satisfacen la condición de monotonía el orden de aplicación no influye a la hora de conseguir la máxima reducción posible.

Los tests de consistencia que empleamos tienen relación con la definición del problema $J||C_{max}$ y se pueden distinguir dos grupos: los *tests de consistencia del dominio* y los *tests de consistencia de la secuencia*.

Los *tests de consistencia del dominio* se basan en la formulación del problema JSS orientada al tiempo: una solución del JSS es el conjunto de tiempos de inicio de las tareas, cada uno escogido de su correspondiente dominio. Así pues los tests de consistencia del dominio permiten reducir los posibles tiempos de inicio de las tareas reduciendo sus correspondientes dominios. Los dominios de los tiempos de inicio de una tarea vienen determinados por las tareas con las que comparte restricciones conjuntivas (restricciones secuenciales) y disyuntivas (restricciones de capacidad).

Los *test de consistencia de la secuencia* se basan en la formulación del problema JSS orientada a la secuencia: una solución del JSS es la elección del orden en que van a ser procesadas las tareas que emplean el mismo recurso o máquina, es decir, la selección de la orientación de los arcos disyuntivos que unen a las tareas que emplean el mismo recurso. Así pues los tests de consistencia de la secuencia reducen el conjunto de posibles elecciones de las orientaciones de los arcos y por tanto detectan el orden de la secuencia de tareas. Los trabajos de Carlier y Pinson [16] y Brucker [12] ofrecen una aproximación de estos tests denominada *Selección Inmediata*, pero existen más aproximaciones como por ejemplo la denominada *edge-finding* propuesta por Applegate y Cook en [2].

Los *tests de consistencia del dominio* y *de la secuencia* son dos conceptos diferentes que se complementan entre sí. A menudo puede ocurrir que sólo sea posible reducir el dominio o que sólo sea posible deducir orientaciones de arcos disyuntivos. Los mejores resultados se obtienen, por tanto, aplicando ambos tipos de tests de consistencia, ya que al fijar la orientación de los arcos disyuntivos se puede producir una reducción del dominio y viceversa. Esto es lo que hace la *Selección Inmediata*.

En [25, 26] se enumeran una serie de test de consistencia que vamos a pasar a resumir. Para describir los test de consistencia emplearemos la notación que se describe a continuación. Δ_i denota al dominio de los posibles tiempos de inicio para la tarea i (del dominio eventualemente se borrarán los tiempos de inicio inconsistentes), evidentemente al principio $\Delta_i = \mathbb{N}_0$. Muchos de los test de consistencia requieren el empleo de una cota superior de la solución UB, por lo tanto podemos establecer $\Delta_i = \{0, 1, \ldots, UB\}$. $ES_i = \min \Delta_i$ y $LS_i = \max \Delta_i$ denotan respectivamente a los tiempos de inicio más temprano y tardío para la tarea i. Por tanto los tiempos de completud más temprano y más tardío para una tarea i se definen respectivamente como $EC_i = ES_i + p_i$ y $LC_i = LS_i + p_i$, donde p_i es la duración de la tarea i. Finalmente, para un subconjunto de tareas A de las que requieren la misma máquina O_c, podemos establecer $ES(A) = \min_{i \in A} ES_i$, $LC(A) = \max_{i \in A} LC_i$ y $P(A) = \sum_{i \in A} p_i$.

Test de Consistencia de Precedencia. Si tenemos fijado el arco (i, j)

(C01) $LC_i = \min\{LC_i, LS_j - p_i\}$,

(C02) $ES_j = \max\{ES_j, EC_i\}$.

Test de Consistencia de Secuencia: Entrada/Salida. Si tenemos un conjunto de operaciones A y una tarea $i \notin A$, estos test determinan si esta tarea ha de ser procesada antes o después de las tareas de A.

(C03) $LC(A \cup \{i\}) - ES(A) < P(A \cup \{i\}) \Rightarrow i \to A$,

(C04) $LC(A) - ES(A \cup \{i\}) < P(A \cup \{i\}) \Rightarrow A \rightarrow i$,

(C05) $\text{máx}_{u \in A, v \in A \cup \{i\}, u \neq v}\{LC_v - ES_u\} < P(A \cup \{i\}) \Rightarrow i \rightarrow A$,

(C06) $\text{máx}_{u \in A \cup \{i\}, v \in A, u \neq v}\{LC_v - ES_u\} < P(A \cup \{i\}) \Rightarrow A \rightarrow i$.

Test de Consistencia de Secuencia: Entrada o Salida. Si tenemos un conjunto de operaciones A y dos tareas $i, j \notin A$, estos test determinan si la tarea i ha de ser procesada la primera o la tarea j ha de ser procesada la última con respecto a $A \cup \{i, j\}$. Por lo tanto, si $i \neq j$, determinan si i ha de ser procesada antes que j.

(C07) $LC(A \cup \{i\}) - ES(A \cup \{j\}) < P(A \cup \{i, j\}) \Rightarrow i \rightarrow j$,

(C08) $\text{máx}_{u \in A \cup \{j\}, v \in A \cup \{i\}, u \neq v}\{LC_v - ES_u\} < P(A \cup \{i, j\}) \Rightarrow i \rightarrow j$.

Test de Consistencia de Dominio: Entrada/Salida. Si tenemos un conjunto de operaciones A y una tarea $i \notin A$, si se deduce que $A \rightarrow i$ o $i \rightarrow A$, estos tests determinan el tiempo de inicio más temprano o tardío, respectivamente, para la tarea i. Para enunciar estos test, es necesario disponer de una cota inferior del makespan resultante de planificar las tareas de A. En caso de que tengamos $A \rightarrow i$, esta cota se obtiene mediante una planificación JPS de las tareas de A y este valor se denota por $C^{pr}_{max}(A)$.

(C09) Si se deduce que $A \rightarrow i \Rightarrow ES_i = \text{máx}\{ES_i, C^{pr}_{max}(A)\}$

Análogamente, si tenemos la situación $i \rightarrow A$ la cota inferior del makespan se puede obtener a partir de una cota superior UB y de una Backward JPS ($BJPS$) obtenida a partir de UB para las tareas de A. El cálculo de la $BJPS$ se describe en el Anexo 2.7.3.

(C10) Si se deduce que $i \rightarrow A \Rightarrow LS_i = \text{mín}\{LS_i, UB - C^{pr}_{max}(A)\}$.

Test de Consistencia de Negación: Entrada/Salida. Si tenemos un conjunto de operaciones A y una tareas $i \notin A$, estos test determinan cuando la tarea i no puede ser procesada antes (después) con respecto a $A \cup i$

(C11) $LC(A) - ES_i < P(A \cup \{i\}) \Rightarrow ES_i = \text{máx}\{ES_i, \text{mín}_{u \in A}\{EC_u\}\}$,

(C12) $LC_i - ES(A) < P(A \cup \{i\}) \Rightarrow LS_i = \text{mín}\{LS_i, \text{máx}_{u \in A}\{LS_u\} - p_i\}$.

La Tabla 2.1, resume las complejidades de estas propagaciones de restricciones. Para tener más detalles sobre las mismas se pueden consultar las referencias incluidas en [25, 26].

2.7.3. Selección Inmediata

La *Selección Inmediata* agrupa una serie de tests o condiciones de consistencia que permiten fijar arcos disyuntivos y por lo tanto mejorar los valores de las cabezas y las colas.

El número de disyunciones fijadas por estos métodos de propagación tiene una importante influencia tanto en la calidad de los algoritmos de búsqueda, como en el cálculo de cotas de la solución óptima, ya que:

Tabla 2.1: Complejidades de los Test de Consistencia

Número de Test	Tipo del Test	Complejidad
(C01), (C02)	Test de Consistencia de Precendencia	$O(num.tareas)$
(C03), (C04)	Test de Consistencia de Secuencia: Entrada/Salida	$O(\|O_c\| \log \|O_c\|)$
(C05), (C06)	Test de Consistencia de Secuencia: Entrada/Salida	$O(\|O_c\|^3)$
(C07)	Test de Consistencia de Secuencia: Entrada o Salida	$O(\|O_c\|^3)$
(C08)	Test de Consistencia de Secuencia: Entrada o Salida	$O(\|O_c\|^4)$
(C09), (C10)	Test de Consistencia de Dominio: Entrada/Salida	$O(\|O_c\|^2)$
(C11), (C12)	Test de Consistencia de Negación: Entrada/Salida	$O(\|O_c\|^2 \log \|O_c\|)$

- Generalmente, el valor de las cotas inferiores se incrementa al tener más arcos disyuntivos fijados.
- Si se tiene la información adicional de que la tarea j sucede a la tarea i en cualquier solución alcanzable desde la situación actual que pueda mejorar a la cota superior actual, entonces el algoritmo empleado para el recorrido del espacio de búsqueda no tendrá en cuenta planificaciones donde j se procese antes que i. Por tanto, ese algoritmo normalmente calculará mejores soluciones.

Por lo tanto estas condiciones de consistencia pueden ser empleados para el cálculo de buenas cotas de la solución óptima, así como para reducir el espacio de búsqueda recorrido por las estrategias de búsqueda.

El método de *Selección Inmediata* repetirá sus tests de consistencia hasta que no se produzca ninguna fijación de arcos disyuntivos ni mejora de cabezas y colas, es decir, hasta que la propagación no sea capaz de producir nuevas mejoras. La *Selección Inmediata* es un método de propagación de restricciones cuyos test de consistencia se basan en los test, enunciados en el apartado anterior, $C01$, $C02$, $C03$, $C04$, $C09$ y $C10$.

Tests de Consistencia de la Selección Inmediata

La *Selección Inmediata* fija arcos disyuntivos adicionales entre tareas que serán procesadas en la misma máquina. Para ello este método emplea una cota superior (UB) del makespan óptimo, así como cotas inferiores (LB) sencillas. Sea I el conjunto de todas las tareas que deben ser procesadas en una máquina dada y n el número de elementos de I.

Sea $J \subset I$ y $c \in I \backslash J$.

Condición 1. Si se cumple la condición:

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB, \tag{2.10}$$

entonces todas las tareas $j \in J$ deben ser procesadas antes que la tarea c si se desea mejorar el valor del actual UB. Esto se basa en el hecho de que la parte izquierda de 2.10 es una cota inferior para todas las planificaciones en las que c no va detrás de todas las tareas de J.

La condición 2.10 es equivalente a:

$$\min_{j\in J\cup\{c\}} r_j + \sum_{j\in J\cup\{c\}} p_j + \min_{j\in J} q_j > UB - 1 \tag{2.11}$$

o

$$\min_{j\in J\cup\{c\}} r_j + \sum_{j\in J\cup\{c\}} p_j > \max_{j\in J} d_j \tag{2.12}$$

Donde $d_j = UB - q_j - 1$.

(J, c) se denomina un *par primal* si la condición 2.10 o sus equivalentes (2.11 y 2.12) se cumplen. Los arcos correspondientes $j \to c$, donde $j \in J$ se denominan *arcos primales*.

Condición 2. Sea $J \subset I$ y $c \in I\backslash J$. Si se cumple la condición:

$$\min_{j\in J} r_j + \sum_{j\in J\cup\{c\}} p_j > \max_{j\in J\cup\{c\}} d_j \tag{2.13}$$

Entonces todas las tareas $j \in J$ deben ser procesadas tras la tarea c si se desea mejorar el valor del actual UB. Esto se basa en el hecho de que la parte izquierda de 2.13 es una cota inferior para todas las planificaciones en las que c va detrás de todos los tareas de J.

(c, J) se denomina un *par dual* si la condición 2.13 se cumple. Los arcos $c \to j$, se denominan *arcos duales*.

Condición 3. Sea $J \subset I$, cuando J contiene una única tarea, entonces la condición 2.10 puede ser sustituida por una condición más débil:

$$r_c + p_c + p_j + q_j \geq UB \tag{2.14}$$

que equivale a

$$p_c + p_j > d_j - r_c \tag{2.15}$$

Cuando se cumplen estas condiciones (2.14 y 2.15), se puede fijar la disyunción $j \to c$. El arco correspondiente se denomina *arco directo*.

En las condiciones 2.12, 2.13 y 2.15 los valores d_j pueden ser interpretados como plazos de fin o deadlines.

En el apartado 2.7.3 se presenta un algoritmo que permite fijar todos los arcos primales y duales con una complejidad $O(n^2)$. Este algoritmo se basa en el método, descrito en el siguiente apartado, que permite mejorar las cabezas y las colas.

Mejora de Cabezas y Colas en la Selección Inmediata

En este apartado describimos un método que permite mejorar las cabezas y las colas en la solución parcial que representa un estado, basándose en la construcción de una planificación factible. Los valores de las cabezas y las colas, son importantes a la hora de calcular cotas inferiores. Recordemos

que:

- La cabeza r_i de la tarea i, es una cota inferior del tiempo de inicio más temprano posible para la tarea i.
- La cola q_i de la tarea i, es una cota inferior del tiempo comprendido entre el tiempo de finalización de la tarea i y el makespan de la mejor solución conocida (UB).

En el cálculo de las cabezas y colas influyen todos los arcos conjuntivos y disyuntivos fijados. Por lo que sus valores dependen del estado del problema para el que se calculen. Como se ha indicado en capítulos anteriores, una forma simple de obtener la cabeza r_i de la tarea i sería calcular la longitud del camino de mayor coste desde el nodo *inicio* al nodo i en el grafo disyuntivo $G = (V, A \cup FD_n)$ que representa el estado del problema JSS. De manera similar, para cada tarea i la cola q_i podría ser calculada como la longitud del camino de mayor coste desde el nodo i al nodo *fin* en el grafo disyuntivo $G = (V, A \cup FD_n)$ que representa el estado del problema $J||C_{max}$.

En el estado inicial, los valores de las cabezas y colas así calculados coinciden con las calculados propagando únicamente las restricciones secuenciales, ya que en este estado las únicas restricciones fijadas son las secuenciales. Sin embargo a la hora de realizar los cálculos de las cabezas y colas en el resto de estados, se empleará un método más sofisticado. Para ello utilizaremos la siguiente notación:

- PJ_i y SJ_i denotan, respectivamente, al predecesor y sucesor conjuntivo de la tarea i (predecesor y sucesor en su trabajo).
- $P(i)$ denota a los predecesores disyuntivos de i, es decir aquellas operaciones que requieren la misma máquina que i (R_i) y están planificadas antes que i. Del mismo modo, $S(i)$ denota a los sucesores disyuntivos de i.

Así, la cabeza de la tarea i se puede calcular en principio como:

$$r_i = \text{máx}\{\max_{w \in P(i)}(r_w + p_w), r_{PJ_i} + p_{PJ_i}\} \tag{2.16}$$

donde $r_{PJ_i} + p_{PJ_i}$ es una cota inferior que viene del predecesor conjuntivo PJ_i, valor que se obtiene de la propagación de la restricción secuencial previa para la tarea i; y $\text{máx}_{w \in P(i)}(r_w + p_w)$ es una cota inferior que viene del conjunto de predecesores disyuntivos $P(i)$, valor que se obtiene de la propagación de las restricciones disyuntivas previas a la tarea i.

La misma idea conduce a la definición de cotas inferiores para el tiempo de procesamiento posterior al procesamiento de la tarea i (cola):

$$q_i = \text{máx}\{\max_{w \in S(i)}(p_w + q_w), p_{SJ_i} + q_{SJ_i}\} \tag{2.17}$$

Así, las cabezas y colas se calculan de forma recurrente teniendo en cuenta que $r_0 = 0$ y $q_{fin} = 0$.

Empleando estas cotas para el cálculo de las cabezas y colas, es posible definir recursivamente tanto el calculo de la cabeza r_i como el de la cola q_i para una tarea i:

$$\begin{aligned} r_0 = 0; \quad r_i = \text{máx}\{\text{máx}_{w \in P(i)}(r_w + p_w), r_{PJ_i} + p_{PJ_i}\} \\ q_0 = 0; \quad q_i = \text{máx}\{\text{máx}_{w \in S(i)}(p_w + q_w), p_{SJ_i} + q_{SJ_i}\} \end{aligned} \tag{2.18}$$

Así pues, podríamos partir de las cabezas y colas nulas y luego calcular recursivamente los valores de las cabezas y colas. Sin embargo es más práctico partir de unas cabezas y colas inicializadas a los valores obtenidos de la propagación de restricciones secuenciales en el estado inicial.

Algoritmo de Mejora de Cabezas y Colas. El algoritmo para la mejora de cabezas y colas se basa en la construcción de una planificación con expulsión factible para las n tareas del conjunto I y de una cota superior UB. Las tareas del conjunto I han de ser planificadas en la misma máquina y tienen asociados unos tiempos de procesamiento p_k, unas cabezas r_k y unos deadlines d_k (o colas q_k). Se trata por lo tanto de construir una planificación para un problema de secuenciamiento de una máquina, problema OMS, formado por las tareas del conjunto I. Recordemos que el correspondiente problema OMS relajado (que permite la expulsión), puede ser resuelto óptimamente construyendo una planificación JPS (*Jackson's Preemptive Schedule*), siendo el makespan de esta solución una buena cota inferior de la solución del problema OMS sin relajar. Esto ha sido explicado con mayor detalle en la sección anterior 2.3.1.

La planificación JPS se calcula de izquierda a derecha (de menor a mayor cabeza) aplicando la siguiente regla: en cada instante de tiempo t, que viene dado por una cabeza o por el tiempo de fin de una tarea, se planifica la tarea no finalizada i con $r_i < t$ y $d_i = \arg\text{mín}\{d_j | r_j < t$, con j no planificada $\}$. En la sección 2.3.1 se explica en profundidad el método de construcción de esta planificación JPS empleando colas (q_i) en lugar de deadlines (d_i). Existe una versión dual del JPS que calcula la planificación de derecha a izquierda (de mayor a menor deadline) aplicando la regla dual: en cada instante t, que viene dado por un deadline o por el tiempo de inicio de una tarea, planificar hacia atrás la tarea i con $d_i \geq t$ y $r_i = \text{máx}\{r_j | d_j \geq t$, con j no planificada$\}$. Las planificaciones de este tipo se denominan Backwards Jackson's Preemptive Schedule ($BJPS$). No resulta difícil ver que una planificación JPS es factible si y sólo si $BJPS$ es factible.

En este caso el objetivo perseguido con la construcción de la planificación no es resolver el problema OMS relajado, sino mejorar la cabeza r_c de una tarea $c \in I$ en el problema original JSS sin expulsión. La idea es calcular una cota inferior S_c para el tiempo de fin de la tarea c bajo el supuesto de que las cabezas y deadlines son respetados, pero permitiendo la expulsión.

Se denomina S_c al tiempo de fin más temprano posible para la tarea c. Claramente, S_c es también una cota inferior para el tiempo de fin de la tarea c con respecto a las planificaciones sin expulsión. Si $S_c - p_c > r_c$ entonces r_c puede ser mejorada a $r_{c'} = S_c - p_c$.

El método empleado para el cálculo de S_c se denomina algoritmo de mejora de cabezas y se puede ver a continuación, donde $d_v \leq d_{n-1} \ldots \leq d_1 = d$ y $r = \text{mín}_{j \in I}\, r_j$.

En relación con la corrección del algoritmo de mejora de cabezas existe el siguiente teorema [11]:

Algoritmo 2.10 Algoritmo de Mejora de Cabeza (c)

Requiere: Tarea c

Produce: Cabeza de c mejorada

1. Calcular JPS hasta r_c;
2. Calcular $BJPS$ sin c en $[r_c, d]$ empleando los tiempos de procesamiento restantes p_k^+.
3. Planificar la tarea c de izquierda a derecha utilizando los periodos de inactividad más tempranos en $[r_c, d]$. S_c será el tiempo de finalización de la tarea c
4. Si $S_c - p_c > r_c$ entonces $r_c = S_c - p_c$

Teorema 2.2. *La planificación S calculada por el algoritmo de mejora de cabezas es factible. El algoritmo de mejora de las cabezas proporciona una cota inferior para el tiempo de fin de la tarea c bajo el supuesto de que la expulsión de tareas está permitida.*

La Figura 2.12.a, muestra la planificación JPS para el conjunto de tareas I; la Figura 2.12.b muestra los resultados de los tres pasos del algoritmo de mejora de cabezas, aplicados a la tarea $c = 3$ y la Figura 2.12.c la planificación resultante tras simplificar el cálculo de S_c (simplificación que se explica más adelante).

El cálculo del tiempo más temprano de fin S_c puede ser simplificado teniendo en cuenta la consideración de que para aplicar el paso 3 del algoritmo, sólo es necesario conocer los periodos de inactividad generados por el $BJPS$. Debido a esto se puede simplificar el paso 2 realizando el cálculo de periodos de inactividad del siguiente modo:

- Comenzando por el instante d, planificar hacia atrás y sin expulsión, en orden decreciente de deadlines, la tarea $j \in I \backslash c$ con un tiempo de proceso p_j^+.

Se puede ver que este algoritmo genera los mismos periodos de inactividad que el algoritmo $BJPS$.

Los deadlines pueden ser mejorados de una forma similar. Debido a la simetría existente entre las cabezas y colas, es posible realizar la mejora de las colas, aplicando el mismo algoritmo de mejora de cabezas, tras intercambiar en el problema los valores de las cabezas por los deadlines y viceversa. Cuando se calcula la mejora de los deadlines se está calculando la mejora de las colas pues el paso de unos a otros es inmediato: $d_j = UB - q_j - 1$ y $q_j = UB - d_j - 1$.

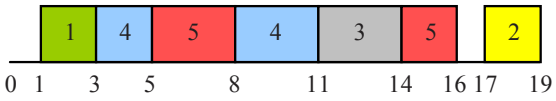
Fijando Arcos Primales y Duales. En este apartado se muestra el algoritmo que permite fijar todos los arcos primales y duales. Se comenzará explicando el procedimiento $SELECT$, que permite fijar todos los arcos directos.

El procedimiento $SELECT$ fija todos los arcos directos asociados al conjunto I de n tareas con una complejidad $O(n^2)$. El Algoritmo 2.11 muestra el procedimiento $SELECT$ empleado para fijar los arcos directos.

Cuando se sustituyen los valores r_i y d_i por las mejoras calculadas para ellos r'_i y d'_i ($r'_i \geq r_i$ y $d'_i \leq d_i$ para $i = 1, \ldots, n$), el algoritmo $SELECT$ es capaz de calcular todos los arcos primales y duales asociados al conjunto de tareas I. La prueba de este resultado viene dado en el siguiente teorema, enunciado y demostrado por Brucker en [11],

i	1	2	**3**	4	5
r_i	1	17	**7**	3	5
p_i	4	2	**3**	5	3
d_i	20	19	**15**	12	10

Problema OMS Primal para el conjunto de tareas I



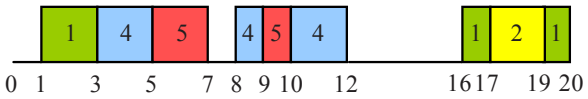
a) Planificación *JPS*

Paso 1:

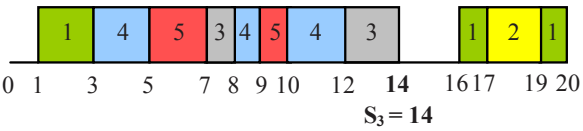
$r_3 = 7$

$p_1^+ = 2, p_2^+ = 2, p_4^+ = 3, p_5^+ = 1$

Paso 2:

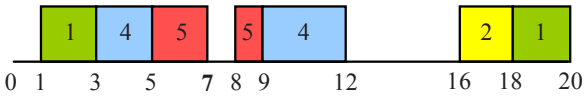


Paso 3:



$S_3 = 14$

b) Pasos del Algoritmo de Mejora de Cabezas para la tarea c=3



c) Planificación resultante tras simplificar el cálculo de S_c

Figura 2.12: Cálculo del tiempo más tempranos de completud para la tarea $c = 3$

Algoritmo 2.11 SELECT. Algoritmo que fija arcos Directos

Requiere: Conjunto de tareas I

Produce: Fija todos los arcos directos que puede para I

para todos $c, j \in I$ **hacer**

Si $p_c + p_j > d_j - r_c$ **entonces** fijar el Arco Directo $j \to c$

fin para

Teorema 2.3. *Sea I un conjunto de n tareas, que van a ser procesadas sobre una máquina, con cabezas y deadlines r_i, d_i. Sean r'_i y d'_i para $i = 1, \ldots, n$, las mejoras calculadas por los algoritmos de mejora de cabezas y colas (Algoritmo 2.10), entonces:*

a) Para cada par primal (J, c) con $J \subset I$, $c \in I \setminus J$ se tiene que $r'_c + p_c + p_j > d_j \geq d'_j$, para todo $j \in J$.
Esto significa que empleando el procedimiento SELECT con las cabezas modificadas r'_i ($i = 1, \ldots, n$), se pueden calcular todos los arcos disyuntivos asociados con los arcos Primales.

b) Para cada par dual (c, J) con $J \subset I$, $c \in I \setminus J$ se tiene que $r'_j + p_j + p_c \geq r_j + p_j + p_c > d'_c$, para todo $j \in J$.
Esto significa que empleando el procedimiento SELECT con los deadlines modificados d'_i ($i = 1, \ldots, n$), se pueden calcular todos los arcos disyuntivos asociados con los arcos Duales.

Basándonos en este teorema podemos decir que los algoritmos que mostramos a continuación permiten fijar todos los arcos primales y duales.

Algoritmo 2.12 SELECT Primal. Algoritmo que fija todos los arcos Primales
Requiere: Conjunto de tareas I
Produce: Fija todos los arcos primales que puede para I
  para todos $c, j \in I$, $c \neq j$ **hacer**
    Si $p_c + p_j > d'_j - r'_c$ **entonces** fijar el Arco Primal $j \to c$
  fin para

Algoritmo 2.13 SELECT Dual. Algoritmo que fija todos los arcos Duales
Requiere: Conjunto de tareas I
Produce: Fija todos los arcos duales que puede para I
  para todos $c, j \in I$, $c \neq j$ **hacer**
    Si $p_j + p_c > d'_c - r'_j$ **entonces** fijar el Arco Dual $c \to j$
  fin para

Así, combinando la mejora de cabezas y colas con el procedimiento *SELECT* se obtiene un procedimiento de complejidad $O(n^2)$ que permite fijar todos los arcos primales y duales para un conjunto I de n tareas que van a ser procesadas en la misma máquina.

Aplicación de la Selección Inmediata

Para que los tests de consistencia que emplea la Selección Inmediata puedan fijar el máximo número de arcos primales y duales, calculando las mejores cabezas y colas, han de ser aplicados más de una vez. Esto es así porque la fijación de un arco o el cálculo de un mejor valor para una cabeza o una cola puede provocar que se fijen otros arcos o se mejoren los valores de otras cabezas o colas.

La Selección Inmediata es un proceso que se repetirá hasta que en una iteración no se produzca ninguna mejora, es decir, hasta que en una iteración no se fije ningún nuevo arco primal o dual, ni se mejore el valor de ninguna cabeza o cola. El algoritmo 2.14 muestra el esquema de la *Selección Inmediata*. En los pasos (1) y (3) mejoramos las cabezas y las colas, respectivamente, y aplicamos

el algoritmo *SELECT*. En los pasos (2) y (4) se calculan las nuevas cabezas y colas, ya que estos valores pueden cambiar debido a los arcos fijados en los pasos anteriores. Todos estos pasos se repiten hasta que no se puedan fijar nuevos arcos.

Algoritmo 2.14 Selección Inmediata

Requiere: Estado n del problema JSS
Produce: Fija todos los arcos adicionales que puede, para el estado n
  mientras Haya mejora **hacer**
    (1)Calcular todos los arcos primales para todas las máquinas;
    (2)Calcular las nuevas cabezas y colas;
    (3)Calcular todos los arcos duales para todas las máquinas;
    (4)Calcular las nuevas cabezas y colas;
  fin mientras

Las cabezas y colas que recibe la *Selección Inmediata* pueden estar inicializadas con los valores calculados propagando las restricciones secuenciales. Cuando esto es así, hemos comprobado experimentalmente que, se reduce el número de iteraciones de la *Selección Inmediata*.

Capítulo 3

BÚSQUEDA HEURÍSTICA EN ESPACIOS DE ESTADOS

3.1. Introducción

La búsqueda heurística en espacios de estados es una de las técnicas clásicas de la Inteligencia Artificial. Es una técnica que permite resolver problemas de optimización combinatoria incorporando conocimiento específico del dominio problema. En este capítulo revisaremos los principales conceptos y algoritmos relativos a esta estrategia. En los capítulos siguientes veremos cómo aplicar uno de los algoritmos más conocidos, el algoritmo A^* propuesto por Nilsson [55, 57, 58], al problema Job Shop Scheduling.

A la hora de diseñar un sistema de búsqueda en espacios de estados para resolver un determinado problema debemos considerar los siguientes componentes: los estados, las reglas u operadores y la estrategia de control. Los estados representan las sucesivas situaciones, o problemas residuales, por los que se pasa durante la resolución de un problema. Las reglas u operadores permiten pasar de un estado a otro, es decir obtener a partir de un estado sus sucesores; el coste asociado a cada regla u operador depende de la función objetivo del problema a resolver.

El conjunto de estados y operadores definen el espacio de búsqueda que, en el paradigma de búsqueda en espacios de estados, tiene forma de grafo dirigido simple. En problemas reales, este grafo es tan grande que no es posible representarlo de forma explícita en la memoria del ordenador, por ello se representa de forma implícita a partir de un único estado inicial y del conjunto de operadores. En general, hay varios nodos del espacio de búsqueda que representan soluciones del problema; estos son los objetivos. En la práctica tampoco es posible almacenarlos de forma explícita ya que en general no se conocen o son muchos, por lo que es preciso disponer de una función que permita caracterizarlos, es decir, una función que compruebe si el estado es objetivo.

La estrategia de control es la que decide el orden de exploración de los estados. Podemos emplear una estrategia de control inteligente, o informada, la cual nos debería llevar a explorar primero los nodos que se encuentran en el camino que nos lleva a una solución óptima. Por otro lado, una

estrategia de búsqueda no informada, o búsqueda a ciegas, considerará igual de buenos todos los sucesores de cada estado, por ello requerirá explorar un mayor número de estados que la búsqueda informada para alcanzar una solución. El objetivo de los algoritmos de búsqueda inteligentes es encontrar buenas soluciones, y si es posible la óptima, expandiendo el menor número de estados.

Como se ha indicado anteriormente, los métodos de búsqueda no informados, como su propio nombre indica, no tienen en cuenta información del dominio del problema a resolver por lo que hacen un recorrido sistemático del espacio de búsqueda. El principal problema de estos métodos es que no diferencian los estados prometedores de los que no lo son, por lo tanto emplean más tiempo en encontrar una solución. Ejemplos de estos algoritmos son los esquemas clásicos de búsqueda primero en anchura y primero en profundidad. Como alternativa a estos métodos, podemos emplear mecanismos que dirijan la búsqueda a zonas prometedoras que nos hagan llegar a una solución de forma más rápida y visitando menos estados. Estos mecanismos se denominan en Inteligencia Artificial *heurísticos* o *heurísticas*.

Los heurísticos son criterios, reglas o métodos que nos ayudan a decidir la mejor alternativa, entre varias posibles, para alcanzar un determinado objetivo. Para ello, tienen en cuenta todo tipo de información sobre el dominio del problema a resolver (pistas, intuición, experiencia). En los sistemas de búsqueda los heurísticos se emplean para decidir, entre los nodos candidatos a ser expandidos, el más prometedor, o bien el orden de aplicación de las reglas a un nodo en la generación de sucesores; también se pueden emplear para detectar nodos que no llevan a nada aún teniendo sucesores válidos.

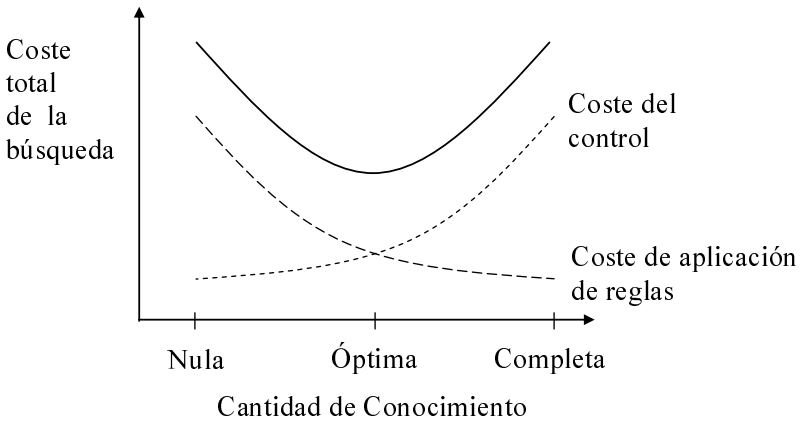


Figura 3.1: Representación del coste total de la búsqueda frente al grado de conocimiento utilizado

El objetivo de los heurísticos es reducir el número de nodos expandidos para alcanzar una solución, sin embargo, su empleo ocasiona a la búsqueda un coste adicional: el de evaluar las distintas alternativas en función del conocimiento disponible. Por ello, lo que se debe perseguir es lograr una situación de compromiso en la que se explote el conocimiento optimizando el coste total de la búsqueda. Esta situación no siempre es fácil de lograr, en la Figura 3.1 se ve la gráfica que relaciona el conocimiento empleado con los diferentes costes de la búsqueda, lo ideal sería lograr la combinación óptima, el punto central de la misma.

Los heurísticos manejan, a veces, conocimiento impreciso o incierto, por lo que es posible que

fallen a la hora de elegir la mejor alternativa. Un buen heurístico es aquel que toma buenas decisiones la mayoría de las veces. Por tanto, un buen heurístico logra mejorar el tiempo medio para alcanzar una solución, aunque en el peor de los casos el tiempo puede ser igual al de una búsqueda no informada.

El término heurístico, en ocasiones hace referencia a procedimientos que no garantizan la calidad de la solución, ni a veces la completitud. Sin embargo, hay algoritmos exactos en los que la búsqueda se guía mediante heurísticos. Este es el caso, por ejemplo del algoritmo A^*.

En este capítulo en primer lugar se introducirá la búsqueda primero el mejor, para pasar a continuación a describir el algoritmo A^*, sus principales propiedades y diferentes versiones del mismo.

3.2. Búsqueda Primero el Mejor

El algoritmo de búsqueda primero el mejor, o BF (Best First), es una especialización del algoritmo general de búsqueda en grafos (propuesto por N. J. Nilsson [59, 58, 57, 55] aunque su principal difusor fue J. Pearl [61]). En él se parte de un grafo definido implícitamente; es decir, se conoce un nodo llamado inicial y un modo de generar el resto de los nodos mediante un conjunto finito de reglas de producción u operadores capaces de producir los sucesores de un nodo. El algoritmo trata de encontrar un nodo solución, meta u objetivo. El conjunto de estos objetivos puede ser vacío (en cuyo caso fracasará el intento) o tener uno o más elementos; en este último caso trataremos de encontrar la mejor solución. Este algoritmo utiliza dos estructuras, que aquí denominaremos $TABLA\_G$ y $TABLA\_A$ siguiendo la descripción propuesta en [22], con el fin de registrar el grafo desarrollado y el mejor camino desde cada nodo al nodo inicial. La $TABLA\_G$ guarda para cada nodo, sus sucesores y el coste del arco que lo lleva a ellos; la $TABLA\_A$ guarda para cada nodo, el nodo anterior siguiendo el camino más corto encontrado hasta el momento y el coste de ese camino.

Expandir un nodo consiste en calcular sus sucesores y los costes correspondientes. Los nodos candidatos a ser expandidos se guardan en la tabla $ABIERTA$, tabla ordenada por una función de evaluación f que para cada nodo n da un valor numérico ($f(n)$) que indica una medida de lo prometedor que es el nodo para ser expandido. Por tanto, la estrategia de control se basa en el criterio de ordenación de $ABIERTA$, luego a la hora de expandir un nodo se elegirá el primero de $ABIERTA$, nodo más prometedor según la función de evaluación f. La función de evaluación f se puede estimar de varias formas, por ejemplo, midiendo la dificultad de resolver el subproblema que plantea el nodo, o bien estimando la calidad de las soluciones que se pueden obtener a partir de ese nodo. De cualquier modo, la función f para un nodo n depende, en general, de la descripción del propio nodo n, de toda la información acumulada en la búsqueda hasta ese punto, y de cualquier conocimiento extra sobre el dominio del problema [61].

Una vez que un nodo ha sido seleccionado para su expansión, sus sucesores se añaden a las estructuras $ABIERTA$, $TABLA\_A$ y $TABLA\_G$ que registran el espacio de búsqueda efectivo. Para ello hay que tener en cuenta que los nodos nuevos pueden estar en una de las siguientes situaciones: en $ABIERTA$, ya han sido expandidos, o aparecen por primera vez (ver Figura 3.2). El algoritmo de búsqueda deberá atender a estas situaciones y determinar en cada caso qué se debe

hacer.

En el algoritmo se utilizan dos funciones de rectificación que permiten actualizar la información acerca del coste para acceder a un nodo desde el inicial. En el caso de que el espacio de búsqueda sea un árbol, este tipo de rectificación no es necesaria; en otros casos, bajo ciertas hipótesis, este tipo de rectificación resulta también innecesaria. Gráficamente la situación de rectificación se produce cuando por ejemplo se da la situación de la Figura 3.3. Para un nodo ya expandido, en el caso de que se tenga que rectificar su coste al inicial, se deberá rectificar lo propio para sus sucesores, para los sucesores de sus sucesores y así sucesivamente.

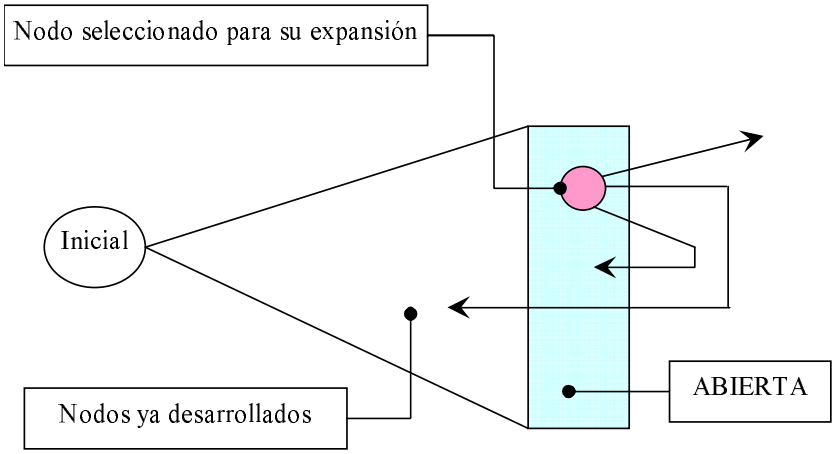


Figura 3.2: Situaciones en las que se puede encontrar un nodo nuevo

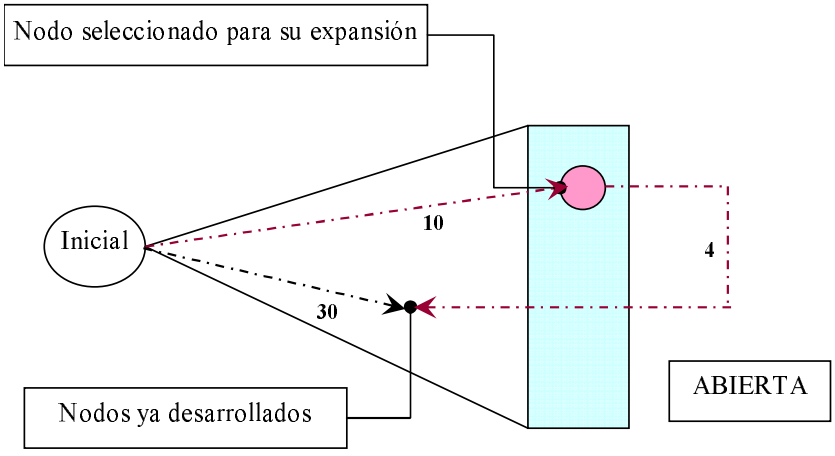


Figura 3.3: Ejemplo de situación de rectificación

3.3. El Algoritmo A^*

El algoritmo A^* se debe a Hart, Nilsson y Raphael, aunque como ya hemos comentado su principal difusor se puede decir que es J. Pearl [61, 55, 57, 58]. Se trata de una especialización o concreción del algoritmo general de búsqueda en grafos ($AGBG$) en su versión BF (Best First), en la que la función de evaluación f se define como una estimación del coste del camino solución condicionado a pasar por el nodo a evaluar [55]. Veamos previamente las siguientes definiciones ([22]):

Definición 3.1. *$g^*(n)$ es el coste del camino más corto, o de menor coste, desde el inicial a n.*

Definición 3.2. *$h^*(n)$ es el coste del camino más corto desde n al objetivo más cercano a n.*

Definición 3.3. *$f^*(n) = g^*(n) + h^*(n)$. Es decir, $f^*(n)$ es el coste del camino más corto desde el inicial a los objetivos condicionado a pasar por el nodo n.*

Definición 3.4. *$C^* = f^*(inicial) = h^*(inicial)$. Es decir, C^* es el coste de la solución óptima.*

Determinar, mediante un método eficiente, los valores de las funciones g^* y h^* para todos los nodos, permitiría lograr un algoritmo de búsqueda que nos conduciría de forma directa a la solución óptima. Para ello, expandiría siempre el nodo de menor f^*, y en caso de empate elegiría un sucesor del último nodo expandido. De este modo solo se expandirían nodos de un camino óptimo. Todos los nodos en el camino óptimo cumplen que $f^*(n) = C^*$, mientras que para aquellos nodos que no se encuentran en un camino óptimo ocurre que $f^*(n) > C^*$. Por tanto, se puede decir que f^* es un "discriminante perfecto" [61], ya que "puede proporcionar señales de rápida advertencia [$f^*(n) > C^*$]" para los nodos fuera del camino óptimo. Si el problema tiene solución seguir el mínimo de f^* constituiría una estrategia perfecta que conduciría directamente a la solución óptima sin salirse del camino en ningún momento. El problema es, por supuesto, que para problemas complejos normalmente no se posee la información suficiente como para poder calcular f^* en un tiempo razonable, por lo que el método de búsqueda anterior no es factible. Lo que se puede hacer es emplear aproximaciones de las funciones g^* y h^*. El algoritmo A^*, utilizan las siguientes aproximaciones.

Definición 3.5. *$g(n)$ es el coste del mejor camino desde el inicial a n obtenido hasta el momento durante la búsqueda.*

Definición 3.6. *$h(n)$ es una estimación positiva del valor de $h^*(n)$, tal que $h(n) = 0$, si n es un objetivo.*

De estas dos definiciones resulta obvio que $g(n) \geq g^*(n)$ y $h(n) \geq 0$. Por último se define la función f del algoritmo BF, para obtener el algoritmo A*, como

Definición 3.7. *$f(n) = g(n) + h(n)$.*

Así, la función de evaluación f que utiliza el algoritmo será una estimación de f^*. A la función h se le suele denominar *función heurística* o simplemente *heurístico*. Su definición no es trivial, y en

ella se tienen en cuenta todo tipo de información obtenida por métodos más o menos formales, como cálculos estadísticos, o cualquier tipo de intuiciones o conocimientos que suelen tener los expertos en la resolución del problema. Más adelante veremos como las propiedades de la función h condicionan el comportamiento del algoritmo A^*.

Por último como consideración general previa habría que advertir que en el libro de Nilsson [57, 58] se denomina a este algoritmo A y sólo se llama A^* cuando la función heurística que se emplea, h, es para todo nodo n menor o igual que la función h^* que debe estimar. Aquí, siguiendo a Pearl y a Nilsson en su libro anterior [55], se llamará A^* a este algoritmo aunque hay que tener en cuenta que sus propiedades más interesantes se cumplen cuando se satisface esta condición respecto a la función heurística h.

3.3.1. Descripción del Algoritmo A^*

Como se ha indicado previamente, el algoritmo A^* es una especialización del algoritmo general de búsqueda en grafos, en su versión BF. El Algoritmo 3.1 muestra la descripción de un algoritmo A^*. Este algoritmo A^* emplea las tablas $TABLA\_G$ y $TABLA\_A$. Siendo la estructura de la $TABLA\_G$ la misma que se ha indicado en la sección 3.2. Sin embargo, la estructura de la tabla $TABLA\_A$ se modifica, guardando para cada nodo además de su antecesor y $g(n)$ (que es lo mismo que el $Coste(inicial, n)$), el valor de $h(n)$. En la descripción aquí planteada se supone que el valor de $h(n)$, una vez encontrado el nodo n, no cambia a lo largo del tiempo. El coste de la evaluación de la función h, en general, no es despreciable, por lo que en esta versión del algoritmo el valor de $h(n)$ se calcula una sola vez para cada nodo n, guardando este valor en la $TABLA\_A$ para su uso posterior.

El algoritmo A^* parte de un único nodo inicial [61], en caso de tener más de un nodo inicial se crearía un nodo ficticio unido a los iniciales mediante arcos de coste 0. Su objetivo es alcanzar un nodo del conjunto de nodos objetivo. Para ello emplea un operador de transición que le permite calcular a partir de un nodo n, sus sucesores. El paso de un nodo a sus sucesores lleva asociado un coste positivo $Coste(n, q)$. En cada iteración se expande el primer nodo de $ABIERTA$, es decir aquel con menor valor de la función de evaluación f. Para cada nodo expandido, calcula sus sucesores y comprueba si es necesario realizar alguna rectificación para éstos. El algoritmo termina o bien cuando se alcanza un nodo objetivo, en cuyo caso devuelve el camino de coste mínimo desde el nodo *inicial*, o bien cuando $ABIERTA$ se queda vacía, debido a que el problema no tiene solución. El algoritmo busca el camino óptimo desde el inicial a los objetivos.

Las funciones de rectificación (Algoritmo 3.2) sirven para poner al día la información sobre cuál es el coste de acceder a un nodo desde el inicial. Este tipo de rectificaciones no siempre son necesarias (en el caso de que el espacio de búsqueda sea un árbol o bajo ciertas hipótesis).

La resolución de un problema mediante el algoritmo A^* debe plantearse como un proceso en el que se ha de describir los siguientes elementos: los estados (estado inicial, estado objetivo), las reglas/operadores que transforman los estados de la búsqueda, los costes de aplicación de las reglas/operadores y la función heurística. Por ejemplo, para el problema JSS podríamos definir cada uno de estos elementos del siguiente modo:

Algoritmo 3.1 Algoritmo A^*

Requiere: Estado inicial para un problema
Produce: Solución del problema, si tiene
  $ABIERTA = (inicial)$;
  mientras $NoVacia(ABIERTA)$ **hacer**
    $n = ExtraePrimero(ABIERTA)$;
    si $EsObjetivo(n)$ **entonces**
      devuelve $Camino(inicial, n)$ y para;
    fin si
    $S = Sucesores(n)$;
    Añade S a la entrada de n en la $TABLA\_A$;
    para cada q de S **hacer**
      si $(q \in TABLA\_A)$ **entonces**
        $Rectificar(q, n, Coste(n, q))$;
        $Ordenar(ABIERTA)$; {*si es preciso*}
      si no
        pone q en la $TABLA\_A$ con
            $Anterior(q) = n$,
            $g(q) = g(n) + Coste(n, q)$,
            $h(q) = Heuristico(q)$;
        $ABIERTA = Mezclar(q, ABIERTA)$;
      fin si
    fin para
  fin mientras
  devuelve "no solución";

- **Estados:** Situaciones en las que algunas tareas están planificadas (tienen asignado un tiempo de inicio) y otras no. En cada estado, se guardarán por tanto las tareas y sus tiempos de inicio (si está planificada), así como el valor de las funciones g y h. El *estado inicial*, será aquel en el que ninguna tarea tenga asignado un tiempo de inicio. Por otra parte, el proceso de búsqueda llegará a alcanzar un *estado objetivo* cuando el estado en estudio tenga planificadas todas sus tareas.

- **Regla/Operador de transición:** Calculará sucesores de un estado planificando una tarea de forma compatible con las que ya están planificadas en el estado.

- **Coste de aplicación de la Regla/Operador de transición:** El coste de aplicación de las reglas u operadores que nos lleven de un estado n a un sucesor n', para este problema, dependerá de la función objetivo. Por ejemplo en el caso de la versión con minimización del makespan, será igual a la diferencia $makespan(n') - makespan(n)$, teniendo en cuenta que el makespan de un estado cualquiera es igual al máximo de los tiempos de finalización de las tareas que tenga planificadas.

Una vez definido el espacio de búsqueda, viene la tarea más difícil que consiste en utilizar el conocimiento experto sobre el problema para establecer la estrategia de control inteligente que limite el espacio de búsqueda efectivo. En el caso de A^*, se trata de definir una función heurística que mejore la eficiencia de la búsqueda.

Algoritmo 3.2 Funciones de rectificación de nodos ya expandidos

Requiere: Estados p y n, y coste de p a n
Produce: Rectifica el coste del camino de p a n

```
Función Rectificar (n, p, costepn);
si (g(p) + costepn < g(n)) entonces
   Modifica la entrada del nodo n en la TABLA_A con g(n) = g(p) + costepn; Anterior(n) = p;
   RectificarLista(n);
fin si

Función RectificarLista (n);
LISTA = Sucesores(n); /* Registrados en la TABLA_A */
para cada q de LISTA hacer
   Rectificar(q, n, Coste(n, q));
fin para
```

3.3.2. Propiedades Formales

En esta sección revisamos las propiedades formales del algoritmo A^* que, como veremos, dependen de la definición de la función h. Concretamente, consideraremos un resumen, sin incluir las demostraciones, de las principales propiedades de terminación, completitud, admisibilidad y eficiencia. Para consultar las demostraciones de las propiedades enunciadas en esta sección o realizar un estudio más exhaustivo de las mismas se pueden seguir los textos de N. J. Nilsson [59, 55, 57, 58], J. Pearl [61] y [22].

Antes del estudio de estas propiedades vamos a introducir la notación utilizada: s denota el estado *inicial* y Γ el conjunto de nodos objetivo. $\Gamma^* \subseteq \Gamma$ es el conjunto de objetivos óptimos. $P_{n-n'}$ denota un camino simple desde el nodo n al nodo n', y $P^*_{n-n'}$ un camino óptimo desde n a n'.

Las diferentes situaciones que se dan a lo largo del proceso de búsqueda se describirán teniendo en cuenta los siguientes conjuntos de nodos: *nodos encontrados*, los que ya tienen una entrada en la $TABLA\_A$, *nodos expandidos* los que fueron seleccionados para su expansión y que por lo tanto ya tienen registrados sus sucesores en el campo correspondiente de la $TABLA\_G$, y los *nodos no encontrados* los que aun no han aparecido durante la búsqueda. Antes de la expansión de un nodo se cumple $NodosEnABIERTA \cup NodosExpandidos = NodosEncontrados$. Esta unión es disjunta, siendo $ABIERTA$ una frontera entre los nodos ya expandidos y aquellos no encontrados todavía durante la búsqueda. Por tanto, cualquier camino $P_{n-n'}$, con n ya expandido y n' aún no expandido, tiene al menos un nodo en $ABIERTA$.

Propiedades Generales de los Algoritmos de Búsqueda

Definición 3.8. *Un algoritmo de búsqueda es completo si termina con una solución siempre que ésta exista.*

Definición 3.9. *Un algoritmo es admisible si devuelve una solución óptima (con un camino desde el nodo inicial con mínimo coste posible) cuando exista solución.*

Definición 3.10. *Un algoritmo A_1 domina a otro A_2 (siendo la única diferencia entre A_1 y A_2 el heurístico; h_1 en el primer caso y h_2 en el segundo) si todo nodo expandido por A_1 también es expandido por A_2. Se dice que A_1 domina estrictamente a A_2 (o, simplemente, que A_1 es más eficiente que A_2) si A_1 domina a A_2 pero A_2 no domina a A_1.*

Definición 3.11. *Un algoritmo es óptimo sobre una clase de algoritmos si domina a todos ellos.*

La eficiencia en los algoritmos de búsqueda viene dada por el número de nodos que expanden o visitan antes de alcanzar la solución si los costes computacionales de evaluación de los heurísticos son similares. Cuantos menos sean estos nodos tanto más rápido (eficiente) será el algoritmo de búsqueda. Por otra parte, un algoritmo de búsqueda, además de rápido, debe cumplir una condición indispensable, que es cumplir la misión para la cual fue diseñado; es decir, encontrar una solución (óptima) siempre que ésta exista. Todas estas definiciones, son conceptos importantes que son utilizados por las propiedades formales. El algoritmo A^* es, en realidad, un esquema de algoritmo más que un algoritmo concreto. Su rendimiento estará en función de las definiciones concretas que se usen en los elementos que dependen de cada implementación: las reglas que marcarán los sucesores de un nodo dado y las funciones que definen la función de evaluación de los nodos; la función f que se define, como se ha visto, a partir de las funciones g y h.

Terminación y Completitud

Veamos ahora algunas proposiciones y definiciones relacionadas con el algoritmo BF descrito en la sección 3.2.

Proposición 3.3.1. *El algoritmo general BF siempre termina en grafos finitos.*

Proposición 3.3.2. *El algoritmo general BF es completo para grafos finitos. Es decir, termina con una solución (no necesariamente óptima) siempre que ésta exista.*

Definición 3.12. *Un grafo es localmente finito si cada nodo tiene un número finito de hijos o sucesores.*

Por tanto en el peor de los casos, para grafos finitos, el algoritmo BF termina tras expandir todos los nodos. Esto es claro ya que el algoritmo expande un nodo en cada iteración y ningún nodo se expande más de una vez. En grafos infinitos, aunque sean localmente finitos, el algoritmo BF puede no terminar, esto ocurre cuando la definición de la función f no garantiza que el algoritmo no se vaya por una rama infinita del espacio de búsqueda. En el algoritmo A^*, esto no ocurre pues f se define como $g + h$, donde h siempre es positiva, y g crece de forma no acotada a lo largo de cualquier camino con un número no finito de nodos. Si en el algoritmo A^* la búsqueda se fuese a través de un camino infinito, en algún momento el valor de f sería suficientemente grande para que el siguiente nodo a través de ese camino no fuese el más prometedor, y por lo tanto se consideraría la búsqueda a través de otros caminos. Podemos por tanto enunciar el siguiente teorema

Teorema 3.1. *A^* es completo en grafos localmente finitos.*

Admisibilidad del Algoritmo A*

En esta sección estudiaremos las condiciones en las que el algoritmo A^* es admisible (encuentra la solución óptima). Para ello, en primer lugar damos la siguiente definición

Definición 3.13. *Una función heurística h es admisible sii $h(n) \leq h^*(n)\ \forall n$.*

Para que el algoritmo A^* sea admisible, es necesario que la función heurística empleada por él, h, sea una estimación optimista del valor de h^* para todos los nodos. Por tanto, supondremos de ahora en adelante que se dispone de un algoritmo A^* que emplea una heurística h admisible y en el que el grafo a explorar tiene solución. En estas condiciones se puede probar la siguiente proposición

Proposición 3.3.3. *En todo momento antes de terminar el algoritmo A^* existe un nodo n en ABIERTA del camino P con $f(n) \leq C^*$.*

Donde P denota un camino del nodo inicial a un nodo objetivo óptimo en el sentido de que el coste de P es el menor posible en estas condiciones.

A partir de este resultado se puede probar el siguiente teorema

Teorema 3.2. *A^* es admisible.*

Como corolario de lo anterior se tiene que los algoritmos de primero en anchura y de coste uniforme son admisibles, pues son casos particulares del algoritmo A^* con $h(n) = 0\ \forall n$. Como consecuencia de esto podemos preguntarnos si estos algoritmos resultan ser iguales o no a otras versiones de A^* con cualquier otro heurístico admisible. La respuesta es que no son iguales, tal y como se indica más adelante, cuando se ve en qué condiciones un heurístico se puede considerar más eficiente que otro, siendo ambos admisibles; y en qué condiciones se puede eliminar la operación de rectificación de nodos ya expandidos, y en consecuencia la necesidad de registrar el grafo expandido hasta el momento.

Comparación de Heurísticos Admisibles

En esta sección se estudia cómo influye el heurístico en la eficiencia del algoritmo A^*. Supondremos que los heurísticos son admisibles y veremos cuándo unos heurísticos son más eficientes que otros. Supondremos que el coste de evaluación de todas las funciones heurísticas es similar, para que el coste, en tiempo de ejecución, de las versiones del algoritmo A^* con los diferentes heurísticos, sea proporcional al número de nodos expandidos para llegar a una solución.

El algoritmo A^* cuando emplea f^* realiza una búsqueda directa, sin embargo con una función heurística $h(n) = 0, \forall n$ la búsqueda es "primero en anchura", seguramente la búsqueda más lenta que se puede hacer dentro del esquema A^*. Obviamente entre estos dos extremos están los casos prácticos usuales. Desde el punto de vista formal la diferencia de estos dos casos está en la estimación de h^*; perfecta en el primer caso y exageradamente optimista (0 en todos los casos) en el segundo. Por tanto, las estimaciones posibles en el esquema A^* de h^* estarán en la franja comprendida entre los valores de h^* para cada nodo y 0 (entre estos dos valores se moverán todas las estimaciones admisibles), es decir, $\forall n, h(n) \in [0, h^*(n)]$. A continuación veremos que cuanto más próxima esté una estimación a h^* mejor será (en el sentido de producir algoritmos más eficientes).

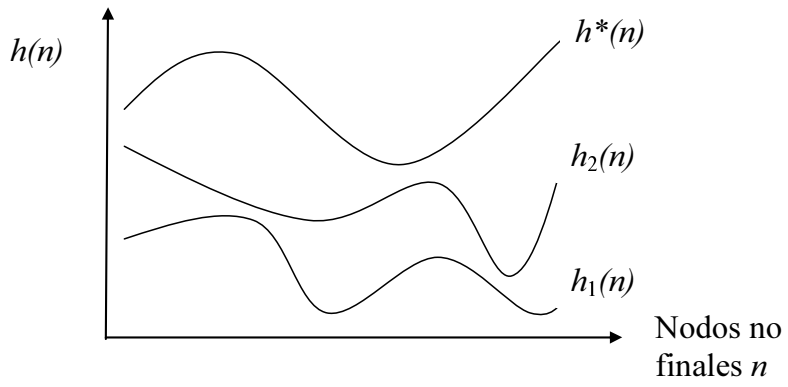


Figura 3.4: Dos heurísticos admisibles con distinto grado de información

Supongamos que tenemos dos heurísticos admisibles h_1 y h_2, tales que $h_2(n) > h_1(n)$ para todo nodo n no final. Intuitivamente se ve que h_2 contiene más información que h_1, pues como se observa en la Figura 3.4 está más cercano al heurístico h^* que es el que contiene la información completa. Por tanto, el algoritmo A^* debería ser más eficiente con h_2 que con h_1. Para que esto sea así, se ha de probar que todo nodo expandido con el heurístico h_2 será también expandido con h_1, siendo en el peor de los casos el número de nodos expandidos el mismo. Si existe algún nodo no final n tal que $h_2(n) \leq h_1(n)$ este resultado formal ya no es cierto. Para probar que todo nodo expandido por h_2 también lo es con h_1 se han de establecer unas condiciones necesaria y suficiente de expansión, y luego probar que si un nodo n cumple la condición necesaria de expansión con h_2, entonces cumple la condición suficiente con h_1. Veamos ahora una serie de definiciones que formalizan las ideas anteriores.

Definición 3.14. *Un heurístico h_2 está más informado que otro heurístico h_1 si los dos son admisibles y $h_2(n) > h_1(n)$ para todo nodo n no final. Se dirá también que el algoritmo A_2^* que utiliza el heurístico h_2 está más informado que el algoritmo A_1^* que utiliza el heurístico h_1.*

Definición 3.15. *Un algoritmo A_2^* domina a otro algoritmo A_1^* si los dos son admisibles y todo nodo expandido por A_2^* es también expandido por A_1^*.*

Para probar, en estas condiciones de dominio, que la versión más informada domina a la menos informada, se han de establecer las condiciones necesaria y suficiente de expansión para heurísticos admisibles.

Teorema 3.3. *[Una condición necesaria para que un nodo sea expandido por A^*]: Cualquier nodo expandido por A^* no puede tener un valor de f que exceda a C^*, es decir, $f(n) \leq C^*$ para todos los nodos expandidos por A^*.*

Teorema 3.4. *[Una condición suficiente para la expansión de un nodo por A^*]: Si un nodo n tiene $f(n) < C^*$, entonces será expandido por A^* antes de terminar.*

Las condiciones de los teoremas 3.3 y 3.4 son dinámicas, dependen de la evolución del algoritmo A^* pues hacen referencia al valor de f que es $g + h$, siendo el valor de g una variable del algoritmo

definida sólo para los nodos encontrados. Este es el motivo por el que aunque un nodo n sea expandido con el heurístico h_2 y cumpla la condición necesaria de expansión ($g(n) + h_2(n) \leq C^*$), no podemos concluir que cumple la condición suficiente de expansión con h_1 ($g(n) + h_1(n) < C^*$). Esto se debe, a que no se puede suponer que con el heurístico h_1 el nodo n va a estar también en *ABIERTA* en algún momento y con el mismo valor de g que en el momento de su expansión con h_2. No obstante, estos dos teoremas nos ayudan a probar dos condiciones de carácter estático (independientes de la ejecución del algoritmo A^*) a partir del espacio de búsqueda y del heurístico. Antes de enunciar estas dos nuevas condiciones, definiremos lo que es un camino C^*-acotado

Definición 3.16. *Un camino P desde el inicial s a un nodo n es C-acotado, sii $\forall n \in P$ se cumple que $g_P(n') + h(n') \leq C$, y es estrictamente C-acotado si $g_P(n') + h(n') < C$. Siendo $g_P(n')$ el coste desde s a n' a través del camino P.*

Teorema 3.5. *Una condición suficiente para que un nodo n sea expandido por A^* es que exista un camino P del inicial a n que sea estrictamente $C^* - acotado$.*

Teorema 3.6. *Una condición necesaria para que A^* expanda un nodo n es que exista un camino $C^* - acotado$ del inicial a n.*

Gracias a estas dos condiciones se puede concluir el siguiente resultado

Teorema 3.7. *Si A_2^* está mas informado que A_1^*, entonces A_2^* domina a A_1^*.*

En la práctica la aplicación de este resultado es limitada. En muchas ocasiones se tienen dos heurísticos que cumplen la relación ($h_2(n) > h_1(n)$) para la mayoría de los estados no finales, pero no en algunos estados muy próximos a los objetivos en los que toman el mismo valor. En estas situaciones, lo que ocurre normalmente es que el heurístico h_2 se comporta mucho mejor que el heurístico h_1, y en general esta mejora es tanto mayor cuanto mayor sea la diferencia entre los dos heurísticos en la mayoría de los estados intermedios.

Heurísticos Consistentes o Monótonos

En la sección anterior hemos comparado heurísticos admisibles a través del número de nodos expandidos. Sin embargo, la eficiencia del algoritmo A^* puede estar condicionada a posibles rectificaciones en los nodos expandidos. Teniendo en cuenta esto, se introducen dos nuevas propiedades de las funciones heurísticas: la consistencia y la monotonía. Se trata de dos propiedades equivalentes que evitan, para el heurístico que las cumple, rectificaciones del contenido de la $TABLA\_A$ para aquellos nodos que ya han sido expandidos.

Definición 3.17. *Un heurístico h es monótono si para todo par de nodos n y n' se cumple que*

$$h(n) \leq k(n, n') + h(n')$$

donde $k(n, n')$ representa el coste mínimo para ir de n a n', y por lo tanto es infinito si no hay un camino desde n a n'.

Definición 3.18. *Un heurístico h es consistente si para todo par de nodos n y n' se cumple que*

$$h(n) \leq c(n, n') + h(n')$$

donde $c(n, n')$ representa el coste de la regla que lleva de n a n', y por lo tanto es infinito si no existe esta regla.

Ambas propiedades se cumplen, para todo nodo n, en el caso de h^* y del heurístico trivial $h(n) = 0$. Puede parecer que la monotonía es menos restrictiva que la consistencia, sin embargo como prueba el siguiente teorema son propiedades equivalentes.

Teorema 3.8. *Monotonía y Consistencia son propiedades equivalentes.*

Una de las consecuencias de la monotonía o consistencia es la admisibilidad.

Teorema 3.9. *Todo heurístico monótono es admisible.*

Aunque la implicación inversa no se cumple, es decir monotonía y admisibilidad no son propiedades equivalentes. En la práctica no es fácil encontrar heurísticos admisibles que no sean a la vez monótonos.

El empleo de heurísticos monótonos tiene muchas ventajas, entre ellas la que enuncia el siguiente teorema.

Teorema 3.10. *Si h es monótono y A^* elige el nodo n para expansión, entonces se cumple que $g(n) = g^*(n)$. Es decir que el camino encontrado hasta el momento desde el inicial a n es óptimo.*

Por el Teorema 3.10, cuando el heurístico es monótono, no es necesario realizar rectificaciones en los nodos ya expandidos, por lo que tampoco es necesario registrar el grafo de búsqueda desarrollado hasta el momento en el campo correspondiente a los sucesores de la $TABLA\_A$, pues los únicos nodos que pueden sufrir rectificaciones son los que se encuentran en $ABIERTA$, por lo que la operación *RectificarLista* nunca se aplica. De este modo el algoritmo A^* es más eficiente. Como se verá más adelante, si $h_2(n) \geq h_1(n)$ para todo n, es decir que se da la igualdad de los dos heurísticos para algunos nodos, si son monótonos, se puede acotar el número de nodos que pueden ser expandidos por h_2 y no por h_1.

Las condiciones, necesaria y suficiente, se simplifican con respecto a las condiciones de los heurísticos admisibles, pues sólo requieren evaluar una condición simple sobre el nodo n y no sobre un camino desde el inicial a n.

Teorema 3.11. *Si h es monótono, la condición necesaria de expansión de un nodo n es*

$$g^*(n) + h(n) \leq C^*$$

y la condición suficiente

$$g^*(n) + h(n) < C^*$$

Por la condición suficiente de expansión, $g^*(n) + h(n) < C^*$, cuanto mejor informado sea el heurístico h, menor será el número de nodos que cumplen la condición, por tanto es de esperar que el número de nodos expandidos sea menor. Estas condiciones tienen un papel importante en la regla de desempate entre nodos con el mismo valor de f en *ABIERTA*. Consideremos diferentes versiones de A^*, que emplean la misma función heurística y que se diferencian únicamente en la regla de desempate. Los nodos que pueden ser expandidos por una versión y no por otra son los que cumplen $g^*(n) + h(n) = C^*$, que al estar en abierta tendrán $f(n) = C^*$. Por tanto el algoritmo A^* expandirá todos los nodos que cumplan $g^*(n)+h(n) < C^*$, alguno de los que cumplan $g^*(n)+h(n) = C^*$, y por supuesto ninguno de los que cumplan $g^*(n)+h(n) > C^*$. Se puede probar que cualquier algoritmo admisible que utilice la misma información h, tiene que expandir todos los nodos tales que $g^*(n)+h(n) < C^*$ para asegurar la optimalidad de la solución, ese es el motivo por el que cuando h es monótono el algoritmo A^* domina ampliamente (véase la definición siguiente) a cualquier algoritmo admisible que utilice la misma información heurística. La prueba formal de este resultado se puede ver en [23] y [61].

Además, gracias a estas dos condiciones podemos realizar una comparación no estricta entre dos algoritmos A_1^* y A_2^* que utilizan heurísticos monótonos h_1 y h_2 respectivamente, que no cumplen la desigualdad estricta para todos los nodos no finales, es decir para los que $h_2(n) \geq h_1(n), \forall n$.

Definición 3.19. *Un algoritmo A_2^* domina ampliamente a otro algoritmo A_1^* si todo nodo expandido por A_2^* es también expandido por A_1^*, excepto, quizás, algunos nodos para los cuales se cumple $h_2(n) = h_1(n) = C^* - g^*(n)$.*

Teorema 3.12. *Si $h_2(n) \geq h_1(n)$ para todo n y los dos son monótonos, entonces A_2^* domina ampliamente a A_1^*.*

Este resultado nos permite, con un cierto grado de confianza concluir que el heurístico h_2 es más eficiente que el heurístico h_1. Para que un nodo sea expandido por h_2 y no por h_1, tienen que producirse dos igualdades, algo poco probable si los heurísticos toman valores reales.

3.4. Relajación de las Condiciones de Optimalidad

Hasta ahora hemos visto las condiciones en las que el algoritmo A^* encuentra soluciones óptimas. Debido a la complejidad de muchos problemas, no es posible diseñar buenos heurísticos, aquellos con un valor muy cercano a h^* y con poco tiempo de evaluación. Esto hace que sea necesario expandir una cantidad de nodos muy elevada para alcanzar una solución. Esto implica un elevado consumo de recursos computacionales (tiempo de búsqueda y espacio de almacenamiento), lo cual para ciertos problemas puede no ser aceptable. Como indica Pearl ([61], p. 86), a menudo el algoritmo A^* emplea mucho tiempo en discriminar entre caminos de coste similar. Esto se debe a que la admisibilidad fuerza al algoritmo a emplear mucho tiempo en la selección del mejor candidato entre varios prácticamente iguales. Esto hace que en problemas de cierta complejidad se agoten los recursos computacionales sin que el algoritmo encuentre una solución, en estos casos, sería preferible que el algoritmo terminase con una solución, aunque no fuese óptima.

Por tanto, puede ser razonable renunciar a la admisibilidad, a cambio de lograr una mayor eficiencia del algoritmo A^*. En la práctica esto es así en la mayoría de problemas reales que se resuelven con técnicas de inteligencia artificial, ya que de ese modo se puede encontrar una solución con cierta calidad en poco tiempo, algo deseable normalmente en este tipo de problemas. En las subsecciones siguientes veremos algunas de las ideas propuestas para diseñar algoritmos con estas características.

3.4.1. Ajuste de los Pesos g y h

Como se ha indicado anteriormente el algoritmo A^* emplea una estrategia de búsqueda basada en el valor de la función de evaluación $f = g+h$. En ella, el sumando g trata de ajustar el algoritmo a una estrategia de búsqueda en anchura o de coste uniforme, garantizando así encontrar soluciones óptimas. Por otro lado, el sumando h trata de dirigir la búsqueda hacia la solución de modo que no sea necesario expandir todos los nodos que disten igual del inicial. Junto con Pohl ([67, 68]), han sido muchos los investigadores que han estudiado el efecto de ponderar los términos g y h en la función de evaluación f, para así lograr que el algoritmo A^* alcance soluciones, cotas superiores de la óptima, con menor esfuerzo computacional.

3.4.2. Algoritmos ε-admisibles

Esta familia de algoritmos sacrifica la obtención de una solución óptima a cambio de obtener mejorar en el rendimiento del proceso, pero controlando la pérdida de la calidad de la solución obtenida mediante el factor ε que representa la distancia máxima al coste óptimo.

Definición 3.20. *Un algoritmo de búsqueda es ε-admisible si siempre encuentra una solución con un coste que no excede el valor $(1+\varepsilon)C^*$ (una solución de este tipo se denomina solución ε-óptima).*

En esta sección veremos las tres versiones más conocidas de esta familia de algoritmos: la ponderación estática, la ponderación dinámica y el algoritmo $A\varepsilon^*$.

Ponderación Estática

Pohl fue el primero en plantear el cálculo de soluciones no óptimas mediante ponderación estática de los términos de f. En [67] propone modificar la función de evaluación f, ponderando la función heurística, es decir propone una nueva función de evaluación f' definida para todo nodo n como $f'(n) = g(n)+wh(n)$, con $w > 1$. Si $w > 1$ la búsqueda no es admisible y la primera solución que se encuentra puede no ser óptima pero normalmente se encuentra mucho más rápidamente. Davis y sus colaboradores en [21] prueban que si h es admisible, el coste de la solución alcanzada ponderando h no puede superar al coste de la solución óptima más de un factor de w. Esta solución alcanzada se dice que es ε-admisible con $\varepsilon = w - 1{,}0$.

Ponderar el heurístico acelera el alcance de una solución, pues hace más atractivos los nodos más cercanos a una solución, dando a la búsqueda un aspecto de búsqueda en profundidad y ajustando implicitamente el equilibrio entre el esfuerzo de la búsqueda y la calidad de la solución.

En ocasiones la función de evaluación ponderada f_w se define, ponderando los términos g y h, del siguiente modo:

$$f_w(n) = (1 - w')g(n) + w'h(n), w' \in [0, 1].$$

Esta función f_w es equivalente a f' cuando $w' = w/(1 + w)$. Cuando $w = 0$ tenemos una estrategia de coste uniforme, con $w = 1/2$ el A^* convencional y con $w = 1$ un BF (con $f = h$). Si h es admisible, es fácil probar que el algoritmo es admisible en el intervalo $0 \leq w \leq 1/2$, pero dependiendo de lo lejana que esté h de h^* se puede perder esta propiedad en el intervalo $1/2 \leq w \leq 1$.

Este tipo de ponderación se denomina ponderación estática, ya que el peso w no varía a lo largo del proceso de búsqueda. Encontrar un valor adecuado para w es la principal dificultad con la que nos encontramos al aplicar esta técnica. En la práctica este valor se suele encontrar de forma experimental. Por ejemplo, Pearl ([61], p. 87) presenta resultados experimentales en este sentido en los que se muestra que mientras que para el problema del $15 - puzzle$ la mayor eficiencia se encuentra en el intervalo $1/2 \leq w \leq 1$, para el $8 - puzzle$, lo mejor es $w = 1/2$. Además comprueba que a medida que se aumenta el valor, se obtiene un número mucho mayor de nodos expandidos.

Ponderación Dinámica

Esta estrategia se debe a Pohl [69]. En este trabajo en lugar de fijar unos pesos estáticos, éstos cambian de manera dinámica a lo largo del proceso de búsqueda. La idea es que h tenga un peso mayor al principio de la búsqueda y que este se reduzca a medida que nos acercamos a una solución para asegurar una búsqueda en anchura que afine lo más posible la solución. La función de evaluación que se utiliza con este método es

$$f(n) = g(n) + h(n) + \varepsilon(1 - d(n)/N)h(n)$$

donde $d(n)$ es la profundidad del nodo n y N es una cota superior de la profundidad de la mejor solución. El número $\varepsilon > 0$ indica la desviación que estamos dispuestos a admitir.

No es difícil probar que si el heurístico es admisible, entonces el algoritmo es ε-admisible.

Teorema 3.13. *Si el heurístico h es admisible, el algoritmo de ponderación dinámica es ε-admisible.*

Algoritmo Aε^*

Esta versión se debe a J. Pearl y J. H. Kim ([61], pp.88). La idea consiste en considerar dentro de $ABIERTA$ una sublista, llamada $FOCAL$, formada por aquellos nodos cuyo valor de f no se separa más de un factor $(1 + \varepsilon)$ del valor mínimo de f en $ABIERTA$, es decir

$$FOCAL = \{n \in ABIERTA / f(n) \leq (1 + \varepsilon) min(f(n'), n' \in ABIERTA)\}.$$

La estrategia de Aε^* es idéntica a la de A^*, salvo que Aε^* desarrolla primero los nodos de la sublista $FOCAL$ según los valores de un segundo heurístico h', donde $h'(n)$ es una estimación del esfuerzo computacional que se necesitaría para llegar desde n a una solución.

Los nodos de *FOCAL* tienen todos más o menos las mismas posibilidades de estar en un camino óptimo, (según las estimaciones de f), por ello, siempre que se disponga de él, es razonable emplear un conocimiento adicional (h') al de la función f para decidir cuál es el nodo más prometedor. Como heurístico h' se podría utilizar el propio h, pero por lo general se podrá añadir alguna información adicional a h', que se puede obtener, por ejemplo, a partir de lo ya explorado. Además, se ha de tener en cuenta que la elección de h' nunca influye negativamente en la ε-admisibilidad del algoritmo Aε^*. En este caso también se puede probar el siguiente resultado.

Teorema 3.14. *Si h es admisible, Aε^* es ε-admisible.*

Comparación de los Algoritmos

Compararemos aquí los algoritmos que emplean las tres técnicas enunciadas anteriormente. Si comparamos el algoritmo A^* empleando ponderación dinámica con Aε^*, vemos que es un método mucho más simple pues maneja únicamente una lista de estados y un heurístico, sin embargo, tiene el inconveniente de que es necesario conocer la profundidad de la solución óptima, o al menos una buena cota superior. Por otra parte, Aε^*, al utilizar un segundo heurístico *h'*, que no tiene que ser admisible, ofrece un mecanismo más flexible del que ofrecen las funciones g y h, para hacer estimaciones del coste computacional de la búsqueda en una determinada dirección.

A pesar del buen comportamiento de los dos algoritmos comparados en el párrafo anterior, la experiencia acumulada por los investigadores a lo largo de décadas sugiere que para aquellos problemas en los que es deseable alcanzar una solución, no necesariamente óptima, de forma rápida la mejor técnica es la ponderación estática, tal y como indican Thayer y Ruml en [87]: "For problems in which a solution with bounded sub-optimality is desirable, weighted A* has reigned for decades as the technique of choice".

Otras Estrategias ε-admisibles

En la literatura, además de las de Pohl ([67, 68]), existen multitud de aproximaciones que emplean la ponderación estática. Además, la ponderación estática en la búsqueda heurística también ha sido empleada con otros algoritmos además del A^*, por ejemplo en versiones del A^* como son el IDA^* y $RBFS$ ([41]), así como $LRTA^*$ (Learning Real-Time A^*) [75], y en algoritmos de búsqueda heurística para grafos AND/OR ([18, 36]).

Recientemente se han presentado nuevas aproximaciones que son competitivas con la versión del algoritmo A^* ponderado estáticamente.

Hansen y Zhou proponen en [35] la transformación del algoritmo A^* en un algoritmo A^* "*anytime*" que encuentra una secuencia de soluciones mejoradas, empleando ponderación estática, y que eventualmente converge a una solución óptima (AWA^*). Este algoritmo emplea para la expansión de los nodos una función de evaluación no admisible f' (con h ponderada estáticamente), continúa la búsqueda tras alcanzar una solución y mantiene una función de evaluación admisible f (una función lower-bound) que se emplea junto con las cotas superiores del coste de la mejor solución alcanzada para podar el espacio de búsqueda y detectar la convergencia a una solución óptima. Hansen aplica el algoritmo AWA^* a la resolución de los problemas n-puzzle, planificación tipo STRIPS y

alineamiento de múltiples secuencias, y lo compara con el A^*. Los resultados que obtiene indican que en general AWA^* es efectivo en problemas para los que el A^* con ponderación estática no lo es, además los resultados mostraron que utilizando w adecuados puede incluso converger a una solución óptima utilizando menos memoria y tiempo que el A^*, esto se debe a que a pesar de expandir más nodos que el algoritmo A^*, el empleo de cotas (superiores e inferiores) permite reducir el número de nodos en $ABIERTA$ por lo que la memoria y el tiempo empleado en la gestión de $ABIERTA$ se reduce.

Otra de las aproximaciones más recientes es la propuesta por Thayer y Ruml en [87]. En este trabajo se presenta una nueva técnica para una búsqueda ε-admisible rápida, denominada algoritmo optimista. Esta propuesta aprovecha el hecho de que el coste de la solución encontrada es, en general, mucho menor que $(1+\varepsilon)C^*$. Este algoritmo consta de dos fases. En la primera realiza una búsqueda voraz para encontrar una solución que no tiene garantías de ser ε-admisible (utilizando una ponderación muy alta), aunque normalmente lo es. Para ello utiliza un algoritmo A^* con una ponderación muy alta ($2(w-1)+1$, con $w = 1+\varepsilon$), en el que al igual que se hace en AWA^* se mantiene tanto la f' como la f. En la segunda fase el algoritmo trata de probar que dicha solución obtenida si es ε-admisible, para ello una vez alcanzada una solución continúa expandiendo nodos hasta que se puede probar que la solución ofrecida es ε-admisible o hasta que se alcanza una solución mejor. El riesgo de está técnica radica en que la solución encontrada en la primera fase no sea ε-admisible, en cuyo caso el algoritmo se comporta como el algoritmo A^* expandiendo todos los nodos con valores de f menores que la solución óptima. Para acotar esta situación, si existe un nodo cuyo valor heurístico no admisible es menor que la solución calculada, según la búsqueda voraz, este nodo se elige para ser expandido. En este trabajo se hace un estudio comparativo de diferentes implementaciones del A^* ponderado (WA^* con ponderación estática, Aε^*, DWA^* con ponderación dinámica y el algoritmo AWA^* empleando la misma ponderación que la búsqueda optimista) aplicadas a diferentes dominios de problemas (planificación temporal (mundo de bloques, logística, satelite, etc...), planificación Grid-world , viajante de comercio TSP, y N-reinas). En general el mejor comportamiento se obtiene con el WA^*, pero este estudio pone de manifiesto que la búsqueda optimista es una técnica simple, predecible y efectiva con resultados similares a WA^* cuando el valor de la ponderación no es el adecuado al problema.

3.5. Diseño Sistemático de Heurísticos

Como ya hemos indicado anteriormente, diseñar buenos heurísticos no es un proceso fácil, y requiere utilizar todo tipo de informaciones e intuiciones sobre el dominio del problema, así como una buena dosis de imaginación. Sin embargo, el diseño de heurísticos para algunos problemas puede resultar una tarea inabordable si no se dispone de un método sistemático que permita controlar la admisibilidad y la calidad de los heurísticos diseñados.

En ([61]), J. Pearl describe los fundamentos de un método sistemático de diseño de heurísticos que se basa en la relajación de las restricciones de un problema. El método considera una versión simplificada o relajada del problema original, que se pueda resolver de forma óptima mediante un algoritmo polinomial. El coste óptimo del problema relajado se toma como una estimación del coste

del problema original. Se trata de una estimación claramente optimista ya que todas las soluciones del problema real son también soluciones del problema relajado, pero no al revés.

A la hora de aplicar el método de forma sistemática, en primer lugar se ha de enunciar el problema reflejando de forma explícita todas las restricciones del mismo. Una vez hecho esto se han de considerar las posibles relajaciones. Para ello se supone que se relaja un subconjunto de las restricciones del problema hasta conseguir una versión del problema suficientemente simple que pueda ser resuelta con un algoritmo polinomial. A la hora de ir realizando estas relajaciones se ha de tener en cuenta que cuanto menos se relaje el problema, mejor será el heurístico resultante pues el problema relajado se parecerá más al problema original. En el desarrollo de esta tesis se ha utilizado está técnica en el diseño de los diferentes heurísticos empleados.

Los heurísticos obtenidos por el método de relajación del problema son admisibles. Además, también son monótonos.

Teorema 3.15. *Todo heurístico h obtenido por el método de la relajación del problema es monótono.*

Este resultado pone de manifiesto la eficacia de este método sistemático de diseño de heurísticos, ya que proporciona heurísticos con buenas propiedades que no es necesario probar al ser consecuencia directa de la correcta aplicación del método. Además, esta forma de actuar es la que tienden a utilizar las personas de forma inconsciente, pues la experiencia demuestra que la mayoría de los heurísticos admisibles diseñados razonando de una forma lógica son también monótonos.

3.6. Búsqueda con Memoria Limitada

El mayor problema del algoritmo A^* es que, aún empleando buenos heurísticos, la memoria consumida por el algoritmo crece exponencialmente con la profundidad. Para buscar solución a este problema han surgido diferentes variantes del algoritmo A^* que tratan de limitar la cantidad de memoria consumida. Una es el algoritmo IDA^* (Iterative Deepening A^*), que es es una extensión del método de búsqueda iterativa en profundidad. Otra es el algoritmo SMA^* (Simplified Memory-Bounded A^*), este es muy similar al algoritmo A^*, pero se diferencia de él en que restringe el tamaño de $ABIERTA$ a un valor máximo prefijado. Por último veremos la *Búsqueda en Frontera*; este método solamente almacena un número muy pequeño de estados ya visitados.

Algoritmo IDA^*

Este algoritmo fue propuesto por R. Korf ([40]), con el objetivo de reducir la memoria requerida por el algoritmo A^*.

Inicialmente establece una longitud límite (coste de un camino) para la búsqueda igual al valor de $f(inicial)$ que siendo el heurístico admisible es una cota inferior de la solución óptima. En cada iteración el algoritmo IDA^* emplea una estrategia en profundidad para descartar los nodos con valor de $f(n)$ superior al límite. Si en una iteración no se encuentra solución, se realiza una nueva iteración comenzando la búsqueda otra vez desde el principio, pero empleando como nueva longitud límite el menor valor de f de los nodos descartados en la iteración anterior. La información del heurístico h se puede utilizar para ordenar los sucesores del nodo expandido antes de ser insertados

en $ABIERTA$, así los sucesores más prometedores de cada nodo expandido serán considerados antes. Esta ordenación puede reducir el número de nodos expandidos en una iteración cuando en ella se encuentra solución. Sin embargo, cuando una iteración no encuentra solución el número de nodos expandidos es el mismo que si se hubiese aplicado otra ordenación.

El algoritmo IDA^* es admisible cuando emplea heurísticos admisibles. Además, al realizar una búsqueda en profundidad tiene un consumo de memoria proporcional a la profundidad de la solución y al factor de ramificación. Sin embargo, en cada iteración tiene una búsqueda con tiempo exponencial en la profundidad límite. La peor situación para el algoritmo IDA^* es que todos los nodos tengan valores de f distintos, cuando esto ocurre en cada iteración sólo se añade un nodo al espacio de búsqueda y en este caso, si el algoritmo A^* necesita expandir N nodos para llegar a una solución, el algoritmo IDA^* necesita expandir $1 + 2 + \cdots + N = O(N^2)$ nodos lo cual aunque no supone espacio sí mucho tiempo. Este problema podría atajarse añadiendo en cada iteración a la longitud límite un factor ε-admisible. El inconveniente de esto es que cuando el valor de de f solo cambia cada cierta profundidad, se require para cada iteración mucho más tiempo que para la iteración anterior, por lo que el tiempo total de la búsqueda es prácticamente el tiempo de la última iteración realizada, tiempo similar al que emplearía el algoritmo A^*.por lo tanto similar al tiempo de A*.

Algoritmo SMA^*

El algoritmo IDA^* sólo recuerda de una iteración a la siguiente el menor valor de f de los nodos descartados, por lo que tiene que reexpandir muchos nodos. El algoritmo SMA^* propuesto por S. Russell ([72]) resuelve este problema utilizando también una cantidad limitada de memoria. Es un algoritmo sofisticado, para conocer sus detalles, propiedades y ver algún ejemplo se pueden consultar los textos de Russell y Norvig ([72, 73]).

En el algoritmo SMA^*, el límite se es el número máximo de nodos que se pueden almacenar en la $TABLA\_A$, y por lo tanto la profundidad máxima de un camino registrado en esta tabla. La idea general del funcionamiento del algoritmo SMA^* es la siguiente. Si necesita expandir un nodo y no tiene espacio en la $TABLA\_A$, elimina un nodo de esta tabla y de $ABIERTA$. A estos nodos se les suele denominar nodos olvidados y son los que tienen un mayor valor de f en $ABIERTA$, es decir los menos prometedores. El algoritmo recuerda en cada nodo la mejor f de los hijos olvidados de ese nodo. Además, para evitar reexplorar subárboles ya eliminados de memoria, el algoritmo SMA^* mantiene en los nodos antepasados información sobre la calidad del mejor camino en cada subárbol descartado. Así, sólo reexplora un subárbol descartado cuando el resto de posibilidades es peor, según las estimaciones. Para hacer esto, una vez descartados todos los sucesores de un nodo n, el algoritmo recuerda el coste del mejor camino a través de n.

Las principales propiedades del algoritmo SMA^* son las siguientes:

- Evoluciona con la memoria disponible.
- Evita estados repetidos, siempre que la memoria disponible se lo permita.

- Es completo si hay memoria disponible suficiente para almacenar el camino a la solución menos profunda.
- Es admisible si hay suficiente memoria para almacenar el camino hasta la solución óptima menos profunda. En otro caso, devuelve la mejor solución que se puede alcanzar con la memoria disponible.
- Si la memoria es suficiente para el árbol de búsqueda completo, la eficiencia de la búsqueda es óptima.

A pesar de estas buenas propiedades, no queda claro si el algoritmo SMA^* es mejor o igual de eficiente que cualquier otro algoritmo, empleando la misma información heurística y la misma cantidad de memoria.

3.6.1. Búsqueda en Frontera

Los algoritmos de búsqueda en grafos emplean la tabla $ABIERTA$ para almacenar los nodos frontera que han sido generados pero aún no han sido expandidos, y la tabla $CERRADA$ para almacenar aquellos que ya han sido expandidos. Almacenar todos los nodos generados tiene dos propósitos. En primer lugar poder reconocer aquellos estados que ya han aparecido por un camino si aparecen por otro, es decir prevenir la generación de nodos duplicados que representan el mismo estado, es decir permiten *detectar duplicados*. En segundo lugar, permitir reconstruir el camino solución tras finalizar la búsqueda empleando el método tradicional de *vuelta atrás*, cada nodo guarda un puntero a su padre a través del mejor camino, recuperando, por tanto, la solución recorriendo esos punteros desde el nodo objetivo hasta el inicial.

La memoria necesaria para almacenar todos estos nodos es precisamente el mayor problema de estas estrategias de búsqueda. Aunque algunos algoritmos de búsqueda en profundidad pueden alcanzar la solución óptima sin almacenar todos los nodos generados, son incapaces de identificar duplicados, lo que normalmente se traduce en un incremento exponencial de la complejidad temporal que puede hacerlos inaceptablemente lentos en la práctica. Korf en ([42]) propuso una estrategia de ahorro de memoria que no renuncia a detectar duplicados, y la aplicó a diferentes estrategias de búsqueda en grafos: con el A^* [45], con los algoritmos de Dijkstra y A^* bidireccional [42], con la búsqueda primero en anchura no informada [44]. Posteriormente autores como Zhou y Hansen propusieron algunas mejoras a esta estrategia [96, 97].

La estrategia se basa en la sospecha de que para detectar duplicados no es necesario almacenar en $CERRADA$ todos los nodos expandidos. Frecuentemente, para detectar duplicados sólo es necesario almacenar nodos que están cerca o en la frontera de la búsqueda. La intuición es que el conjunto de nodos generados forma un volumen que abarca al nodo inicial y crece hasta el exterior a medida que la frontera de la búsqueda es expandida. Por tanto si el grafo tiene una estructura particular, o la estrategia de búsqueda emplea ciertas técnicas, los nodos en la frontera no pueden regenerar nodos en el interior del volumen de la búsqueda. Por tanto no es necesario almacenar en memoria nodos interiores para poder detectar duplicados.

Si borramos de memoria los nodos en el interior del volumen de la búsqueda (nodos de $CERRADA$), no podremos recuperar el camino solución por el método tradicional de la vuelta atrás. Por tanto los algoritmos de búsqueda que emplean esta técnica para el ahorro de memoria, dependen de una técnica divide y vencerás para recuperar la solución. Cada nodo n almacena información sobre un nodo intermedio a través del mejor camino desde n al inicial. Podría ser un nodo en la mitad del espacio de búsqueda, pero no tiene porqué ser así. Para estimar si un nodo está en el medio de la búsqueda o en otra posición se puede hacer de varios maneras, por ejemplo, su posición puede ser estimada comparando su valor de g con su valor de f, o comparando su profundidad con una estimación de la longitud de la solución. Una vez que el problema de búsqueda es resuelto, se emplea la información de este nodo intermedio para dividir el problema de búsqueda en dos subproblemas: el problema de encontrar un camino óptimo desde el nodo inicial al nodo intermedio, y el problema de encontrar un camino óptimo del nodo intermedio a un objetivo. Cada uno de estos problemas es resuelto por el mismo algoritmo de búsqueda con el fin de encontrar un nodo intermedio a lo largo de sus caminos óptimos. El proceso se repite recursivamente hasta que se alcanzan subproblemas primitivos (aquellos en los que que el camino óptimo consiste en un sólo arco), y en consecuencia todos los nodos del camino solución para el problema original están identificados. El tiempo empleado para resolver todos esos subproblemas es normalemente muy corto comparado con el tiempo que lleva resolver el problema original. Esta estrategia produce un ahorro sustancial de memoria a cambio de invertir un poco de tiempo en recuperar la solución. Los algoritmos que emplean esta estrategia de ahorro de memoria difieren en pequeños detalles en la detección de duplicados y en la recuperación del camino solución.

Por ejemplo, Korf en [42, 45, 46] no almacena ningún nodo en $CERRADA$ y evita la duplicación de estados almacenando en cada uno de los nodos en $ABIERTA$ una lista de bits que representan los operadores utilizados para llegar a cada uno de los sucesores. De este modo cada vez que se expande un nodo, sólo se emplean los operadores que aún no han sido utilizados para generar sucesores. Sin embargo, Zhou y Hansen en [96] mantienen la tabla $CERRADA$ y para controlar la duplicidad de estados, guardan en cada nodo un contador (*contador predecesor*) que se inicializado al número de predecesores de este nodo en el grafo implícito. Cada vez que se expande un nodo, el contador de cada uno de sus sucesores se decrementa en uno. Los nodos son eliminados de $CERRADA$ cuando su contador está a 0 lo que asegura que no pueden ser regenerados. Entre otras, la principal ventaja de esta estrategia es que permite emplear cotas superiores del coste de la solución óptima para podar los nodos en $ABIERTA$, algo que no se puede hacer en la propuesta planteada por Korf, ya que de hacerlo se perderían los operadores empleados por los nodos podados, lo que podría provocar la regeneración del mismo nodo. Otra de las ventajas de esta estrategia frente a la de Korf, es que en problemas con un factor de ramificación alto la memoria consumida por el contador es menor que la consumida por la lista de operadores. Sin embargo, la ventaja de la estrategia de Korf es el ahorro de memoria, ya que no necesita almacenar nodos en $CERRADA$.

Korf en [45, 46] emplea una estrategia divide y vencerás para reconstruir la solución. En ella cada nodo llega hasta la mitad del espacio de búsqueda (mediante propagación desde su padre), almacenando toda la información sobre un nodo, a mitad de camino entre el nodo inicial y los objetivos, a lo largo del mejor camino hasta él. Después de que un nodo objetivo es expandido,

se emplea el nodo identificado a mitad de camino para reconstruir la solución con una estrategia divide y vencerás. Zhou y Hansen en [96], emplean una técnica diferente para recuperar la solución a partir de un nodo intermedio a través del mejor camino. Cada nodo guarda un puntero al un nodo intermedio, denominado nodo *relevo* y almacenado en memoria. Aunque de este modo el número de nodos en memoria se incremente, la memoria global puede ser menor ya que la estructura de los nodos es menor al sólo almacenar un puntero y no la información completa de los nodos intermedios. De hecho la estructura del nodo es la de los nodos de $ABIERTA$, pudiendo ser utilizado el puntero para apuntar al padre del nodo o al nodo relevo. Si hay suficiente memoria disponible para resolver un problema o subproblema empleando A^*, esto evita emplear divide y vencerás para reconstruir la solución manteniendo todos los nodos en memoria, dejando que cada nodo guarde un puntero al padre y empleando la vuelta atrás para recuperar la solución. Esta técnica empleada en la búsqueda en frontera propuesta por Korf tiene también un buen comportamiento.

Hasta ahora hemos visto dos versiones de la búsqueda en frontera, la propuesta por Korf y la propuesta por Zhou y Hansen. En [98], Zhou y Hansen comparan la búsqueda en profundidad y en anchura utilizando distintas versiones de los algoritmos de búsqueda en grafos A^* y ramificación y poda, que emplean búsqueda en frontera. Para ello los aplican a la resolución de los problemas 15-puzzle, torres de hanoi y planificación tipo strips. Los resultados mostraron que cuando se adopta la técnica de búsqueda en frontera combinada con divide y vencerás para reconstruir la solución la búsqueda en anchura tiene algunas ventajas sobre la búsqueda en profundidad: la frontera en anchura es menor que en profundidad por lo que se requiere menos memoria para prevenir la regeneración de los nodos en $CERRADA$, además en la búsqueda en anchura para grafos de coste uniforme se tiene la garantía de que el coste f de un nodo es óptimo desde el momento en que se genera el nodo, lo que hace que la búsqueda sólo necesite un heurístico admisible y no necesariamente consistente.

3.7. Conclusiones

En este capítulo hemos visto la búsqueda primero el mejor como base para describir el algoritmo A^*, estrategia de búsqueda heurística empleada en esta tesis. El algoritmo A^* es una especialización del algoritmo general de búsqueda en grafos ($AGBG$) en su versión BF ("best-first"), en la que la función de evaluación f es una estimación del coste del camino solución condicionado a pasar por el nodo a evaluar. Una vez descrito el algoritmo, hemos pasado a enunciar sus principales propiedades en cuanto a terminación y completud, admisibilidad y monotonía. Además, hemos visto la influencia que el heurístico tiene en las propiedades del algoritmo, y hemos descrito la técnica de diseño de heurísticos basada en la relajación de las restricciones del problema.

El algoritmo A^*, es una estrategia que emplea mucho tiempo y memoria en alcanzar una solución óptima. Por ello, cada vez se plantean problemas más complejos y de mayor tamaño se requiere la aplicación de nuevos métodos de búsqueda en los que el conocimiento juega un papel fundamental para llegar a soluciones satisfactorias, no necesariamente óptimas, en poco tiempo. En estos casos, se puede relajar la optimalidad dándole pesos a los dos sumandos de la función de evaluación g y h. Concretamente, hemos visto tres técnicas que permiten realizar esto, las ponderaciones estática y dinámica, así como el algoritmo Aε^*. Por otro lado, hemos visto dos variantes clásicas del algoritmo

A^* en sus versiones de búsqueda con memoria limitada (IDA^* y SMA^*), así como la reciente variante de búsqueda en frontera.

Actualmente existen en la literatura otras propuestas que han demostrado ser eficientes en muchos problemas. Por ejemplo los métodos de búsqueda con discrepancia limitada (LDS) que se basan en considerar que el heurístico hace una buena elección un porcentaje alto de veces. Por tanto, realizan una búsqueda visitando las ramas del árbol de menor a mayor discrepancia con el heurístico, es decir, primero la rama del árbol correspondiente a la primera elección del heurístico, rama sin discrepancias heurísticas; a continuación consideran las ramas con una sola discrepancia, y así sucesivamente. Algunas de las propuestas más interesantes de este tipo de búsqueda se pueden ver en el artículo de P. Meseguer y T. Walsh ([52]).

Recientemente ha aparecido el término metaheurística que hace referencia a técnicas generales de diseño de algoritmos heurísticos. Los métodos de búsqueda en espacios de estados, la búsqueda local y la computación evolutiva son un caso particular de metaheurísticas. No obstante, existen otras que no se han tratado en esta tesis, pero también muy interesantes, como son los algoritmos GRASP o la búsqueda dispersa. Para tener una visión global sobre estas técnicas se puede consultar la monografía publicada en el número 19 de la Revista Iberoamericana de Inteligencia Artificial, del año 2003, editada por la Asociación Española para la Inteligencia Artificial ($AEPIA$). Para una visión más profunda de estas técnicas se puede recurrir al texto de Z. Michalewicz y D. B. Fogel ([53]), en él se tratan también métodos de resolución de problemas basados en técnicas de programación lineal.

Capítulo 4

JOB SHOP SCHEDULING CON MINIMIZACIÓN DEL MAKESPAN

4.1. Introducción

En este capítulo se aborda el problema Job Shop Scheduling (JSS) con minimización del makespan mediante búsqueda heurística tipo primero el mejor. Esta versión del problema se denota como $J||C_{max}$ y es sin duda la versión más estudiada en la literatura, por lo que se pueden encontrar referencias a propuestas que resuelven el problema con diferentes métodos, la mayoría de ellos basados en el *camino crítico*. Entre los métodos exactos, debemos destacar el más relevante de ellos que es el algoritmo de *ramificación y poda* propuesto por Brucker y colaboradores en [12, 11]. Este algoritmo fue desarrollado a partir de conceptos y técnicas propuestas por otros autores como Carlier y Pinson en [16, 17]. En lo que respecta a los métodos no exactos, los más relevantes son las técnicas de *búsqueda local* basadas en en las estructuras de vecindad propuestas inicialmente por P. Van Laarhoven, E. Aarts y K. Lenstra en [89] y que posteriormente emplearon y desarrollaron otros muchos autores, a menudo en combinación con una o varias meta-heurísticas tales como los *algoritmos genéticos* ([51, 60, 94]), la *búsqueda tabú* [24] o el *temple simulado*([95]).

Para resolver el problema mediante búsqueda heurística en espacios de estados hemos elegido el algoritmo A^* y el espacio de las planificaciones activas. Una de las razones principales para elegir este espacio, es que es dominante, es decir contiene al menos una solución óptima, para las funciones objetivo regulares, como por ejemplo el makespan. La principal contribución de este capítulo es la formalización y aplicación en este contexto de un método de poda por dominancia que permite reducir de manera muy significativa tanto el espacio de búsqueda efectivo como el tiempo de ejecución.

Este capítulo está organizado del siguiente modo. Comenzamos con la formulación del problema en la sección 4.2. En la sección 4.3 se describe el espacio de búsqueda de las planificaciones activas para el problema JSS. La sección 4.4 resume las principales características del algoritmo A^*. En la sección 4.5 se describen las estrategias heurísticas empleadas para guiar la búsqueda del algoritmo

A^* en la resolución del problema JSS. La sección 4.6 es la parte central del capítulo; en ella se introduce el concepto de poda por dominancia y se describen las reglas que permiten realizar la poda por dominancia para el problema JSS con minimización del makespan. La sección 4.7 incluye un estudio experimental en el que se evalúan distintas estrategias de aplicación del método de poda. Por último, en la sección 4.8 se resumen las principales conclusiones del capítulo. Al final del capítulo se incluyen también dos anexos. El primero de ellos contiene un resumen de la notación utilizada en el capítulo, y el segundo recoge las tablas con todos los resultados obtenidos en el estudio experimental.

4.2. Formulación del Problema $J||C_{max}$

El problema Job Shop Scheduling conocido como $J||C_{max}$ en la literatura, consiste en planificar un conjunto de N trabajos $\{J_1, \ldots, J_N\}$ sobre un conjunto de M recursos o máquinas físicas $\{R_1, \ldots, R_M\}$. Cada trabajo J_i consta de un conjunto de tareas u operaciones $\{\theta_{i1}, \ldots, \theta_{iM}\}$ que han de ser planificadas secuencialmente. Cada tarea θ_{il} requiere una única máquina $R_{\theta il}$, tiene una duración fijada $p_{\theta il}$ y un tiempo de inicio $st_{\theta il}$ a determinar. El problema tiene tres restricciones: precedencia, capacidad y no-expulsión. Las restricciones de precedencia o secuenciales se traducen en desigualdades lineales del tipo $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$. Las restricciones de capacidad se traducen en restricciones disyuntivas de la forma $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$ si $R_v = R_w$. Las restricciones de no-expulsión requieren que un recurso o máquina sea asignado a una tarea sin interrupción durante todo su tiempo de procesamiento. El objetivo es encontrar una planificación factible tal que el tiempo de completud (tiempo de fin de la última tarea), es decir el makespan, sea mínimo.

4.2.1. Representación del Problema

Como se ha visto en la sección 2.2.1 las instancias de este problema se suelen representar por un grafo dirigido $G = (V, A \cup E)$. Cada nodo en el conjunto V representa una tarea, con la excepción de los nodos ficticios $inicio(O)$ y $fin(*)$, los cuales representan tareas con tiempo de procesamiento 0. Los arcos de A se denominan arcos *conjuntivos* y representan a las restricciones de precedencia, y los arcos de E, denominados arcos *disyuntivos*, representan a las restricciones de capacidad. El conjunto E es la unión disjunta de los subconjuntos E_i con $E = \cup_{\{i=1,\ldots,M\}} E_i$, donde E_i incluye un $arco(v, w)$ por cada par de tareas v y w que requieren el recurso R_i. El peso de los arcos es el tiempo de procesamiento de la tarea del nodo origen. El nodo *inicio* esta conectado con la primera tarea de cada trabajo (en este caso los pesos corresponden a los tiempos de inicio mínimos de los trabajos) y las últimas tareas de cada trabajo están conectadas con el nodo *fin*.

Una planificación factible se representa como un subgrafo sin ciclos G_s de G, $G_s = (V, A \cup H)$, donde $H = \cup_{i=1,\ldots,M} H_i$, siendo H_i un orden de procesamiento para las tareas que requieren R_i. El makespan es el coste del *camino crítico*. Un camino crítico es la ruta más larga en G_s desde el nodo *inicial* hasta el nodo *final*.

Con el fin de simplificar las expresiones, utilizaremos la siguiente notación para una planificación factible. La *cabeza* r_v de una operación v es el coste del camino más largo desde el nodo *inicio* al nodo v, es decir es el valor de st_v. La *cola* q_v se define de modo que el valor $q_v + p_v$ es el coste del

camino más largo desde v al nodo *fin*. Por lo tanto, $r_v + p_v + q_v$ es el makespan si v está en el camino crítico, en otro caso, es una cota inferior. PM_v y SM_v denotan, respectivamente, al predecesor y sucesor de la tarea v en la secuencia de su máquina; así mismo PJ_v y SJ_v denotan, respectivamente, al predecesor y sucesor de la tarea v en su trabajo.

Una planificación parcial se representa por un subgrafo de G donde algunos de los arcos disyuntivos aún no han sido fijados. En tal planificación, las cabezas y las colas se definen del siguiente modo

$$\begin{aligned} r_v &= \text{máx}\{\text{máx}_{w \in P(v)}(r_w + p_w), r_{PJ_v} + p_{PJ_v}\} \\ q_v &= \text{máx}\{\text{máx}_{w \in S(v)}(p_w + q_w), p_{SJ_v} + q_{SJ_v}\} \end{aligned} \tag{4.1}$$

donde $P(v)$ denota a los predecesores disyuntivos de v, es decir aquellas operaciones que requieren la máquina R_v y están planificadas antes que v. Del mismo modo, $S(v)$ denota a los sucesores disyuntivos de v. Por lo tanto, el valor $r_v + p_v + q_v$ es una cota inferior del coste de la mejor solución que puede ser alcanzada desde esa planificación parcial.

La Figura 4.1 muestra el grafo dirigido que representa un problema con 2 trabajos y 3 máquinas, así como un subgrafo solución para dicho problema (en este los arcos en negrita representan un camino crítico de coste 7).

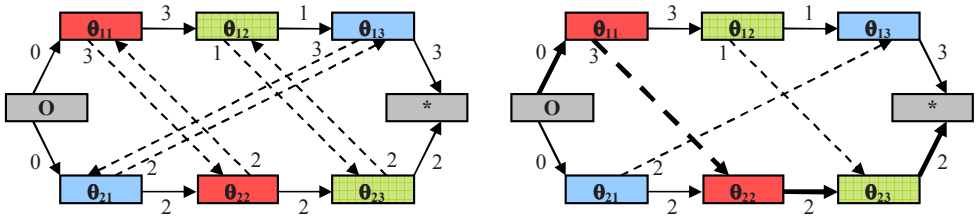


Figura 4.1: Grafos de un problema y una solución

4.3. El Espacio de las Planificaciones Activas

Como se ha indicado anteriormente, consideraremos el espacio de búsqueda de las planificaciones activas. Recordemos que una planificación es *activa* si para que una tarea comience antes, al menos otra debe ser retrasada.

Como algoritmo de expansión para obtener los sucesores de un nodo utilizaremos una extensión del algoritmo $G\&T$ propuesto en [31], y que se ha descrito en la sección 2.4.1. Se trata de un algoritmo voraz que produce planificaciones activas en $N * M$ pasos. En cada paso el algoritmo $G\&T$ realiza una elección no determinista. Por lo tanto, considerando todas las elecciones posibles se tiene un árbol de búsqueda completo, apropiado para ser recorrido por estrategias tales como *ramificación y poda*, *backtracking* o A^*. Este es uno de los esquemas de ramificación más usuales para el problema JSS, como se puede ver en [13], y es la aproximación empleada, por ejemplo, en

[91] y [77, 76].

El Algoritmo 4.1 muestra la operación de expansión que genera el conjunto completo de planificaciones activas cuando es aplicado sucesivamente desde el estado inicial. El estado inicial se corresponde con la situación en la que ninguna tarea del problema está planificada. Mientras que en el resto de estados ya están planificadas un subconjunto de tareas. Como se ha visto en el capítulo 2, el espacio de estados generado de este modo, considerando un estado como un conjunto de tareas planificadas junto con sus tiempos de inicio, es un árbol.

A continuación se describe la notación utilizada en la descripción del algoritmo. O denota el conjunto de tareas u operaciones de que consta un problema, y n_1 y n_2 son dos estados del espacio de búsqueda. En n_1, O se puede obtener como la unión disjunta $SC(n_1) \cup US(n_1)$, donde $SC(n_1)$ es el conjunto de tareas planificadas en n_1 y $US(n_1)$ el conjunto de tareas no planificadas en ese mismo estado. $D(n_1) = |SC(n_1)|$ es la profundidad, en el espacio de búsqueda, del nodo n_1. Dado $O' \subseteq O$, $\boldsymbol{r}_{n_1}(O')$ es el vector de las cabezas de las tareas de O' en el estado n_1. $\boldsymbol{r}_{n_1}(O') \leq \boldsymbol{r}_{n_2}(O')$ si y sólo si para cada tarea $v \in O'$, $r_v(n_1) \leq r_v(n_2)$, siendo $r_v(n_1)$ y $r_v(n_2)$ las cabezas de la tarea v en los estados n_1 y n_2 respectivamente. Análogamente, $\boldsymbol{q}_{n_1}(O')$ es el vector de las colas de las tareas de O' en el estado n_1. El coste de pasar de un estado n a otro n', $c(n, n')$, se obtiene del valor máximo entre cero y la diferencia $r_w + p_w - C_{max}(n)$, donde $r_w + p_w$ es el tiempo de finalización de la tarea w planificada al pasar de n a n' y $C_{max}(n)$ el tiempo máximo de fin de las tareas de $SC(n)$.

4.4. Búsqueda Primero el Mejor

Para la búsqueda primero el mejor hemos elegido el algoritmo A^* propuesto por Nilsson ([56]), descrito en el capítulo 3. El algoritmo A^* parte de un nodo o estado inicial s y trata de alcanzar un nodo del conjunto de estados objetivos Γ. Para ello utiliza un operador de transición SUC (Algoritmo 4.1) tal que para cada nodo n del espacio de búsqueda, $SUC(n)$ retorna el conjunto de nodos sucesores del estado n. La transición o paso de un nodo n a cada uno de sus sucesores n' tiene asociado un coste positivo $c(n, n')$. $P^*_{s\text{-}n}$ denota al camino de coste mínimo desde el nodo s al nodo n. El algoritmo busca el camino óptimo desde el inicial a los objetivos, es decir $P^*_{s\text{-}o}$ con

Algoritmo 4.1 SUC(estado n). Problema $J||C_{max}$: Algoritmo para expandir el estado n

Requiere: Estado n del problema JSS
Produce: Estados sucesores del estado n

1. $A = \{v \in US(n);\ PJ_v \in SC(n)\}$;
2. $v = \arg\min\{r_u + p_u;\ u \in A\}$;
3. $B = \{w \in A;\ R_w = R_v \text{ y } r_w < r_v + p_v\}$;

para cada $w \in B$ **hacer**

4. $SC(n') = SC(n) \cup \{w\}$ y $US(n') = US(n) \backslash \{w\}$;
5. $G_{n'} = G_n \cup \{w \rightarrow v;\ v \in US(n'),\ R_v = R_w\}$;
   {Se planifica w en el estado n' con un tiempo de inicio r_w}
6. $c(n, n') = \max(0, r_w + p_w - C_{max}(n))$;
7. Añadir n' a los sucesores;

fin para

8. Retornar sucesores;

$o = \arg\min\{P^*_{s\text{-}o'} | o' \in \Gamma\}$.

El conjunto de nodos candidatos a ser expandidos se mantiene en una lista ordenada denominada $ABIERTA$. El siguiente nodo a ser expandido es aquel con el menor valor de la función de evaluación f, definida como $f(n) = g(n) + h(n)$; donde $g(n)$ es el coste del mejor camino conocido hasta el momento desde s a n y $h(n)$ es una estimación heurística positiva de la distancia mínima de n al objetivo más próximo.

El algoritmo A^* tiene una serie de propiedades importantes que dependen de la función h [59, 55, 57, 58, 61]. Siempre que la función heurística subestime el mínimo coste real, $h^*(n)$, de n a los objetivos, es decir $h(n) \leq h^*(n)$, para cada nodo n, el algoritmo es admisible, lo que se traduce en que encuentra una solución óptima, siempre y cuando los recursos computacionales sean suficientes. Además, si $h(n_1) \leq h(n_2) + c(n_1, n_2)$ para cada par de nodos n_1, n_2 del grafo de búsqueda, h es consistente. Dos de las propiedades más importantes de los heurísticos consistentes son la admisibilidad y el hecho de que la secuencia de valores $f(n)$ de los nodos expandidos es no decreciente. La función heurística $h(n)$ representa el conocimiento del que se dispone sobre el dominio del problema, por tanto cuanto más se aproxime a h^*, el algoritmo será más eficiente ya que en general necesitará expandir un número menor de nodos para alcanzar una solución óptima.

4.5. Estrategias Heurísticas para el Problema $J||C_{max}$

En esta sección se describen los heurísticos utilizados para guiar la búsqueda. Todos ellos se obtienen por relajación de alguna de las restricciones del problema. El problema residual correspondiente a un estado n viene dado por las tareas no planificadas en n junto con sus cabezas, duraciones y colas; este problema lo denotaremos por

$$P(n) = (US(n), \mathbf{r}_n(US(n)), \mathbf{p}_n(US(n)), \mathbf{q}_n(US(n))). \tag{4.2}$$

En general, en un estado n hay un número de trabajos con todas sus tareas planificadas y otros con tareas aún sin planificar, estos subconjuntos de J los denotamos respectivamente como

$$\begin{aligned} J_{US}(n) &= \{J_i \in J; \exists l, 1 \leq l \leq M, \theta_{il} \in US(n)\}, \\ J_{SC}(n) &= J \backslash J_{US}(n). \end{aligned} \tag{4.3}$$

Asimismo, denotaremos por $C_{max}(J_{SC}(n))$ el tiempo de completud máximo de los trabajos en $J_{SC}(n)$, es decir

$$C_{max}(J_{SC}(n)) = \max\{r_{\theta_{iM}} + p_{\theta_{iM}}, J_i \in J_{SC}(n)\}. \tag{4.4}$$

con $C_{max}(J_{SC}(n)) = 0$ si $J_{SC}(n) = \varnothing$. Análogamente, $C_{max}(P(n))$ denotará el coste óptimo del problema $P(n)$, es decir el coste óptimo de planificar las tareas $US(n)$ teniendo en cuenta las cabezas y colas de estas tareas en el estado n. De este modo se tiene que

$$f^*(n) = \max(C_{max}(J_{SC}(n)), C_{max}(P(n))). \tag{4.5}$$

y

$$g(n) = \text{máx}(C_{max}(J_{SC}(n)), \underset{J_i \in J_{US}(n)}{\text{máx}}(\text{mín}\{r_{\theta_{ij}}, \theta_{ij} \in US(n)\})) = C_{max}(n). \tag{4.6}$$

Para ilustrar la forma en que se aplican los diferentes heurísticos, consideraremos el estado intermedio para el problema $FT06$ representado en la Figura 4.2 mediante un gráfico de Gantt. Como se puede observar en este estado hay 6 tareas planificadas, concretamente $SC(n) = \{\theta_{11}, \theta_{12}, \theta_{13}, \theta_{21}, \theta_{31}, \theta_{32}\}$, mientras que las 30 restantes están aún sin planificar. En este caso no hay ningún trabajo totalmente planificado, con lo que $J_{SC} = \varnothing$ y en consecuencia $C_{max}(J_{SC}(n)) = 0$. Además, $g(n) = 15$ de acuerdo con la expresión 4.6. El coste de la solución óptima para este problema es 55.

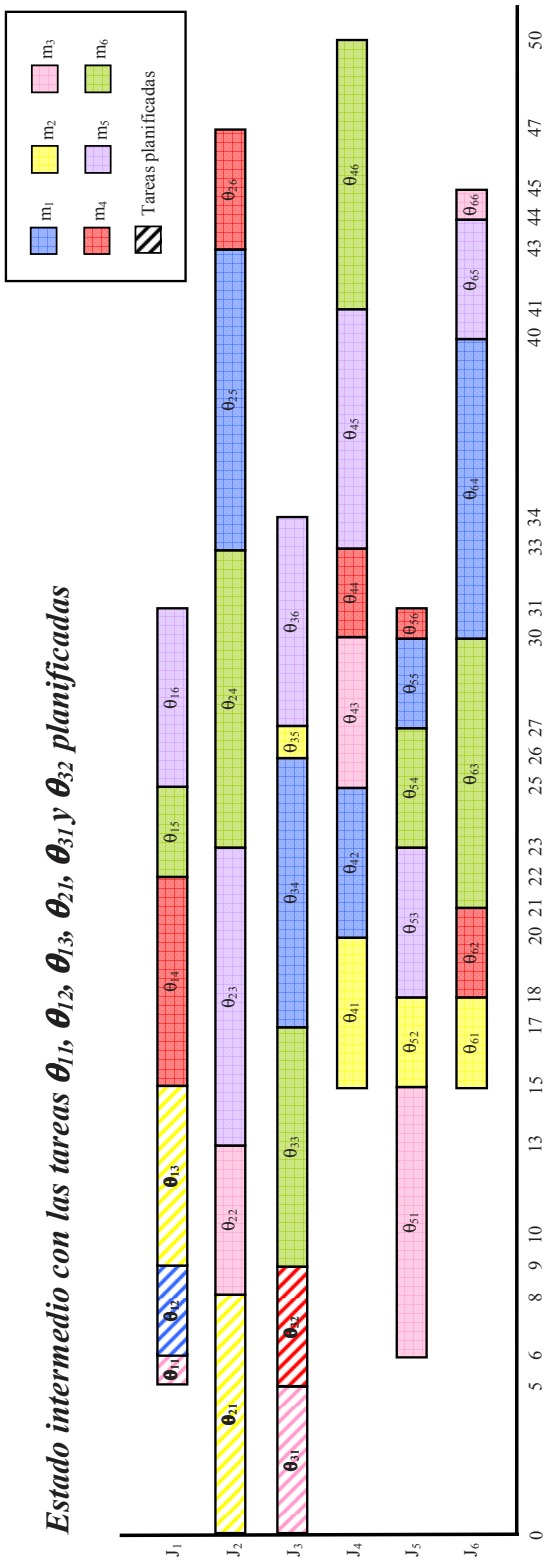


Figura 4.2: Gráfica de Gantt para un estado intermedio del problema $FT06$

4.5.1. Heurístico h_1

Esta estimación heurística se obtiene a partir de una sencilla simplificación del problema, resoluble en un tiempo polinomial, que consiste en relajar todas las restricciones de capacidad que afectan a las tareas sin planificar en un estado n.

La gráfica de la Figura 4.3 muestra la solución óptima del problema relajado correspondiente al estado en el que están planificadas las tareas $\{\theta_{11}, \theta_{12}, \theta_{13}, \theta_{21}, \theta_{31}, \theta_{32}\}$. En esta solución relajada las tareas se planifican en la forma que se indica en el propio gráfico. De modo que el coste óptimo del problema relajado, o lo que es lo mismo la estimación heurística del problema real, se calcula como $\text{máx}_{1 \leq i \leq N}(r_{\theta_{iM}} + p_{\theta_{iM}})$. De este modo se obtiene una cota inferior de $f^*(n)$. Por lo tanto, para obtener una cota inferior de $h^*(n)$ es necesario restar el valor de $g(n)$. Esta estimación heurística se calcula del siguiente modo:

$$\begin{aligned} h_1(n) &= F_1(n) - g(n); \\ F_1(n) &= \text{máx}_{1 \leq i \leq N}(r_{\theta_{iM}} + p_{\theta_{iM}}) \end{aligned} \tag{4.7}$$

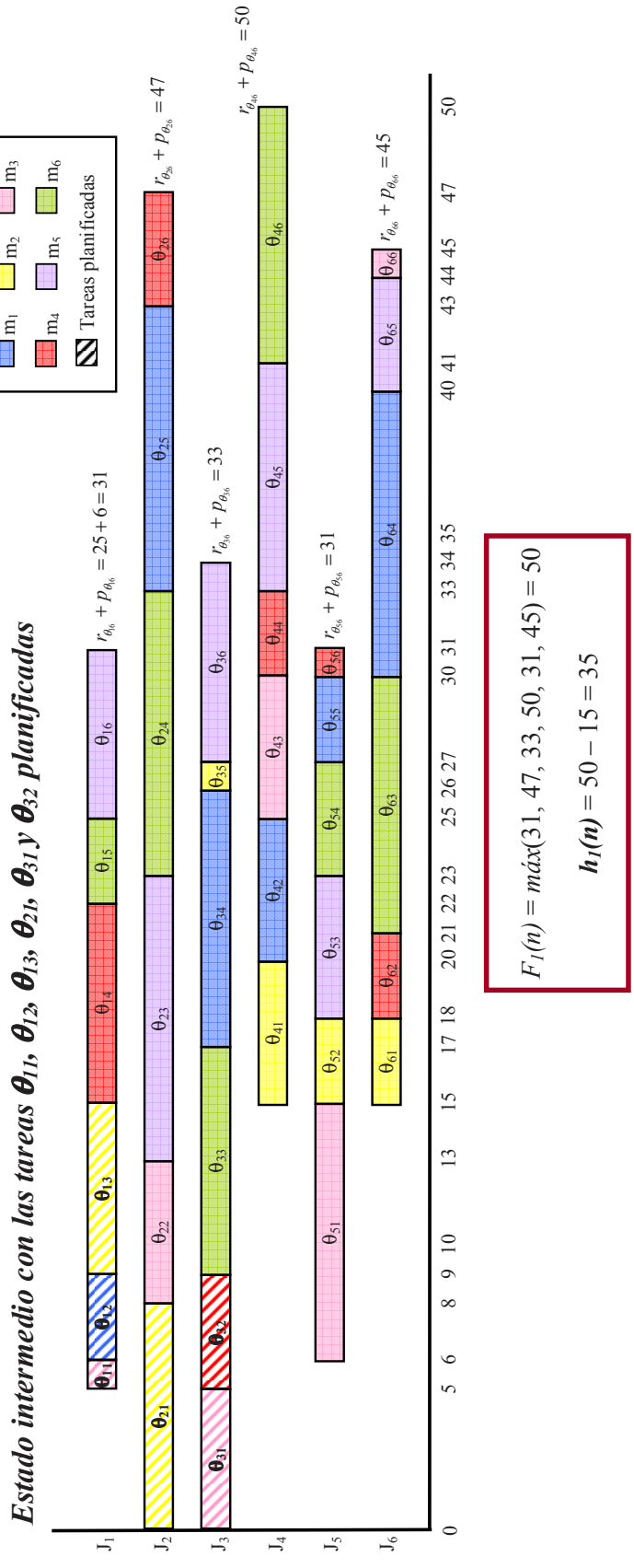


Figura 4.3: Valor del heurístico h_1 para un estado intermedio del problema $FT06$

4.5.2. Heurístico h_2

Este heurístico fue propuesto por E. Taillard en [84]. Para obtener el valor de este heurístico se realiza una relajación del problema en dos pasos. En primer lugar se relajan las restricciones de capacidad de todas las máquinas salvo una, la máquina m. El problema simplificado se denota como

$$P(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m),\ \mathbf{p}_n(US(n)|_m),\ \mathbf{q}_n(US(n)|_m)), \tag{4.8}$$

donde $US(n)|_m$ representa el conjunto de operaciones sin planificar en el estado n que requieren la máquina m. El problema $P(n)|_m$ se conoce en la literatura como el problema de secuenciamiento de una máquina (One Machine Sequencing o OMS) con cabezas y colas. En este problema cada operación v está definida por su cabeza r_v (cota inferior del tiempo de inicio de la tarea v), su tiempo de procesamiento p_v en la máquina m, y su cola q_v (cantidad de tiempo mínima que debe transcurrir después de terminar la tarea v). El problema $P(n)|_m$ es todavía *NP-duro*, por lo que se realiza una segunda relajación en la asignación de cabezas y colas para la máquina m. Esta relajación permite que a cada tarea de $US(n)|_m$ se le asignen la cabeza y la cola menores de todas las tareas. El coste óptimo del problema relajado resultante se calcula en tiempo polinomial sin más que añadir la menor de las cabezas y la menor de las colas a la suma de las duraciones de todas las tareas. Iterando para todas las máquinas y tomando el valor máximo de los costes de los problemas relajados, se obtiene una cota inferior de $f^*(n)$ debido al hecho de que las cabezas de las operaciones de $US(n)$ se calculan a partir de las operaciones planificadas en n, $SC(n)$. Por lo tanto, para obtener una cota inferior de $h^*(n)$, de forma análoga al heurístico anterior, se debe descontar el tiempo de completud mayor de las tareas de $SC(n)$, es decir $g(n)$. Así, el heurístico, que denominaremos h_2, se calcula del siguiente modo

$$\begin{gathered} h_2(n) = F_2(n) - g(n); \\ F_2(n) = \text{máx}(C_{max}(J_{SC}(n)), \text{máx}_{m\in R}\{r^m_{min}(n) + P^m_{sum}(n) + q^m_{min}(n)\}) \end{gathered} \tag{4.9}$$

donde

$$\begin{aligned} r^m_{min}(n) &= \arg\text{mín}\{r_v; v \in US(n)|_m\}; \\ q^m_{min}(n) &= \arg\text{mín}\{q_v; v \in US(n)|_m\}; \\ P^m_{sum}(n) &= \textstyle\sum_{v\in US(n)|_m} p_v; \end{aligned} \tag{4.10}$$

La Figura 4.4 muestra un el valor del heurístico para un estado intermedio del problema $FT06$.

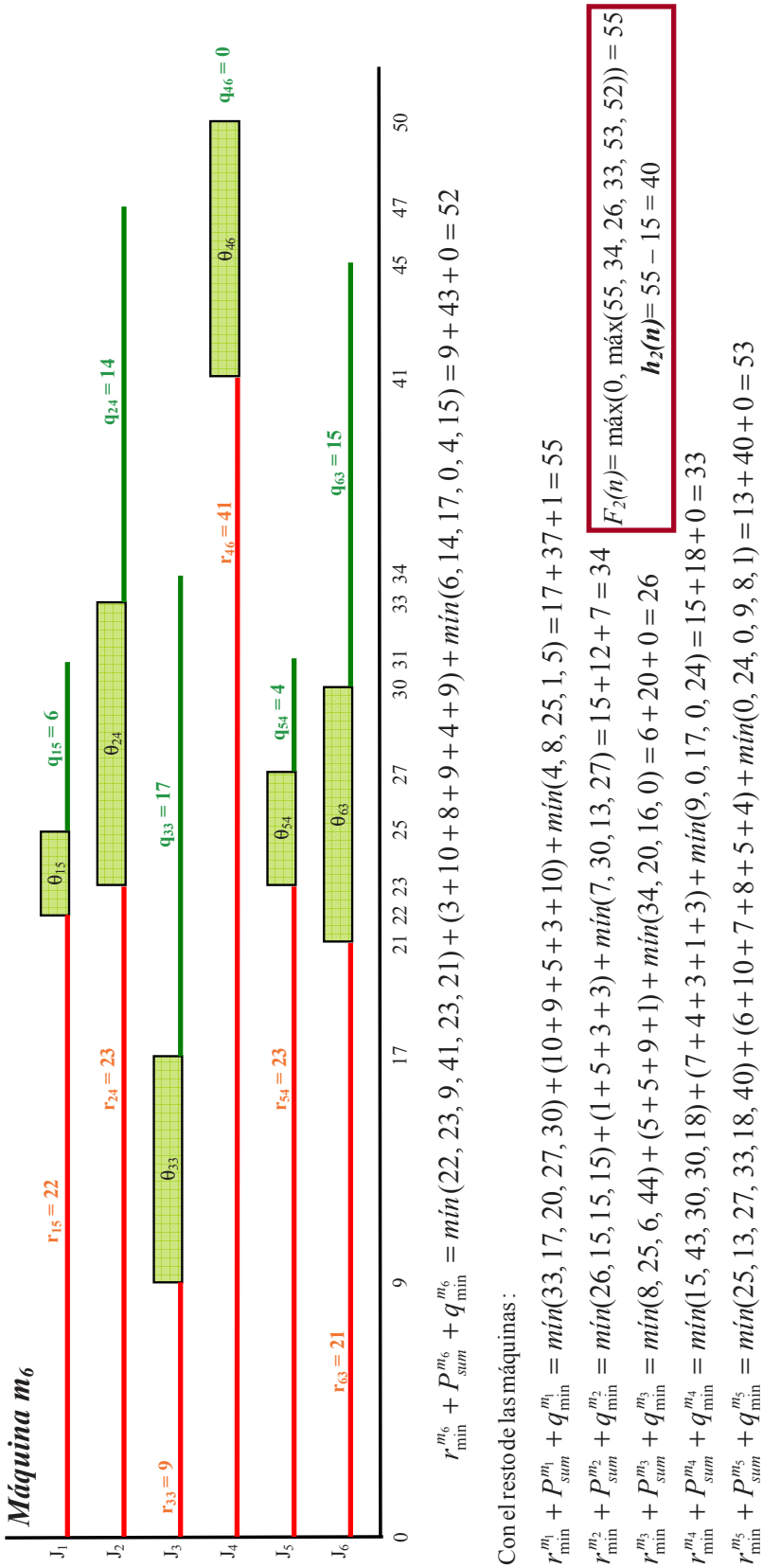


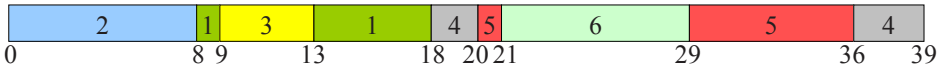
Figura 4.4: Cálculo del heurístico h_2 para un estado intermedio del problema $FT06$

4.5.3. Heurístico h_3

Para obtener una cota inferior del makespan en el estado intermedio n, este heurístico también realiza una relajación del problema en dos pasos. Como en el caso anterior, en primer lugar consideramos el problema simplificado $P(n)|_m$. Pero en este caso la segunda relajación consiste en eliminar la restricción de no expulsión de la máquina m. Una solución óptima al problema resultante de esta segunda relajación se obtiene construyendo una planificación denominada Jackson's Preemptive Schedule o JPS [16, 17]. La Figura 4.5 muestra un ejemplo de una instancia de un problema OMS y una planificación JPS para él.

v	1	2	3	4	5	6
r_v	4	0	9	15	20	21
p_v	6	8	4	5	8	8
q_v	20	25	30	9	14	16

a) Instancia de un problema OMS



b) JPS con makespan = 50, dado por el tiempo de completud del trabajo 5 (36 + 14)

Figura 4.5: a) Una instancia del problema OMS. b) Planificación JPS para la instancia del problema a)

La JPS se construye mediante el algoritmo de Schrage (sección 2.3.1). La forma en que este algoritmo construye la planificación JPS es relativamente sencilla: en cada instante de tiempo t, dado por la cabeza o el tiempo de fin de una tarea, desde la mínima cabeza r_v hasta que todos los trabajos están completamente planificados, se planifica la tarea que pueda comenzar en ese instante y que tenga la mayor cola. Carlier y Pinson probaron en [16, 17] que el cálculo de la planificación JPS tiene una complejidad de $O(k \times log_2(k))$, siendo k el número de tareas.

La planificación JPS del problema $P(n)|_m$ proporciona una cota inferior de $C_{max}(J_{US}(n))$ debido al hecho de que las cabezas de las operaciones de $US(n)|_m$ se ajustan a partir de las operaciones planificadas en n, $SC(n)$. Quedándonos con el mayor de esos valores de entre todas las máquinas y teniendo en cuenta el valor $C_{max}(J_{SC}(n))$, es decir el tiempo máximo de finalización de los trabajos con todas las tareas planificadas, obtenemos una cota inferior de $f^*(n)$. Por lo tanto, se debe descontar, como en los heurísticos anteriores, el valor de $g(n)$ para obtener una cota inferior de $h^*(n)$. Así pues calculamos el heurístico h_3 de la siguiente manera

$$\begin{aligned} h_3(n) &= F_3(n) - g(n); \\ F_3(n) &= \text{máx}(C_{max}(J_{SC}(n)), JPS(P(n))); \\ JPS(P(n)) &= \text{máx}_{m \in R}\, JPS(P(n)|_m) \end{aligned} \tag{4.11}$$

Como se ha indicado en la descripción del problema OMS, sección 2.3.1, los tiempos r_i y q_i tienen un comportamiento simétrico, por lo que el problema OMS *Dual* resulta también de interés. Debido

a esta simetría el algoritmo de Schrage permite calcular también la planificación *JPS Dual* ($DJPS$) con solo intercambiar cabezas por colas. Es fácil ver que la solución del problema primal (sin relajar la restricción de no expulsión) es también solución del problema dual. Sin embargo el coste del $DJPS$ puede ser diferente del coste del problema *JPS Primal* ($PJPS$). Experimentalmente hemos comprobado que a pesar de que ambas planificaciones JPS (primal y dual) pueden tener costes diferentes, en la práctica no se consigue una mejora del heurístico calculando las dos planificaciones. Por lo que hemos optado por aplicar el cálculo de la JPS solamente al problema *Primal*.

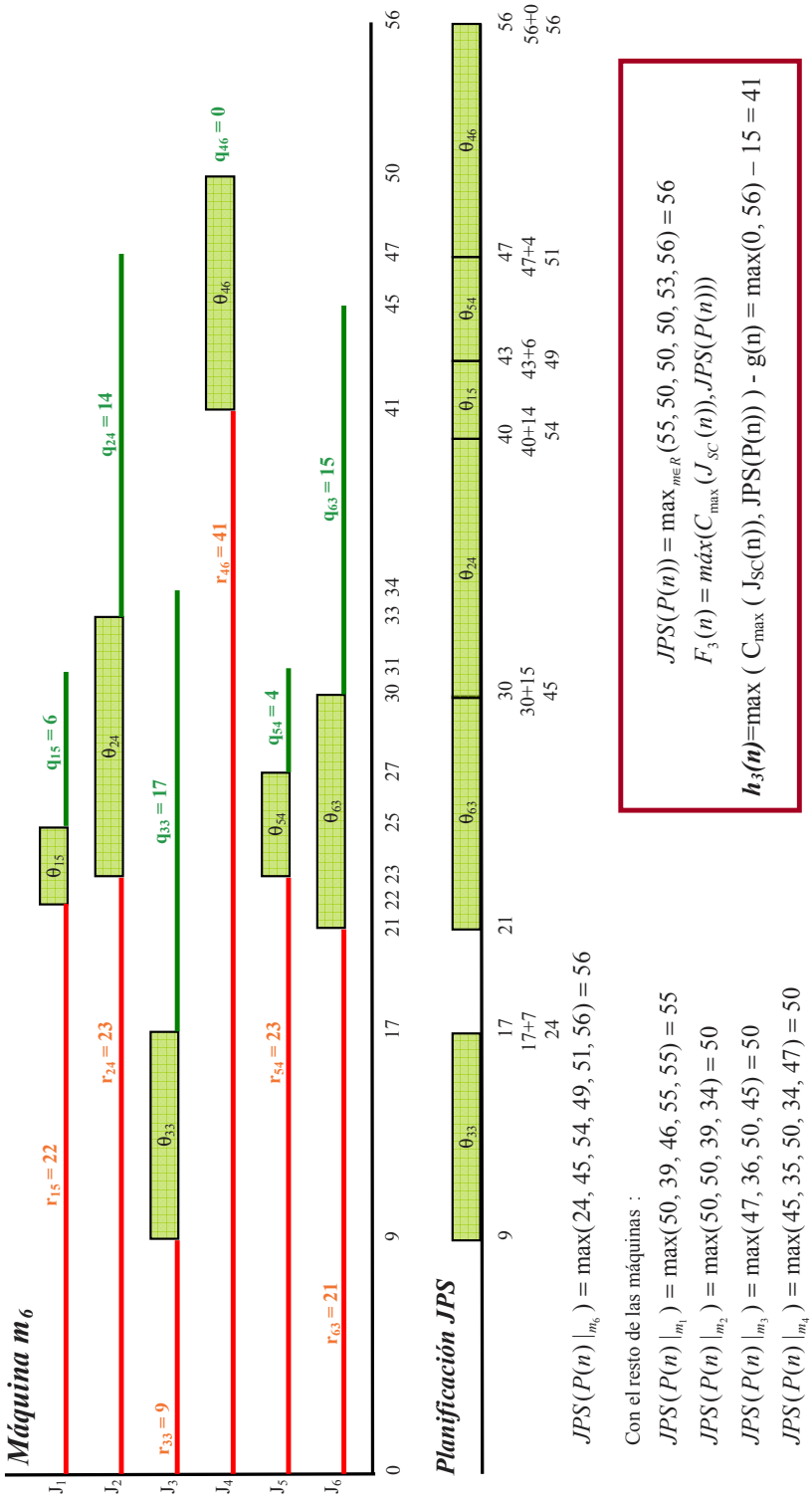


Figura 4.6: Valor del heurístico h_3 para un estado intermedio del problema FT06

4.6. Propiedades de Dominancia

En esta sección se formaliza un método de reducción del espacio de búsqueda efectivo basado en propiedades de dominancia entre estados. Normalmente este tipo de técnicas se implementan mediante reglas que permiten descartar alguno de los sucesores de un determinado estado. La idea general es que se puede comprobar que la mejor solución alcanzable desde ese estado no es mejor que la mejor solución alcanzable desde alguno de los demás estados sucesores del mismo nodo.

El método que proponemos aquí es más general, ya que permite establecer relaciones de dominancia entre estados que no son sucesores del mismo nodo. Este hecho complica la aplicación del método, pero a la vez ofrece la posibilidad de hacer reducciones muy significativas en el espacio de búsqueda como veremos a continuación.

Dados dos estados n_1 y n_2, decimos que n_1 domina a n_2 si y sólo si la mejor solución que se puede alcanzar desde el estado n_1 es mejor, o al menos igual, que la mejor solución que se puede alcanzar desde n_2. En algunas ocasiones es posible detectar esta situación, lo que permitiría realizar la poda temprana del nodo dominado.

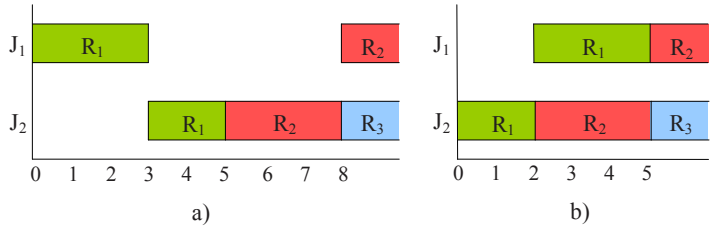


Figura 4.7: Planificación parcial de dos estados de la búsqueda, el estado b) domina al estado a)

Vamos a considerar un pequeño ejemplo. La Figura 4.7 muestra los diagramas de Gantt de dos planificaciones parciales, con tres tareas planificadas, correspondientes a dos estados del espacio de búsqueda para un problema con 2 trabajos y 3 o más máquinas. Si la segunda operación del trabajo J_1 requiere la máquina R_2 y la tercera operación del trabajo J_2 requiere la máquina R_3, es sencillo ver que la mejor solución que se puede alcanzar desde el estado de la Figura 4.7a no puede ser mejor que la mejor solución alcanzable desde el estado de la Figura 4.7b. La razón es que el problema residual, es decir, la parte del problema sin resolver, contiene el mismo conjunto de tareas en ambas situaciones y además las cabezas de todas las operaciones en la primera situación son mayores o iguales que en la segunda. En consecuencia, el estado de la Figura 4.7a puede ser podado si ambos estados se encuentren simultáneamente en memoria. Evidentemente, un buen heurístico guiaría la búsqueda de modo que el estado de la Figura 4.7b fuese explorado primero, si ambos estados se encontrasen en la tabla $ABIERTA$ al mismo tiempo. No obstante, el estado de la Figura 4.7a, junto con algunos de sus descendientes, podría ser expandido más adelante en la búsqueda. Por tanto, la poda temprana de este estado reduciría el espacio y, si el proceso de comparación de estados para comprobar su dominancia es realizado de forma eficiente, también reduciría el tiempo empleado en la búsqueda. Algunas ideas similares se han aplicado a otros problemas de búsqueda, por ejemplo, en [54] se propone un método para el problema de planificación de proyectos (Project Scheduling

Problem) y en [43] y [44] se proponen, respectivamente, una serie de métodos de poda para el problema de empaquetamiento en cubos (Bin Packing Problem) y para la versión del problema de corte de piezas bidimensional (two-dimensional Cutting Stock Problem).

De manera más formal, se puede definir la dominancia entre dos estados del siguiente modo.

Definición 4.1. *Dados dos estados n_1 y n_2, tales que $n_1 \notin P^*_{s\text{-}n_2}$ y $n_2 \notin P^*_{s\text{-}n_1}$, n_1 domina a n_2 si y sólo si $f^*(n_1) \leq f^*(n_2)$.*

Por supuesto, establecer la condición de dominancia entre dos estados depende del problema que se desea resolver y, en general, no es sencillo. Además, resulta muy complicado disponer de una estrategia eficiente que permita definir un método completo para determinar la dominancia y aplicarla a cada par de estados del espacio de búsqueda. Lo que hemos hecho es establecer una condición suficiente que permite detectar de forma eficiente algunas situaciones de dominancia entre un par de nodos para el problema $J||C_{max}$. Además, hemos establecido dos condiciones necesarias para que dicha condición suficiente se cumpla. Estas condiciones contribuirán a la eficiencia del método. Algunas de estas propiedades dependen del heurístico utilizado, por lo que es importante remarcar que consideramos el heurístico h_3 que como hemos visto se basa en el cálculo de JPS. Como podremos comprobar estas condiciones pueden ser evaluadas eficientemente, por lo que se consigue que el proceso de comprobar la dominancia sea eficiente pero a costa de no detectar todos los estados dominados.

Teorema 4.1. *Sean n_1 y n_2 dos estados, si $SC(n_2) \subseteq SC(n_1)$ y $\boldsymbol{r}_{n_1}(US(n_1)) \leq \boldsymbol{r}_{n_2}(US(n_1))$, entonces se cumplen las siguientes condiciones:*

1. $\boldsymbol{q}_{n_1}(US(n_1)) = \boldsymbol{q}_{n_2}(US(n_1))$.
2. $JPS(P(n_1)) \leq JPS(P(n_2))$.
3. $C_{max}(P(n_1)) \leq C_{max}(P(n_2))$

Demostración. La condición 1 viene del hecho de que cada tarea $v \in US(n_1)$ es una tarea no planificada en ambos estados n_1 y n_2. Consecuentemente, v no tiene ningún sucesor disyuntivo aún. Así que, según las ecuaciones (4.1), $q_v(n_1) = p_{SJ_v} + q_{SJ_v}(n_1)$ y $q_v(n_2) = p_{SJ_v} + q_{SJ_v}(n_2)$. Como $q_{end}(n_1) = q_{end}(n_2) = 0$, razonando por inducción desde el nodo *final* hacia atrás, tendremos finalmente $q_v(n_1) = q_v(n_2)$. Por lo tanto, $\boldsymbol{q}_{n_1}(US(n_1)) = \boldsymbol{q}_{n_2}(US(n_1))$.

Para probar la condición 2, denotamos por $P(n_2 \setminus n_1)$ al subproblema constituido por las tareas no planificadas en n_1 pero considerando las cabezas y colas de las mismas tareas en n_2, es decir $P(n_2 \setminus n_1) = (US(n_1), \mathbf{r}_{n_2}(US(n_1)), \mathbf{p}_{n_2}(US(n_1)), \mathbf{q}_{n_2}(US(n_1)))$. Los problemas $P(n_2 \setminus n_1)$ y $P(n_1)$ tienen las mismas tareas, pero las cabeza de cada tarea en $P(n_1)$ es menor o igual que la que tiene en $P(n_2 \setminus n_1)$, siendo las colas iguales. Por tanto, cualquier planificación con expulsión para las tareas de $US(n_1)|_m$ con las cabezas y colas que tienen en el problema $P(n_2 \setminus n_1)$ es también una planificación factible con expulsión para esas tareas con las cabezas y colas que tienen en el problema $P(n_1)$. Así, se tiene que $JPS(P(n_1)) \leq JPS(P(n_2 \setminus n_1))$. Por el mismo razonamiento $JPS(P(n_2 \setminus n_1)) \leq JPS(P(n_2))$, ya que las tareas en $P(n_2 \setminus n_1)$ son un subconjunto de las de $P(n_2)$, siendo las cabezas y colas de las tareas en común las mismas.

La condición 3 podemos probarla a través de un razonamiento análogo al utilizado para probar la condición 2, dado que $C_{max}(P(n_1)) \leq C_{max}(P(n_2 \setminus n_1)) \leq C_{max}(P(n_2))$. □

Teorema 4.2. *Sean n_1 y n_2 dos estados tales que $n_2 \notin P^*_{s\text{-}n_1}$. Si $SC(n_2) \subseteq SC(n_1)$, $\boldsymbol{r}_{n_1}(US(n_1)) \leq \boldsymbol{r}_{n_2}(US(n_1))$ y $f(n_1) \leq f(n_2)$, entonces se cumplen las siguientes condiciones, donde $D(n)$ denota la profundidad de un nodo n en el árbol de búsqueda:*

1. $D(n_1) \geq D(n_2)$.
2. $n_1 \notin P^*_{s\text{-}n_2}$.
3. *n_1 domina a n_2.*

Demostración. La condición 1 es trivial por $SC(n_2) \subseteq SC(n_1)$, ya que $D(n) = |SC(n)|$. A partir de la condición 1, la única posibilidad para que la condición 2 no se cumpla es que n_1 y n_2 sean el mismo nodo, pero esto no puede ser cierto ya que $n_2 \notin P^*_{s\text{-}n_1}$. Por lo tanto la condición 2 se cumple. Para probar la condición 3, debemos tener en cuenta que para un estado n, $f(n) = \text{máx}\{g(n), JPS(P(n))\}$ y $f^*(n) = \text{máx}\{g(n), C_{max}(P(n))\}$. Para los estados n_1 y n_2, por el teorema 4.1, se cumple que $JPS(P(n_1)) \leq JPS(P(n_2))$ y $C_{max}(P(n_1)) \leq C_{max}(P(n_2))$. Teniendo en cuenta que $f(n_1) \leq f(n_2)$ se cumple:

(a) $g(n_1) \leq g(n_2)$ o bien

(b) $g(n_1) > g(n_2)$ y $JPS(P(n_2)) \geq g(n_1)$.

Si se cumple (a), como por el teorema 4.1 se cumple $C_{max}(P(n_1)) \leq C_{max}(P(n_2))$, tenemos que $f^*(n_1) \leq f^*(n_2)$.

Si se cumple (b), se cumple $C_{max}(P(n_2)) \geq JPS(P(n_2)) \geq g(n_1) > g(n_2)$, entonces $f^*(n_1) \leq C_{max}(P(n_2)) \leq f^*(n_2)$.

Por lo tanto n_1 domina a n_2. □

En la práctica, la condición $\mathbf{r}_{n_1}(US(n_1)) \leq \mathbf{r}_{n_2}(US(n_1))$, que aparece en los teoremas anteriores, no hay que comprobarla para todas las tareas sin planificar del estado n_1. Como consecuencia del resultado que se muestra a continuación (teorema 4.3), es suficiente hacer la comprobación para las tareas cuyas cabezas en el estado n_1 tengan un valor menor que $g(n_1)$. En nuestro caso, como el método comprueba, en caso de igualdad, si n_1 domina a n_2 o si n_2 domina a n_1, se comparara hasta $\text{máx}(g(n_1), g(n_2))$.

Teorema 4.3. *Sean n_1 y n_2 dos estados. Si $SC(n_2) \subseteq SC(n_1)$ y $\forall \theta \in US(n_1)$ con $r_\theta(n_1) \leq g(n_1)$ se cumple $r_\theta(n_1) \leq r_\theta(n_2)$, entonces se cumple que $\boldsymbol{r}_{n_1}(US(n_1)) \leq \boldsymbol{r}_{n_2}(US(n_1))$.*

Demostración. La prueba se basa en el hecho de que si una tarea θ en n_1 cumple $r_\theta(n_1) > g(n_1)$, entonces la cabeza de θ viene de la tarea anterior en su trabajo $r_\theta(n_1) = r_{PJ_\theta}(n_1) + p_{PJ_\theta}(n_1)$, ya que $g(n_1)$ es el tiempo máximo de finalización de las tareas planificadas en n_1. Si θ es la primera tarea de su trabajo que cumple $r_{PJ_\theta}(n_1) \leq g(n_1)$ y $r_\theta(n_1) > g(n_1)$, entonces $r_{PJ_\theta}(n_1) \leq r_{PJ_\theta}(n_2)$ y en consecuencia $r_\theta(n_1) \leq r_\theta(n_2)$ y todas las tareas posteriores a θ en su trabajo cumplirán la condición. □

4.6.1. Reglas de Dominancia

Una vez que se tiene la descripción formal de las propiedades de dominancia entre estados, es necesario diseñar una serie de reglas que permitan comprobar esta condición, entre dos nodos, en algoritmo A^*.

Para establecer que un nodo n_1 domina a un nodo n_2 se deben verificar las siguientes condiciones:

1. $n_2 \notin P^*_{s\text{-}n_1}$.
2. $SC(n_2) \subseteq SC(n_1)$.
3. $\mathbf{r}_{n_1}(US(n_1)) \leq \mathbf{r}_{n_2}(US(n_1))$.
4. $f(n_1) \leq f(n_2)$.

Además, podemos tener en cuenta que si $D(n_1) < D(n_2)$ no se pueden cumplir las cuatro condiciones simultáneamente, tal y como indica el teorema 4.2. Así, en principio, cuando un nodo n_1 aparece durante la búsqueda, podemos optar por compararlo con cada nodo n_2 encontrado previamente, es decir con todos aquellos que están en $ABIERTA$ y con los que ya fueron expandidos y que almacenaremos en una lista $CERRADA$. En este proceso consideraremos los nodos n_2 de $ABIERTA$ tales que $f(n_1) \leq f(n_2)$, es decir los que están detrás de n_1, una vez insertado n_1 en $ABIERTA$. Para cada uno de estos nodos comprobaremos que $D(n_1) \geq D(n_2)$ y luego las condiciones 2 y 3. Hay que tener en cuenta que si $f(n_1) = f(n_2)$ y $D(n_1) = D(n_2)$ puede ocurrir que n_1 domine a n_2 o que n_2 domine a n_1. La condición 1 se cumple siempre por el hecho de estar n_2 en $ABIERTA$. En cuanto a los nodos de $CERRADA$ el proceso se complica debido a que esta lista no está en principio ordenada con ningún criterio. En cualquier caso, con este método se comparan muchos nodos nuevos n_1 que nunca serán expandidos por el algoritmo A^* y que suponen una parte importante de los nodos visitados durante la búsqueda.

Para mejorar la eficiencia del proceso proponemos una alternativa que consiste en retrasar el proceso de chequeo de dominancia al momento de expansión de los estados. Así, cuando un nodo n_1 se elige para expansión, se compara con los nodos n_2 de $ABIERTA$ tales que $f(n_1) = f(n_2)$, estos nodos están al principio de $ABIERTA$ y en este caso n_1 puede dominar a n_2 y también n_2 a n_1. La comparación de n_1 con nodos n_2 de $ABIERTA$ tales que $f(n_1) < f(n_2)$ puede retrasarse al momento de expansión de n_2; en ese momento n_1 estará en la lista $CERRADA$. Además el nodo n_1 debe compararse con los nodos n_2 de $CERRADA$. En este caso $f(n_1) \geq f(n_2)$, por la consistencia del heurístico h_3, en consecuencia lo que procede es chequear las condiciones 2 y 3 para los nodos n_2 tales que $D(n_1) \leq D(n_2)$, para comprobar si n_2 domina a n_1.

De forma más concreta, el proceso de chequeo de dominancia se realiza con las dos reglas siguientes:

1. Cuando un nodo n es seleccionado por el algoritmo A^* para ser expandido, n se compara con todo nodo n' en $ABIERTA$ con $f(n) = f(n')$. Si alguno de los nodos es dominado, se poda. En el caso de que ambos se dominen mutuamente, se poda el nodo n'.

2. Si el nodo n no es podado por el paso 1, se compara con aquellos nodos n' que se encuentran en $CERRADA$ y que cumplen que $D(n') \geq D(n)$ ($f(n') \leq f(n)$ como consecuencia de la consistencia del heurístico empleado). Si n' domina a n, entonces el nodo n se poda.

4.7. Estudio Experimental

En esta sección vamos a mostrar los diferentes experimentos realizados para el problema $J||C_{max}$ en condiciones de optimalidad. En primer lugar analizaremos el comportamiento de los diferentes heurísticos propuestos. En segundo lugar estudiaremos la técnica de poda tal y como ha sido descrita en la sección 4.6.1, así como de algunas variantes que tienen como objetivo mejorar el tiempo de ejecución. Además, trataremos de poner de manifiesto cuál es el tamaño límite de problemas que el algoritmo A^* es capaz de resolver y la influencia de la técnica de poda tanto en el espacio de memoria como en el tiempo de ejecución del algoritmo.

Para realizar los experimentos se emplean una serie de problemas seleccionados del repositorio *OR-library*, originalmente descrito por J.E.Beasley en [7] y que se puede encontrar en [6]. Este repositorio contiene instancias de problemas JSS de diferente tamaño y dificultad. Concretamente experimentamos con las siguientes baterías de problemas: $FT06$ (6×6), $LA01 - 20$ (10×5, 15×5, 20×5, 10×10), $ORB01 - 10$ (10×10), $ABZ05 - 06$ (10×10). Asimismo, hemos considerado versiones reducidas de los problemas, incluidos los problemas de la baterías YN y *Selectos* (problemas identificados como difíciles [2]): $FT20$ (20×5), $FT10$ (10×10), $ABZ07 - 09$ (20×15, 20×20), $LA21 - 25$ (15×10), $LA27 - 29$ (20×10), $LA38 - 40$ (15×15) y $YN1 - 4$ (20×20). Además, también hemos considerado el conjunto de instancias propuestas por Sadeh y Fox en [74]. Se trata de un conjunto de 60 instancias de tamaño 10×5 organizadas en 6 grupos de 10 problemas cada uno. Cada grupo se caracteriza por dos parámetros: BK (número de recursos cuello de botella) y RG (parámetro de rango). Un recurso es un cuello de botella si aparece en la misma posición en la secuencia de máquinas para todos los trabajos. RG controla la distribución de los tiempos de inicio y fin de los trabajos (due dates).

La máquina donde se realizan todos los experimentos tiene un procesador Intel Core 2 Duo a $2,13$ GHz y con $7,6$ Gb. de RAM y los experimentos se han ejecutado sobre Ubuntu $V8{,}04$, una distribución de GNU/Linux. El tiempo límite establecido para la resolución de los problemas es de una hora. Cuando los problemas no se hayan resuelto, bien por que se ha agotado la memoria disponible, bien porque se ha superado el tiempo límite establecido, se mostrarán en negrita los valores alcanzados hasta ese momento.

4.7.1. Comparación de Heurísticos

Todos los heurísticos propuestos se obtienen a partir de las soluciones óptimas de problemas relajados, por lo que son consistentes; en esta sección estudiamos su comportamiento. Utilizaremos para estas pruebas el $FT06$ instancia de tamaño 6×6 del conjunto de problemas FT y con el conjunto de problemas LA del 01 al 15, cuyos tamaños son de 10×5, 15×5 y 20×5.

La Tabla 4.1 muestra los resultados de aplicar los heurísticos anteriormente descritos a la ins-

Tabla 4.1: Resultados para el problema $FT06$ resuelto con los diferentes heurísticos

Heurístico	Gen.	Exp.	T.(s)
h_0	6645904	4270444	375
h_1	24822	14157	2
h_2	679	453	0
h_3	177	105	0
h_3^*	177	105	0

Tabla 4.2: Resultados para los problemas LA01-15 resueltos con los diferentes heurísticos

	h_2			h_3		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA01$	6267	3612	0	810	418	0
$LA02$	1639181	911785	136	123337	57103	12
$LA03$	**4425506**	**2290093**	**367**	658	249	0
$LA04$	876009	528515	80	140087	63969	13
$LA05$	**7093914**	**4296250**	**576**	20939	8397	2
$LA06$	1242826	719031	145	34744	14270	4
$LA07$	237540	127558	28	4865	1853	1
$LA08$	3558172	1988304	425	7525	2926	1
$LA09$	27177	15060	3	1551	678	0
$LA10$	**2874314**	**1682222**	**322**	**2069243**	**877104**	**261**
$LA11$	**2047588**	**1149135**	**307**	**1376265**	**477883**	**232**
$LA12$	1197934	684166	196	4559	1689	1
$LA13$	1991097	1092645	304	371787	114076	68
$LA14$	117150	68097	20	678	258	0
$LA15$	1784507	886125	278	241157	76967	42

tancia del problema $FT06$. Para cada heurístico se muestran el número de nodos generados y expandidos, así como el tiempo requerido para llegar a una solución. El problema $FT06$ es una instancia muy pequeña que puede ser resuelta incluso con el heurístico trivial $h_0(n) = 0, \forall n$. h_3^* se refiere al heurístico h_3 pero calculando dos veces la JPS, una para el problema Primal (problema relajado original) y otra para el problema Dual (problema relajado intercambiando cabezas por colas), tal y como se sugiere en 4.5.3, ya que los valores pueden ser distintos. Como se puede observar el heurístico que produce mejores resultados es el h_3, se trata claramente de un heurístico mejor informado que orienta la búsqueda hacia caminos más prometedores que el resto de heurísticos, consiguiendo reducir de forma considerable el espacio que es necesario explorar para alcanzar una solución óptima. Como cabe esperar, la búsqueda sin el empleo de un heurístico resulta una tarea ardua incluso para un problema pequeño como es el $FT06$.

La Tabla 4.2 muestra los resultados sobre las instancias LA01-15. Para cada instancia se muestran el número de nodos generados y expandidos, así como el tiempo requerido para llegar a una solución. Para los problemas no resueltos se muestran estos valores en el momento en el que la memoria se agotó. En este caso los heurísticos h_0 y h_1 no son capaces de resolver ninguna de las instancias. Tampoco hemos considerado el heurístico h_3^*, ya que las mejoras con respecto a h_3 son

prácticamente nulas. El heurístico h_2 es capaz de resolver todas las instancias del problema exceptuando $LA03$, $LA05$, $LA10$ y $LA11$, mientras que el heurístico h_3 solamente deja sin resolver las instancias $LA10$ y $LA11$.

Aunque los heurísticos h_2 y h_3 no son estrictamente comparables, ya que proceden de relajaciones del problema diferentes, en la práctica h_3 se comporta mucho mejor que h_2. En las instancias que resuelven los dos heurísticos tanto el número de nodos expandidos como el tiempo de ejecución son mucho menores con h_3 que con h_2.

A partir de esta sección utilizaremos siempre el heurístico h_3, ya que es claramente el más eficiente de todos.

4.7.2. Análisis de la Técnica de Poda

En esta sección se describen los resultados obtenidos con las reglas de poda descritas en la sección 4.6.1; así como los obtenidos con algunas variantes de este método. Para estos experimentos se utilizaron las instancias $LA01 - 15$.

Para analizar por separado la eficiencia de las podas realizadas con las dos reglas enunciadas en la sección 4.6.1, en primer lugar se llevó a cabo un estudio experimental utilizando únicamente la regla 1. De acuerdo con esta regla, cada nodo expandido se compara solamente con aquellos que han aparecido previamente en la búsqueda y que aún no han sido expandidos. La Tabla 4.3 muestra los resultados de las dos formas de aplicar la poda, es decir comparando el nodo expandido n con todos los de $ABIERTA$ con mayor o igual $f (f >=)$ y comparando solamente con aquellos de $ABIERTA$ con igual f $(f ==)$. Como se puede observar, salvo para alguna instancia, la reducción del número de nodos expandidos es insignificante, y en cualquier caso el ahorro de nodos expandidos no compensa el coste del chequeo de la condición de dominancia. El comportamiento es especialmente malo en el caso en el que se compara el nodo expandido con todos lo que están en ese momento en $ABIETA$.

Tras estos resultados experimentales decidimos, como se indicó en la regla 1 de poda por dominancia, comparar sólo el nodo expandido n con aquellos nodos n' en $ABIERTA$ con $f(n) = f(n')$, dejando de comprobar con los nodos n' en $ABIERTA$ con $f(n) < f(n')$. En esta situación n podría dominar a n' y no ser detectado; sin embargo esto puede ser detectado más tarde si el nodo n' es seleccionado para su expansión, cuando n esté en la tabla $CERRADA$. No obstante, la Tabla 4.3 pone de manifiesto que la realización de podas únicamente en $ABIERTA$, aun logrando reducir el número de nodos expandidos, no resulta competitiva en cuanto al tiempo empleado para alcanzar la solución óptima. El mejor tiempo de ejecución se obtiene cuando no se utiliza la poda por dominancia, ya que sin realizar podas se resuelven las mismas instancias que aplicando la poda a nodos con igual valor de f y en un tiempo similar o menor. Esto ocurre porque el número de podas realizado de este modo es reducido con lo que el ahorro de memoria es pequeño y el tiempo de ejecución se incrementa con las operaciones de chequeo de la condición de dominancia. Aún así, estos resultados nos indicaron que la poda por dominancia es una técnica interesante ya que el ahorro de memoria puede ser bastante elevado en algunos casos, como ocurre por ejemplo en la instancia $LA04$ en la que para resolverla utilizando la poda, es necesario expandir la mitad de nodos que si no se utiliza.

En cualquier caso, la regla 1 sola no es eficiente. Para aplicar la regla 2, es preciso incorporar al

algoritmo A^* la tabla $CERRADA$ que, como se ha indicado anteriormente, contiene aquellos nodos que han sido expandidos previamente. Para poder aplicar el paso 2 de forma eficiente es necesario que la estructura de la tabla $CERRADA$ permita encontrar aquellos nodos n' con $D(n') \geq D(n)$ de forma rápida. Para lograr esto, la tabla $CERRADA$ debe mantenerse ordenada de acuerdo con la profundidad de los nodos expandidos. Los resultados de estos experimentos se muestran en la Tabla 4.4. Como se puede observar, la realización de las podas en $CERRADA$ no sólo reduce considerablemente el número de nodos expandidos para alcanzar la solución, sino que además se resuelve la instancia $LA10$. Sin embargo, solamente se produce una reducción del tiempo de ejecución cuando el número de nodos expandidos se reduce a su vez de forma muy elevada.

Estos resultados ponen de manifiesto, como conclusión preliminar, que la incorporación de podas por dominancia desde la tabla $CERRADA$ reduce considerablemente el espacio de búsqueda efectivo, pero no supone un ahorro de tiempo. A la vista de estos resultados, nos planteamos simplificar el mecanismo de aplicación de las reglas de dominancia. Para ello analizamos la forma en la que el algoritmo va realizando las podas, y en particular las diferencias de profundidad de cada par de nodos que da lugar a una poda. Este estudio experimental pone de manifiesto que la mayoría de las podas se realizan cuando se comparan pares de nodos que están a una profundidad muy parecida. Los resultados de este estudio se muestran en la Tabla 4.5. Para cada instancia se muestra el número de nodos podados en la tabla $ABIERTA$ al comparar con el nodo expandido, las podas de nodos expandidos desde la tabla $CERRADA$, la máxima diferencia de profundidad a la que se produce alguna poda, y el número de nodos podados a las distintas diferencias de profundidad. Como se puede observar la mayoría de las podas se producen cuando los nodos tienen igual profundidad, es decir igual número de tareas planificadas. Mientras que el número de podas se reduce a medida

Tabla 4.3: Resultados para los problemas LA01-15 con el heurístico h_3 y realizando podas en la tabla $ABIERTA$ con nodos con mayor o igual f o con nodos con igual f

	No poda			Poda $f >=$			Poda $f ==$		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
LA01	810	418	0	810	418	0	810	418	0
LA02	123337	57103	12	121112	55971	1006	121112	55971	13
LA03	658	249	0	652	247	0	652	247	0
LA04	140087	63969	13	70512	31031	306	73192	32218	11
LA05	20939	8397	2	20939	8397	13	20939	8397	3
LA06	34744	14270	4	34736	14265	66	34736	14265	6
LA07	4865	1853	1	4865	1853	2	4865	1853	1
LA08	7525	2926	1	7525	2926	8	7525	2926	2
LA09	1551	678	0	1551	678	0	1551	678	0
LA10	**2069243**	**877104**	**261**	**206574**	**86759**	**3600**	**2072187**	**879558**	**404**
LA11	**1376265**	**477883**	**232**	**265598**	**90409**	**3600**	**1376336**	**477874**	**424**
LA12	4559	1689	1	4559	1689	4	4559	1689	2
LA13	371787	114076	68	**267969**	**77331**	**3600**	367608	112309	103
LA14	678	258	0	678	258	0	678	258	0
LA15	241157	76967	42	**231668**	**73147**	**3600**	240006	76534	62

que aumenta la diferencia de profundidad. Es razonable que esto sea así, ya que los nodos con una diferencia de profundidad elevada representan estados muy diferentes, y en consecuencia es menos probable que se den las condiciones de poda.

Estos resultados sugieren reducir las comprobaciones de dominancia en la tabla $CERRADA$ de forma que el nodo expandido n se compare solamente con nodos n' en $CERRADA$ con $D(n') =$

Tabla 4.4: Resultados para los problemas LA01-15 con el heurístico h_3 y realizando podas en la tabla $ABIERTA$ y la tabla $CERRADA$

	No poda			Poda AB			Poda AB y CE		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
LA01	810	418	0	810	418	0	343	158	0
LA02	123337	57103	12	121112	55971	13	22375	9454	40
LA03	658	249	0	652	247	0	571	216	0
LA04	140087	63969	13	73192	32218	11	17928	7309	17
LA05	20939	8397	2	20939	8397	3	8781	3518	5
LA06	34744	14270	4	34736	14265	6	4858	1935	2
LA07	4865	1853	1	4865	1853	1	3017	1158	1
LA08	7525	2926	1	7525	2926	2	3859	1494	1
LA09	1551	678	0	1551	678	0	1034	436	1
LA10	**2069243**	**877104**	**261**	**2072187**	**879558**	**404**	105606	37894	638
LA11	**1376265**	**477883**	**232**	**1376336**	**477874**	**424**	**277955**	**94973**	**3600**
LA12	4559	1689	1	4559	1689	2	2751	952	1
LA13	371787	114076	68	367608	112309	103	40717	13111	48
LA14	678	258	0	678	258	0	675	257	0
LA15	241157	76967	42	240006	76534	62	60947	20022	151

Tabla 4.5: Diferencias de profundidad en las que se producen las podas en $ABIERTA$ y $CERRADA$, resultados para los problemas LA01-15 con el heurístico h_3

	AB		CE		Diferencias de Profundidad							
Instancia	Podas	Max DP	Podas	Max DP	0	1	2	3	4	5	6	7
LA01	1	48	10	21	6		2	3				
LA02	16	48	1790	38	1279	443	77	7				
LA03	1	43	9	12	8	2						
LA04	228	42	1112	41	1017	238	68	5	1	10	1	
LA05	1	48	442	18	344	95	3	1				
LA06	1	72	313	24	207	86	11	9	1			
LA07	1	72	135	16	90	45			1			
LA08	4	73	211	20	195	14	3	1	2			
LA09	2	73	25	20	22	3	2					
LA10	45	73	10659	35	6811	3020	620	159	81	13		
LA11	18	92	39642	42	25825	11049	2246	475	59	5	1	
LA12	1	98	66	20	52	15						
LA13	41	98	3180	30	2892	223	81	4	13	6	1	1
LA14	1	98	1	12	1	1						
LA15	19	98	3908	39	2759	976	164	25	3			

$D(n)$. La Tabla 4.6 muestra los resultados de los experimentos realizados teniendo en cuenta esta simplificación. Como se puede observar, los tiempos se reducen considerablemente. A la vez, se produce un incremento en el número de nodos expandidos que no es muy elevado y se resuelven el mismo número de instancias. Estos resultados nos llevaron a adoptar una solución de compromiso que consiste en realizar podas en $CERRADA$ sólo a igual profundidad; de este modo aunque el número de nodos expandidos es un poco mayor, los tiempos son mucho menores.

Una observación importante que se puede hacer en este punto es que en el paso 2, el nodo n puede dominar a n' si $f(n) = f(n')$. En este caso, n' y todos los descendientes de n' a cualquier nivel de la búsqueda, algunos de los cuales pueden estar en la tabla $ABIERTA$, pueden ser podados también. Para poder hacer esto es necesario buscar en toda la tabla $CERRADA$ y mantener la traza de los sucesores para cada nodo expandido. Se realizan una serie de experimentos en los que se poda en $ABIERTA$ y en $CERRADA$ (con $D(n') = D(n)$), podando además en $ABIERTA$ los hijos de los nodos podados en $CERRADA$ (esta estrategia se representa por AB y CEH en la Tabla 4.7). Los resultados experimentales mostraron (ver Tabla 4.7) que esta posibilidad no es eficiente. La razón de esto es que la mayoría de los nodos en $ABIERTA$ podados de este modo, son podados también por comparación con otros nodos.

Tomando la versión que parece más prometedora, si cada nodo expandido se compara solamente con nodos de igual profundidad en $CERRADA$, la única posibilidad de que se realice una poda es que las tareas planificadas en los dos estados sean las mismas. Este hecho permite representar $CERRADA$ mediante una estructura de tipo diccionario (contenedor asociativo *map* de la Standard Template Library de C++) en el que cada entrada contiene los nodos con las mismas tareas planificadas. Las claves son mapas de bits que indican cuáles son las tareas que están planificadas y cuáles

Tabla 4.6: Resultados para los problemas LA01-15 con el heurístico h_3 y realizando podas en la tabla $ABIERTA$ y la tabla $CERRADA$, a diferencia de profundidad 0 (igual profundidad)

	No poda			Poda AB y CE			Poda AB y CE $(DP = 0)$		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
LA01	810	418	0	343	158	0	351	165	0
LA02	123337	57103	12	22375	9454	40	24374	10509	5
LA03	658	249	0	571	216	0	572	217	0
LA04	140087	63969	13	17928	7309	17	18989	7888	4
LA05	20939	8397	2	8781	3518	5	9240	3731	1
LA06	34744	14270	4	4858	1935	2	5380	2220	2
LA07	4865	1853	1	3017	1158	1	3242	1243	0
LA08	7525	2926	1	3859	1494	1	3894	1517	1
LA09	1551	678	0	1034	436	1	1038	439	0
LA10	**2069243**	**877104**	**261**	105606	37894	638	140194	52713	114
LA11	**1376265**	**477883**	**232**	**277955**	**94973**	**3600**	**799520**	**287609**	**3600**
LA12	4559	1689	1	2751	952	1	2766	965	1
LA13	371787	114076	68	40717	13111	48	41635	13599	18
LA14	678	258	0	675	257	0	675	257	0
LA15	241157	76967	42	60947	20022	151	65565	22068	37

no lo están. Con esta estructura el tiempo de ejecución se reduce notablemente. Los resultados de los experimentos realizados con esta estrategia se muestran en la Tabla 4.8, donde *CEM* se refiere a la tabla *CERRADA* implementada mediante la estructura indicada anteriormente. Claramente,

Tabla 4.7: Resultados para los problemas *LA*01-15 con el heurístico h_3 y realizando podas en la tabla *ABIERTA* y la tabla *CERRADA* ($DP = 0$), podando en *ABIERTA* los hijos de los nodos podados en *CERRADA*

	No poda			Poda *AB* y *CE* ($DP = 0$)			Poda *AB* y *CEH* ($DP = 0$)		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*LA*01	810	418	0	351	165	0	351	165	0
*LA*02	123337	57103	12	24374	10509	5	24374	10509	6
*LA*03	658	249	0	572	217	0	572	217	0
*LA*04	140087	63969	13	18989	7888	4	18978	7883	4
*LA*05	20939	8397	2	9240	3731	1	9240	3731	1
*LA*06	34744	14270	4	5380	2220	2	5380	2220	2
*LA*07	4865	1853	1	3242	1243	0	3242	1243	0
*LA*08	7525	2926	1	3894	1517	1	3894	1517	1
*LA*09	1551	678	0	1038	439	0	1038	439	1
*LA*10	**2069243**	**877104**	**261**	140194	52713	114	140194	52713	168
*LA*11	**1376265**	**477883**	**232**	**799520**	**287609**	**3600**	**663393**	**239904**	**3600**
*LA*12	4559	1689	1	2766	965	1	2766	965	1
*LA*13	371787	114076	68	41635	13599	18	41635	13599	22
*LA*14	678	258	0	675	257	0	675	257	1
*LA*15	241157	76967	42	65565	22068	37	65565	22068	43

Tabla 4.8: Resultados para los problemas *LA*01-15 con el heurístico h_3 y realizando podas en la tabla *ABIERTA* y la tabla *CERRADA*, utilizando para *CERRADA* un diccionario que agiliza la poda por dominancia

	No poda			Poda *AB* y *CE* ($DP = 0$)			Poda *AB* y *CEM*		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*LA*01	810	418	0	351	165	0	351	165	0
*LA*02	123337	57103	12	24374	10509	5	24374	10509	3
*LA*03	658	249	0	572	217	0	572	217	0
*LA*04	140087	63969	13	18989	7888	4	18989	7888	2
*LA*05	20939	8397	2	9240	3731	1	9240	3731	2
*LA*06	34744	14270	4	5380	2220	2	5380	2220	1
*LA*07	4865	1853	1	3242	1243	0	3242	1243	0
*LA*08	7525	2926	1	3894	1517	1	3894	1517	0
*LA*09	1551	678	0	1038	439	0	1038	439	1
*LA*10	**2069243**	**877104**	**261**	140194	52713	114	140194	52713	31
*LA*11	**1376265**	**477883**	**232**	**799520**	**287609**	**3600**	**1059363**	**384154**	**446**
*LA*12	4559	1689	1	2766	965	1	2766	965	1
*LA*13	371787	114076	68	41635	13599	18	41635	13599	14
*LA*14	678	258	0	675	257	0	675	257	0
*LA*15	241157	76967	42	65565	22068	37	65565	22068	20

con esta simplificación el tiempo se reduce mucho con lo que la poda por dominancia se vuelve una técnica muy efectiva tanto en la reducción del espacio como del tiempo.

Veamos ahora que ocurre si aplicamos la técnica de poda por dominancia, con la simplificación indicada en el párrafo anterior, a la resolución del problema $FT06$ (6×6) para los diferentes heurísticos propuestos. La Tabla 4.9 recoge los resultados de estos experimentos. Como se observa, la técnica de poda por dominancia logra, para todos los heurísticos, una importante reducción del número de nodos que es necesario expandir para alcanzar una solución. Esta reducción es grandísima cuando no se emplea heurístico, es decir cuando se usa el heurístico h_0. Aún siendo el $FT06$ un problema pequeño, este experimento pone de manifiesto que el efecto de la técnica de poda por dominancia está en relación inversa con el grado de información del heurístico.

Tabla 4.9: Resultados para el problema $FT06$ resuelto con los diferentes heurísticos y empleando la técnica de poda

	No poda			Poda		
Heurístico	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
h_0	6645904	4270444	375	17999	11241	2
h_1	24822	14157	2	4046	2532	0
h_2	679	453	0	342	222	0
h_3	177	105	0	107	62	0
h_3^*	177	105	0	107	62	0

Tras esta serie de simplificaciones las reglas de dominancia quedan del siguiente modo:

1. Cada vez que un nodo n es seleccionado por le algoritmo A^* para ser expandido, n se compara con cada nodo n' en $ABIERTA$ con $f(n) = f(n')$. Si uno de ellos es dominado por el otro, se poda. Si los dos se dominan mutuamente, se poda n'.

2. Si el nodo n no es podado por el paso 1, se compara con aquellos nodos n' que se encuentran en $CERRADA$ y que cumplen que $D(n') = D(n)$ $(f(n') \leq f(n))$. Si n' domina a n, entonces el nodo n es podado.

4.7.3. Eficiencia de la Técnica de Poda en condiciones de Optimalidad

Los experimentos de esta sección permiten determinar el tamaño límite de problemas que somos capaces de resolver en condiciones de optimalidad y en comparación con otros métodos, así como comprobar el efecto que la técnica de poda por dominancia produce tanto en el espacio de búsqueda como en el tiempo de ejecución. Para estos experimentos utilizamos problemas de las baterías FT, LA, ORB, ABZ, YN y *Selectos*. En algunos casos hemos realizado una reducción del tamaño de las instancias, eliminando trabajos y máquinas, ya que su tamaño real es mucho mayor del tamaño de los problemas que el algoritmo es capaz de resolver. Además, hemos experimentado con los 60 problemas propuestos por Sadeh en [74].

En todos los casos se utiliza el heurístico h_3. Los resultados de estos experimentos se recogen en las tablas del Anexo 4.10. En ellas para cada problema se muestra el número de nodos generados y

expandidos, y el tiempo de ejecución empleado para resolver los problemas. Asimismo, se incluyen datos estadísticos como la media y la desviación típica de los mismos.

En esta sección, para ver el efecto de la poda a medida que el tamaño de los problemas crece, se muestran una serie de tablas resumen en las que para cada tamaño de problema se puede ver el número de instancias resueltas, así como las medias y las desviaciones típicas del número de nodos generados, expandidos y del tiempo de ejecución para aquellos problemas resueltos con y sin la técnica de poda.

Instancias de Tamaño 6×6 a 9×9

La Tabla 4.10, muestra un resumen de los resultados de los experimentos realizados con los problemas desde un tamaño 6×6 hasta a un tamaño 9×9.

Para ver tanto el efecto de la poda como el límite de tamaño que somos capaces de resolver óptimamente se redujeron algunos de los problemas de las baterías utilizadas para convertirlos en problemas más pequeños. El método empleado para la reducción de los problemas es el siguiente: partiendo de un problema de tamaño $n \times m$ si queremos reducir a un tamaño menor lo que se hace es reducir el número de trabajos y máquinas borrando los trabajos de mayor número y las máquinas de mayor número. Veamos un ejemplo: imaginemos que tenemos el problema $FT06$ que tiene tamaño 6×6, para obtener el problema reducido de 4×3 ($FT06 - 4 \times 3$) se borran los dos últimos trabajos (trabajos 5 y 6) y del resto de trabajos se eliminan las tareas que requieren a las máquinas 4, 5 y 6. De este modo se respeta la distribución de las máquinas entre todos los trabajos.

Como el problema más pequeño de la *OR-library* es de tamaño 6×6, decidimos recortar los problemas de las diferentes baterías empleadas a ese tamaño, e ir incrementando el tamaño hasta un tamaño de 10×10, que es el tamaño umbral a partir del cual se considera que los problemas dejan de ser pequeños. Estos problemas fueron resueltos utilizando la técnica de poda y sin ella. Los resultados de estos experimentos se recogen en las Tablas 4.12, 4.13, 4.14, 4.15 y 4.16. Para poder ver el efecto de la poda, a medida que el tamaño de los problemas crece, incluimos una tabla resumen (Tabla 4.10) en la que se pueden ver para cada tamaño de problema en primer lugar el número de instancias resueltas, y en segundo lugar las medias y las desviaciones típicas del número de nodos generados, expandidos y del tiempo de ejecución para aquellos problemas resueltos con y sin la técnica de poda. Los resultados recogidos en esta tabla muestran que a medida que el tamaño de los problemas crece la eficacia de la técnica de poda se incrementa, es decir, cuanto mayor es el espacio de búsqueda más efectiva resulta la técnica de poda. Para los problemas de tamaño 6×6 la reducción en media del número de nodos expandidos no es muy significativa, siendo las medias de los tiempos similares. En los problemas de tamaño 7×7 comienza a apreciarse el efecto de la poda, siendo los tiempos en media aún muy parecidos. Es en los problemas de tamaño 8×8 en los que se aprecia un salto en cuanto a la reducción del número de nodos expandidos y del tiempo de ejecución, resolviéndose con poda un problema que no se resuelve sin poda (Tabla 4.14). En los problemas de tamaño 9×9 el efecto de la poda es mucho mayor y hace que se resuelvan 8 problemas para los que sin poda se agota la memoria sin alcanzar solución (Tabla 4.15). Sin embargo para este tamaño de problemas ya hay una instancia que no se resuelve en el tiempo límite establecido

la $ORB08-9\times9$. Esta instancia, sin embargo, es resuelta cuando no se establece límite de tiempo, en 9861 segundos, generando 1903538 nodos y expandiendo 1050179 nodos. Esto nos hace pensar que el límite de problemas que podemos resolver de forma óptima va a estar próximo al umbral de tamaño para el cual los problemas dejan de considerarse pequeños, tamaño 10×10.

Tabla 4.10: Resultados resumen para problemas y problemas recortados de las baterías *LA*, *ORB*, *ABZ* y *Selectos*, con el heurístico h_3 y realizando podas

6×6						
	No poda			Poda		
Resueltos	32/32			32/32		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	1246,50	724,41	0,09	540,31	312,78	0,06
Desviación	1582,11	921,99	0,29	500,12	288,85	0,24
7×7						
	No poda			Poda		
Resueltos	32/32			32/32		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	6615,56	3812,88	0,59	2027,84	1138,38	0,22
Desviación	7737,10	4454,37	0,82	1515,89	840,65	0,41
8×8						
	No poda			Poda		
Resueltos	31/32			32/32		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	120946,68	66897,16	14,94	17043,32	9324,61	3,35
Desviación	145594,47	79269,65	18,14	16472,05	8733,14	3,79
9×9						
	No poda			Poda		
Resueltos	23/32			32/32		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	678814,68	379322,27	91,50	66150,95	36738,23	22,36
Desviación	687502,82	379322,37	95,70	61524,24	33170,66	30,24
10×10						
	No poda			Poda		
Resueltos	5/32			18/32		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	442587,80	242877,00	72,60	38824,40	21195,40	11,60
Desviación	494136,02	266086,77	81,46	19464,22	10159,56	6,95

Instancias de Tamaño 10×10

Veamos ahora qué ocurre para los problemas que se encuentran en el tamaño umbral. En las Tablas 4.10 y 4.16 se observa que el efecto de la poda sigue el comportamiento esperado, siendo el número de nodos expandidos y el tiempo empleado para encontrar la solución mucho menor usando poda que sin ella. Además, con poda se resuelven cinco instancias de problemas no recortados (*LA*18, *LA*20, *ABZ*6, *ORB*07 y *ORB*10) que sin poda no es posible resolver, así como todas las

instancias recortadas excepto una ($LA40 - 10 \times 10$). En resumen, empleando la técnica de poda se resuelven 13 instancias de problemas que sin poda el algoritmo no es capaz de resolver antes de agotar la memoria disponible. Además, para los problemas que se resuelven de ambas formas el número de nodos expandidos y el tiempo de ejecución es considerablemente menor cuando se emplea la técnica de poda. Por otro lado, como se puede ver en la Tabla 4.24 del Anexo, para los problemas no resueltos, tanto las cotas inferiores como la máxima profundidad alcanzada son iguales o mejores empleando la técnica de poda.

Instancias de Sadeh y Fox

En el siguiente grupo de experimentos consideramos el conjunto de instancias propuestas por Sadeh y Fox en [74]. Como se puede ver en las Tablas de la 4.18 a la 4.22 y en la Tabla resumen

Tabla 4.11: Resultados resumen para los problemas de *Sadeh* organizadas por grupos según sus parámetros BK y RG, con el heurístico h_3 y realizando podas

$BK = 1, RG = 0,0$, Resueltos 10, Expandiendo 51 estados 8						
	No poda			Poda		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	358,00	159,50	0,10	287,10	127,70	0,00
Desviación	738,68	323,50	0,30	526,98	228,77	0,00
$BK = 2, RG = 0,0$, Resueltos 10, Expandiendo 51 estados 0						
	No poda			Poda		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	79938,80	32120,20	8,00	9569,50	3899,80	2,30
Desviación	227965,40	90915,01	23,02	23108,24	9260,68	5,92
$BK = 1, RG = 0,1$, Resueltos 10, Expandiendo 51 estados 6						
	No poda			Poda		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	187,30	89,00	0,00	181,70	86,30	0,00
Desviación	236,45	106,12	0,00	219,67	98,03	0,00
$BK = 2, RG = 0,1$, Resueltos 10, Expandiendo 51 estados 2						
	No poda			Poda		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	116921,10	45265,50	11,50	8446,40	3426,50	2,30
Desviación	320565,01	121424,37	31,54	17639,86	6843,43	5,64
$BK = 1, RG = 0,2$, Resueltos 10, Expandiendo 51 estados 6						
	No poda			Poda		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	219,10	115,20	0,00	153,90	79,60	0,00
Desviación	311,05	169,12	0,00	117,28	63,21	0,00
$BK = 2, RG = 0,2$, Resueltos 10, Expandiendo 51 estados 4						
	No poda			Poda		
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	1546,00	715,70	0,20	626,50	276,70	0,10
Desviación	3334,82	1497,67	0,40	1023,94	423,41	0,30

4.11, todas las instancias son resueltas óptimamente. De esos resultados podemos concluir que los problemas con 2 cuellos de botella son más difíciles de resolver que los que sólo tienen uno. Sin embargo, parece que el parámetro RG no es significativo desde el punto de vista de la dificultad del problema. En el trabajo [74], Sadeh y Fox muestran resultados de aplicar un algoritmo de backtracking guiado por un heurístico de ordenación de variables y valores diseñado para hacer frente a problemas con recursos que son cuellos de botella. En este estudio, no buscan soluciones óptimas sino soluciones que no sean peores que un 20 % de la solución óptima. Incluso con este límite, la búsqueda sólo alcanzó una solución factible en 52 de los 60 problemas, siendo el tiempo medio empleado para resolverlos de unos 2 segundos en un DECstation5000/200. Como se observa, el algoritmo A^* encuentra la solución óptima para todos los problemas sin poda, necesitando cuando emplea la técnica de poda menor número de nodos expandidos y tiempo para alcanzar la solución. Además, en los casos en los que $BK = 2$, la poda tiene mayor efecto. Por otro lado, en ambos casos hay una serie de problemas que se resuelven expandiendo 51 nodos, el número mínimo de nodos que se deben expandir para llegar a una solución.

4.7.4. Comparación con el Algoritmo de Brucker

El algoritmo de ramificación y poda propuesto por P. Brucker, B. Jurisch y B. Sievers en [12] es la mejor aproximación conocida hasta el momento para resolver de forma exacta el problema $J||C_{max}$. En consecuencia, cualquier nuevo método debe ser comparado con este algoritmo. No es necesario realizar un estudio experimental exhaustivo para demostrar que el algoritmo de Brucker sigue siendo la mejor aproximación, ya que es capaz de resolver en pocos segundos cualquier instancia de tamaño 10×10 o 20×5, e incluso algunas mayores. En la *OR-library* solamente hemos encontrado una instancia, el problema $FT20$, que nuestra aproximación resuelve en pocos segundos, mientras que el algoritmo de Brucker no es capaz de resolver en tres días de ejecución.

Como hemos indicado en el Capítulo (2), el algoritmo de Brucker recorre un espacio de búsqueda que solamente es completo si la función objetivo es el makespan. El esquema de ramificación de este algoritmo se basa en el concepto de camino crítico y en la práctica este esquema no es generalizable de forma eficiente a otras funciones objetivo. Esto mismo ocurre con otras aproximaciones, en general no exactas, como los métodos de búsqueda local que son muy eficientes para el problema $J||C_{max}$. Las estrategias de vecindad de estos métodos ([24, 89, 60, 51, 60, 94]) se basan en intercambiar arcos en el camino crítico y tampoco son fáciles de generalizar cuando la función objetivo es diferente del makespan.

Sin embargo la aproximación que proponemos aquí recorre un espacio de búsqueda que es completo para las funciones objetivo regulares, como por ejemplo el tiempo de flujo total, el tardiness, el lateness, etc. Desde este punto de vista, nuestra aproximación tiene la ventaja de que se puede adaptar a otras versiones del problema. Para ello solamente hay que cambiar el heurístico y adaptar el método de poda por dominancia a la nueva función objetivo. Esto es lo que se hace en el Capítulo (5), en dónde consideramos el problema JSS con minimización del tiempo de flujo total.

4.8. Conclusiones

En este capítulo hemos formulado el problema JSS con minimización del makespan, $J||C_{max}$, en el marco de la búsqueda en espacios de estados, en concreto en el marco del algoritmo A^*. Para ello hemos utilizado el espacio de búsqueda de las planificaciones activas, así como una serie de heurísticos admisibles y monótonos basados en distintas relajaciones del problema. Como aportación principal, hemos formulado y descrito una técnica de poda por dominancia que permite reducir el espacio de búsqueda efectivo, así como el tiempo de ejecución, y la hemos aplicado a esta versión del problema. Por último, hemos presentado los resultados de un estudio experimental en el que se comparan los heurísticos, se analiza la eficiencia de distintas versiones de la técnica de poda por dominancia y se muestra el umbral del tamaño de las instancias que el método permite resolver.

Los resultados experimentales muestran que el mejor de los heurísticos para esta versión del problema es el heurístico h_3 que, como hemos visto, se basa en el cálculo de planificaciones tipo JPS. Hemos comprobado que el límite del tamaño de los problemas que podemos resolver óptimamente, gracias al empleo de la técnica de poda por dominancia, se acerca al umbral de tamaño que deja de considerarse pequeño, es decir 10×10. Nuestra aproximación no alcanza la eficiencia del algoritmo de Brucker, que sigue siendo la mejor aproximación conocida hasta el momento. Sin embargo, es más eficiente que otras aproximaciones que, como la nuestra, no se basan en el camino crítico y que por lo tanto son generalizables a otras funciones objetivo. Este es el caso, por ejemplo, del algoritmo de backtracking guiado con heurísticos de ordenación de variables y valores propuesto en [74]. Lo esencial de estos resultados se recoge en [77, 78, 81].

En el capítulo siguiente proponemos la generalización del método a otra versión del problema, el JSS con minimización del flujo total. Como veremos, al cambiar la función objetivo el problema es más costoso de resolver, y el método que proponemos es realmente competitivo con otros métodos actuales para el mismo problema.

4.9. Resumen Notación

- $J = \{J_1, \ldots, J_N\}$, Conjunto de N trabajos.
- $R = \{R_1, \ldots, R_M\}$, Conjunto de M recursos o máquinas físicas.
- $J_i = \{\theta_{i1}, \ldots, \theta_{iM}\}$, Conjunto de tareas u operaciones de que consta el trabajo J_i.
- $R_{\theta_{il}}$, Máquina que requiere la tarea θ_{il}.
- $p_{\theta_{il}}$, Duración de la tarea θ_{il}.
- $r_{\theta_{il}}$, Cabeza de la tarea θ_{il}.
- $q_{\theta_{il}}$, Cola de la tarea θ_{il}.
- $st_{\theta_{il}}$, Tiempo de inicio de la tarea θ_{il}.
- PM_v, Predecesor de la tarea v en la secuencia de su máquina.
- SM_v, Sucesor de la tarea v en la secuencia de su máquina.
- PJ_v, Predecesor de la tarea v en la secuencia de su trabajo.
- SJ_v, Sucesor de la tarea v en la secuencia de su trabajo.
- $P(v)$, Predecesores disyuntivos de la tarea v, tareas que requieren la misma máquina que v y están planificadas antes que ella.
- $S(v)$, Sucesores disyuntivos de la tarea v, tareas que requieren la misma máquina que v y están planificadas después que ella.
- $SC(n)$, Tareas planificadas en el estado n.
- $US(n)$, Tareas sin planificar en el estado n.
- $P(n) = (US(n), r_n(US(n)), q_n(US(n)))$, Problema residual representado por las tareas sin planificar en el estado n,sus cabezas y sus colas.
- $J_{US}(n) = \{J_i \in J; \exists l, 1 \leq l \leq M, \theta_{il} \in US(n)\}$, Trabajos con tareas sin planificar en el estado n.
- $J_{SC}(n) = J \backslash J_{US}(n)$, Trabajos con todas sus tareas planificadas en el estado n.
- $C_{max}(J_{SC}(n)) = max\{r_{\theta_{iM}}(n) + p_{\theta_{iM}}, J_i \in J_{SC}(n)\}$, Tiempo de completud máximo de los trabajos en $J_{SC}(n)$.
- $C_{max}(J_{US}(n))$, Coste óptimo de planificar las tareas $US(n)$ a partir del estado n.
- $P(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m),\ \mathbf{p}_n(US(n)|_m),\ \mathbf{q}_n(US(n)|_m))$, Problema OMS para la máquina m con cabezas y colas, y minimización del makespan. Donde $US(n)|_m$ es el conjunto de tareas sin planificar en el estado n que requieren la máquina m.

- $JPS(J_{US}(n)) = JPS(P(n)) = max_{m \in R} JPS(P(n)|_m)$.
- $C_{max}(J_{US}(n)) = C_{max}(P(n))$, Coste óptimo de $P(n)$.
- $f(n) = \text{máx}\{C_{max}(J_{SC}(n)), JPS(P(n))\}$.
- $f^*(n) = \text{máx}\{C_{max}(J_{SC}(n)), C_{max}(P(n))\}$, $JPS(P(n)) \leq C_{max}(P(n))$.
- $g(n) = \text{máx}(C_{max}(J_{SC}(n)), \text{máx}_{J_i \in J_{US}(n)}(\text{mín}\{r_{\theta_{ij}}, \theta_{ij} \in US(n)\})) = C_{max}(n)$.
- $P(n_2 \setminus n_1) = (US(n_1), \mathbf{r}_{n_2}(US(n_1)), \mathbf{q}_{n_2}(US(n_1)))$, Tiene que darse $US(n_1) \subseteq US(n_2)$.
- $C_{max}(P(n_2 \setminus n_1))$, Coste óptimo de $P(n_2 \setminus n_1)$.

4.10. Anexo. Tablas

4.10.1. Heurístico h_3

Tabla 4.12: Resultados para una serie de problemas de tamaño 6×6: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 6 \times 6$	693	357	0	456	245	0
$LA17 - 6 \times 6$	98	58	0	98	58	0
$LA18 - 6 \times 6$	844	548	0	357	227	0
$LA19 - 6 \times 6$	262	157	0	170	103	0
$LA20 - 6 \times 6$	110	69	0	104	65	0
$ORB01 - 6 \times 6$	3726	2240	0	1671	979	0
$ORB02 - 6 \times 6$	1047	622	1	634	373	0
$ORB03 - 6 \times 6$	786	415	0	481	257	0
$ORB04 - 6 \times 6$	5776	3328	0	1451	834	0
$ORB05 - 6 \times 6$	66	41	0	66	41	0
$ORB06 - 6 \times 6$	224	118	0	193	101	0
$ORB07 - 6 \times 6$	2946	1720	0	1842	1075	0
$ORB08 - 6 \times 6$	5473	3210	1	1663	934	1
$ORB09 - 6 \times 6$	1327	815	0	838	510	0
$ORB10 - 6 \times 6$	910	582	0	359	223	0
$ABZ5 - 6 \times 6$	751	409	0	509	273	0
$ABZ6 - 6 \times 6$	134	85	0	114	72	0
$ABZ7 - 6 \times 6$	648	375	0	410	238	0
$ABZ8 - 6 \times 6$	317	200	0	207	131	0
$ABZ9 - 6 \times 6$	249	158	0	190	125	0
$FT10 - 6 \times 6$	3509	1847	0	810	443	0
$LA21 - 6 \times 6$	345	214	0	267	164	0
$LA24 - 6 \times 6$	194	111	0	176	100	0
$LA25 - 6 \times 6$	708	406	0	473	275	0
$LA27 - 6 \times 6$	196	134	0	127	85	0
$LA29 - 6 \times 6$	1323	815	0	465	285	0
$LA38 - 6 \times 6$	4540	2728	1	1406	835	0
$LA40 - 6 \times 6$	1010	478	0	437	224	0
$YN1 - 6 \times 6$	264	154	0	183	106	0
$YN2 - 6 \times 6$	470	229	0	507	254	0
$YN3 - 6 \times 6$	544	307	0	387	222	1
$YN4 - 6 \times 6$	398	251	0	239	152	0
Media	1246,50	724,41	0,09	540,31	312,78	0,06
Desviación	1582,11	921,99	0,29	500,12	288,85	0,24

Tabla 4.13: Resultados para una serie de problemas de tamaño 7×7: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 7 \times 7$	29780	17166	3	7360	4190	1
$LA17 - 7 \times 7$	9799	5662	1	2023	1135	0
$LA18 - 7 \times 7$	1868	1158	0	962	589	0
$LA19 - 7 \times 7$	3354	1927	0	1785	1021	0
$LA20 - 7 \times 7$	4628	2706	1	1463	887	0
$ORB01 - 7 \times 7$	5111	2618	0	2507	1282	0
$ORB02 - 7 \times 7$	4585	2694	1	1963	1129	0
$ORB03 - 7 \times 7$	2766	1580	0	1583	858	1
$ORB04 - 7 \times 7$	2473	1340	0	1067	559	0
$ORB05 - 7 \times 7$	3619	2313	1	1111	724	0
$ORB06 - 7 \times 7$	27757	16184	2	5474	3075	1
$ORB07 - 7 \times 7$	6133	3189	0	2682	1367	0
$ORB08 - 7 \times 7$	6925	3705	1	2876	1486	0
$ORB09 - 7 \times 7$	7876	4557	1	4009	2250	1
$ORB10 - 7 \times 7$	517	330	0	390	253	0
$ABZ5 - 7 \times 7$	7333	4385	0	1586	834	0
$ABZ6 - 7 \times 7$	1002	617	0	668	409	0
$ABZ7 - 7 \times 7$	7797	4256	0	3277	1759	0
$ABZ8 - 7 \times 7$	1889	1057	1	1242	689	0
$ABZ9 - 7 \times 7$	763	449	0	591	350	0
$FT10 - 7 \times 7$	29435	16490	3	4673	2582	1
$LA21 - 7 \times 7$	2121	1242	0	1164	694	0
$LA24 - 7 \times 7$	2036	1113	0	1110	632	0
$LA25 - 7 \times 7$	12297	7971	1	1872	1088	0
$LA27 - 7 \times 7$	3125	1907	0	1199	736	1
$LA29 - 7 \times 7$	6985	3743	1	2656	1449	0
$LA38 - 7 \times 7$	4160	2569	1	1477	894	0
$LA40 - 7 \times 7$	3015	1618	0	1030	576	0
$YN1 - 7 \times 7$	824	453	0	604	331	0
$YN2 - 7 \times 7$	8148	4874	1	2528	1431	1
$YN3 - 7 \times 7$	2198	1382	0	1044	668	0
$YN4 - 7 \times 7$	1379	757	0	915	501	0
Media	6615,56	3812,88	0,59	2027,84	1138,38	0,22
Desviación	7737,10	4454,37	0,82	1515,89	840,65	0,41

Tabla 4.14: Resultados para una serie de problemas de tamaño 8×8: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 8 \times 8$	423897	242946	51	45383	26127	11
$LA17 - 8 \times 8$	44321	25028	6	6838	3889	1
$LA18 - 8 \times 8$	12017	7400	1	2601	1584	0
$LA19 - 8 \times 8$	16255	9818	2	4459	2670	1
$LA20 - 8 \times 8$	2366	1387	0	1020	610	0
$ORB01 - 8 \times 8$	439788	230303	56	62415	32506	12
$ORB02 - 8 \times 8$	19484	11302	2	8226	4741	1
$ORB03 - 8 \times 8$	318427	166692	42	46740	24215	10
$ORB04 - 8 \times 8$	173353	92323	22	34376	18135	7
$ORB05 - 8 \times 8$	441320	244602	53	39700	21569	7
$ORB06 - 8 \times 8$	437471	235985	53	45298	23817	10
$ORB07 - 8 \times 8$	235768	133565	29	34383	19118	11
$ORB08 - 8 \times 8$	**3682481**	**1995880**	**433**	107637	58099	35
$ORB09 - 8 \times 8$	63279	37138	8	15142	8352	2
$ORB10 - 8 \times 8$	113593	69309	13	13252	7728	2
$ABZ5 - 8 \times 8$	32067	16379	4	8521	4231	1
$ABZ6 - 8 \times 8$	84753	49904	9	15696	9451	3
$ABZ7 - 8 \times 8$	152643	82546	18	23540	12769	5
$ABZ8 - 8 \times 8$	23846	13347	3	7337	4093	1
$ABZ9 - 8 \times 8$	18552	10699	2	8542	4950	1
$FT10 - 8 \times 8$	267615	147000	37	29411	15654	6
$LA21 - 8 \times 8$	1526	846	1	1136	625	0
$LA24 - 8 \times 8$	23082	13723	2	4035	2411	1
$LA25 - 8 \times 8$	175887	100076	21	22960	12601	4
$LA27 - 8 \times 8$	37122	21702	4	7409	4407	1
$LA29 - 8 \times 8$	41279	23092	5	7360	4118	1
$LA38 - 8 \times 8$	11380	6570	2	5253	3024	1
$LA40 - 8 \times 8$	651	359	0	553	308	0
$YN1 - 8 \times 8$	53370	31170	6	9272	5459	2
$YN2 - 8 \times 8$	18769	10995	3	4844	2830	1
$YN3 - 8 \times 8$	15602	9320	1	5553	3264	0
$YN4 - 8 \times 8$	49864	28286	7	7088	3807	1
Media	120946,68	66897,16	14,94	17043,32	9324,61	3,35
Desviación	145594,47	79269,65	18,14	16472,05	8733,14	3,79

Tabla 4.15: Resultados para una serie de problemas de tamaño 9×9: problemas recortados de las baterías LA, ORB, ABZ, YN y *Selectos*

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16-9\times9$	**3006094**	**1662931**	**410**	430010	245730	650
$LA17-9\times9$	1380284	765716	176	59846	33738	15
$LA18-9\times9$	1050061	584480	141	102092	57077	32
$LA19-9\times9$	129805	73635	17	18174	10305	3
$LA20-9\times9$	226569	123200	28	25510	14678	5
$ORB01-9\times9$	408090	229245	56	67308	36043	19
$ORB02-9\times9$	497619	268186	65	57020	31714	14
$ORB03-9\times9$	**2963579**	**1620453**	**434**	1184877	640289	2991
$ORB04-9\times9$	1313326	705060	191	154896	79629	56
$ORB05-9\times9$	**3148347**	**1805215**	**422**	1074224	617081	1868
$ORB06-9\times9$	**2664821**	**1321679**	**399**	358438	182174	232
$ORB07-9\times9$	992760	561617	142	132827	74528	83
$ORB08-9\times9$	**2766342**	**1423274**	**397**	**1101716**	**595319**	**3600**
$ORB09-9\times9$	**3214588**	**1871399**	**441**	481566	272595	474
$ORB10-9\times9$	1893382	1010224	267	191616	106407	79
$ABZ5-9\times9$	257149	142800	35	34673	18737	7
$ABZ6-9\times9$	151280	87783	21	36628	21243	9
$ABZ7-9\times9$	884375	497722	116	90123	50622	35
$ABZ8-9\times9$	563928	328581	76	66924	38408	29
$ABZ9-9\times9$	420275	242998	51	41444	23180	11
$FT10-9\times9$	**2714288**	**1371098**	**410**	316139	160042	196
$LA21-9\times9$	**3446734**	**2103504**	**412**	134085	81836	51
$LA24-9\times9$	746769	445127	95	40188	23823	10
$LA25-9\times9$	2392620	1353514	321	86356	47562	23
$LA27-9\times9$	333685	184986	46	66401	36588	19
$LA29-9\times9$	335230	186554	45	36120	19942	8
$LA38-9\times9$	139506	81886	18	38785	22531	8
$LA40-9\times9$	**3081519**	**1738323**	**393**	63175	35341	15
$YN1-9\times9$	38694	22326	5	8738	4993	2
$YN2-9\times9$	91678	50969	13	23077	12835	5
$YN3-9\times9$	289701	169861	37	55377	31744	15
$YN4-9\times9$	397137	228620	51	21198	11914	5
Media	678814,68	379322,27	91,50	66150,95	36738,23	22,36
Desviación	687502,82	379322,37	95,70	61524,24	33170,66	30,24
Resolvemos $ORB08-9\times9$ con Poda y sin límite de tiempo						
$ORB08-9\times9$				1903538	1050179	9861

Tabla 4.16: Resultados para una serie de problemas de tamaño 10×10: problemas de las baterías LA, ABZ, ORB, problema $FT10$ y problemas recortados de las baterías YN y *Selectos*

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16$	**2189738**	**1072798**	**371**	**818976**	**439998**	**3600**
$LA17$	**2334718**	**1242474**	**365**	**1276974**	**720585**	**2974**
$LA18$	**2306216**	**1213747**	**385**	881218	493813	1651
$LA19$	**2422259**	**1330026**	**391**	**1189636**	**661035**	**3206**
$LA20$	**2630258**	**1538057**	**410**	1001920	591080	2243
$ABZ5$	**2382172**	**1269354**	**412**	**1219604**	**687467**	**2300**
$ABZ6$	**2499666**	**1407500**	**370**	239746	139661	217
$ORB01$	**2221395**	**1129033**	**393**	**1183989**	**616592**	**1735**
$ORB02$	**2448412**	**1356115**	**413**	**1215573**	**683649**	**2281**
$ORB03$	**1904133**	**811715**	**360**	**1234631**	**566452**	**2825**
$ORB04$	**1912207**	**819555**	**348**	**1187851**	**587788**	**1833**
$ORB05$	**2281810**	**1189598**	**393**	**1197710**	**644165**	**2802**
$ORB06$	**2258905**	**1166612**	**406**	**1221615**	**637606**	**2054**
$ORB07$	**2447000**	**1354772**	**427**	679478	383895	1764
$ORB08$	**2503804**	**1411482**	**387**	**1179879**	**671119**	**2245**
$ORB09$	**2224277**	**1131754**	**389**	**1179127**	**602524**	**2539**
$ORB10$	**2424452**	**1332338**	**400**	170077	94209	100
$FT10$	**1927594**	**835007**	**352**	**1211856**	**579451**	**2194**
$ABZ7 - 10 \times 10$	178726	107090	28	28182	16395	7
$ABZ8 - 10 \times 10$	**2623452**	**1531300**	**430**	312321	179936	587
$ABZ9 - 10 \times 10$	330494	182398	54	50336	27473	16
$LA21 - 10 \times 10$	**2783335**	**1691145**	**426**	271414	158094	191
$LA24 - 10 \times 10$	**2570899**	**1478722**	**412**	168353	97457	97
$LA25 - 10 \times 10$	**2480822**	**1388516**	**402**	614034	343265	770
$LA27 - 10 \times 10$	**684954**	**1592787**	**431**	301262	178193	265
$LA29 - 10 \times 10$	1415569	766457	233	71137	37758	23
$LA38 - 10 \times 10$	61154	33330	10	17043	9295	4
$LA40 - 10 \times 10$	**2606201**	**1514016**	**407**	**1227745**	**716219**	**2235**
$YN1 - 10 \times 10$	**2518627**	**1426332**	**393**	166482	96773	125
$YN2 - 10 \times 10$	**2544767**	**1452568**	**399**	109654	62821	58
$YN3 - 10 \times 10$	**2466518**	**1374312**	**395**	659297	368375	1766
$YN4 - 10 \times 10$	226996	125110	38	27424	15056	8
Media	442587,80	242877,00	72,60	38824,40	21195,40	11,60
Desviación	494136,02	266086,77	81,46	19464,22	10159,56	6,95

Tabla 4.17: Resultados para el único problema resuelto de la batería *Selectos*: el $FT20$

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$FT20$	19520	9014	4	6369	2756	2

Tabla 4.18: Resultados para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 1$)

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*01	103	51	0	103	51	0
*Sadeh*02	115	51	0	115	51	0
*Sadeh*03	113	51	0	113	51	0
*Sadeh*04	114	57	0	111	55	0
*Sadeh*05	113	51	0	113	51	0
*Sadeh*06	109	51	0	109	51	0
*Sadeh*07	2574	1130	1	1868	814	0
*Sadeh*08	117	51	0	117	51	0
*Sadeh*09	109	51	0	109	51	0
*Sadeh*10	113	51	0	113	51	0
Media	358,00	159,50	0,10	287,10	127,70	0,00
Desviación	738,68	323,50	0,30	526,98	228,77	0,00

Tabla 4.19: Resultados para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 2$)

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*11	149	70	0	149	70	0
*Sadeh*12	746	363	0	624	300	0
*Sadeh*13	2018	869	0	1207	491	1
*Sadeh*14	1170	446	0	776	293	0
*Sadeh*15	407	156	0	399	152	0
*Sadeh*16	329	164	0	313	156	0
*Sadeh*17	125	57	0	125	57	0
*Sadeh*18	29122	13772	3	10895	5015	1
*Sadeh*19	1957	705	0	2941	1114	1
*Sadeh*20	763365	304600	77	78266	31350	20
Media	79938,80	32120,20	8,00	9569,50	3899,80	2,30
Desviación	227965,40	90915,01	23,02	23108,24	9260,68	5,92

Tabla 4.20: Resultados para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 1$)

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*21	97	51	0	97	51	0
*Sadeh*22	108	51	0	108	51	0
*Sadeh*23	107	51	0	107	51	0
*Sadeh*24	104	54	0	104	54	0
*Sadeh*25	110	51	0	110	51	0
*Sadeh*26	103	51	0	103	51	0
*Sadeh*27	896	407	0	840	380	0
*Sadeh*28	137	68	0	137	68	0
*Sadeh*29	109	55	0	109	55	0
*Sadeh*30	102	51	0	102	51	0
Media	187,30	89,00	0,00	181,70	86,30	0,00
Desviación	236,45	106,12	0,00	219,67	98,03	0,00

Tabla 4.21: Resultados para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 2$)

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*31	109	51	0	109	51	1
*Sadeh*32	143	63	0	143	63	0
*Sadeh*33	256	104	0	256	104	0
*Sadeh*34	7155	3721	1	1696	794	0
*Sadeh*35	1907	664	0	1748	603	0
*Sadeh*36	28869	13424	3	2117	933	0
*Sadeh*37	119	54	0	119	54	0
*Sadeh*38	53226	25782	5	19589	9318	3
*Sadeh*39	114	51	0	114	51	0
*Sadeh*40	1077313	408741	106	58573	22294	19
Media	116921,10	45265,50	11,50	8446,40	3426,50	2,30
Desviación	320565,01	121424,37	31,54	17639,86	6843,43	5,64

Tabla 4.22: Resultados para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 1$)

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*41	97	51	0	97	51	0
*Sadeh*42	102	51	0	102	51	0
*Sadeh*43	102	51	0	102	51	0
*Sadeh*44	1149	621	0	497	265	0
*Sadeh*45	109	51	0	109	51	0
*Sadeh*46	111	57	0	111	57	0
*Sadeh*47	186	92	0	186	92	0
*Sadeh*48	137	76	0	137	76	0
*Sadeh*49	99	51	0	99	51	0
*Sadeh*50	99	51	0	99	51	0
Media	219,10	115,20	0,00	153,90	79,60	0,00
Desviación	311,05	169,12	0,00	117,28	63,21	0,00

Tabla 4.23: Resultados para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 2$)

	No poda			Poda		
Instancia	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*51	105	51	0	105	51	0
*Sadeh*52	117	57	0	117	57	0
*Sadeh*53	242	103	0	242	103	0
*Sadeh*54	104	51	0	104	51	0
*Sadeh*55	102	51	0	102	51	0
*Sadeh*56	2003	991	0	968	454	1
*Sadeh*57	322	152	1	322	152	0
*Sadeh*58	959	527	0	610	311	0
*Sadeh*59	106	51	0	106	51	0
*Sadeh*60	11400	5123	1	3589	1486	0
Media	1546,00	715,70	0,20	626,50	276,70	0,10
Desviación	3334,82	1497,67	0,40	1023,94	423,41	0,30

Tabla 4.24: Cotas Inferiores (LB) y Máximas profundidades (MP) alcanzadas para los problemas no resueltos

	No poda		Poda	
Instancia	LB	MP	LB	MP
$LA16$	901	42	921	59
$LA17$	748	64	779	83
$LA19$	823	67	840	92
$ABZ5$	1185	60	1217	78
$ORB01$	1037	57	1052	87
$ORB02$	854	61	873	73
$ORB03$	954	41	975	54
$ORB04$	967	34	991	62
$ORB05$	852	65	870	77
$ORB06$	981	54	999	66
$ORB08$	894	80	899	91
$ORB09$	917	52	928	71
$ABZ7$	650	121	650	121
$ABZ8$	623	55	623	56
$ABZ9$	626	41	626	41
$FT10$	889	36	915	86
$LA21$	1033	80	1033	84
$LA24$	904	63	911	70
$LA25$	925	49	934	79
$LA27$	1235	75	1235	75
$LA29$	1114	47	1114	54
$LA38$	1099	74	1105	74
$LA40$	1197	91	1200	91
$YN1$	777	63	777	63
$YN2$	802	72	802	74
$YN3$	793	90	793	90
$YN4$	871	46	871	48
$LA40 - 10 \times 10$	738	70	750	91

4.10.2. Soluciones y Cotas Inferiores

En las dos tablas siguientes (Tablas 4.25 y 4.26) se muestra un resumen de los valores de las soluciones óptimas o de las mejores cotas inferiores obtenidas para los problemas utilizados en el estudio experimental, empleando el heurístico h_3. Los valores en negrita indican cotas inferiores y a continuación se indica si se han obtenido con o sin la técnica de poda (cuando se obtiene con ambos no se pone nada).

La Tabla 4.25 se refiere a los problemas originales y la Tabla 4.26 a los problemas recortados. En la Tabla 4.25 se muestra además, la solución óptima o en su defecto la mejor solución conocida en la literatura (en este caso se indica con un $*$ delante de ella). El único problema recortado no resuelto es el $LA40 - 10 \times 10$, para este problema la mejor cota LB alcanzada es 750, siendo su solución óptima 755 (resuelto con el algoritmo de ramificación y poda propuesto por P. Brucker, B. Jurisch y B. Sievers en [12]).

Tabla 4.25: Problemas Baterías: Mejores soluciones en la literatura, Soluciones alcanzadas y Cotas inferiores (en negrita)

Tamaño	Ins.	Opt./Mejor	Sol./LB	Tamaño	Instancia	Opt./Mejor	Sol./LB
6×6	$FT06$	55	55	10×10	$ORB01$	1059	**1052, P**
10×5	$LA01$	666	666		$ORB02$	888	**873, P**
	$LA02$	655	655		$ORB03$	1005	**975, P**
	$LA03$	597	597		$ORB04$	1005	**991, P**
	$LA04$	590	590		$ORB05$	887	**870, P**
	$LA05$	593	593		$ORB06$	1010	**999, P**
15×5	$LA06$	926	926		$ORB07$	397	397
	$LA07$	890	890		$ORB08$	899	**899, P**
	$LA08$	863	863		$ORB09$	934	**928, P**
	$LA09$	951	951		$ORB10$	944	944
	$LA10$	958	958	20×5	$FT10$	930	**915, P**
20×5	$LA11$	1222	1222	15×10	$FT20$	1165	1165
	$LA12$	1039	1039		$LA21$	1046	**1033, P**
	$LA13$	1150	1150		$LA24$	935	**911, P**
	$LA14$	1292	1292	20×10	$LA25$	977	**934, P**
	$LA15$	1207	1207		$LA27$	1235	**1235, P**
10×10	$LA16$	945	**921**	15×15	$LA29$	*1153	**1114, P**
	$LA17$	784	**779**		$LA38$	1196	**1105, P**
	$LA18$	848	848	20×15	$LA40$	1222	**1200, P**
	$LA19$	842	**840**		$ABZ7$	*665	**650, P**
	$LA20$	902	902		$ABZ8$	*670	**623, P**
	$ABZ5$	1234	**1217, P**	20×20	$ABZ9$	*686	**626, P**
	$ABZ6$	943	943		$YN1$	*888	**777, P**
					$YN2$	*912	**802, P**
					$YN3$	*898	**793, P**
					$YN4$	*977	**871, P**

Tabla 4.26: Problemas Recortados: Soluciones y Cotas inferiores (en negrita)

Inst.	Sol.	Ins.	Sol.	Inst.	Sol.	Inst.	Sol.
$LA16-6\times6$	579	$LA20-7\times7$	613	$ORB04-8\times8$	798	$ORB08-9\times9$	812
$LA17-6\times6$	473	$ORB01-7\times7$	667	$ORB05-8\times8$	681	$ORB09-9\times9$	865
$LA18-6\times6$	469	$ORB02-7\times7$	638	$ORB06-8\times8$	761	$ORB10-9\times9$	867
$LA19-6\times6$	525	$ORB03-7\times7$	704	$ORB07-8\times8$	323	$ABZ5-9\times9$	1086
$LA20-6\times6$	462	$ORB04-7\times7$	673	$ORB08-8\times8$	771	$ABZ6-9\times9$	898
$ORB01-6\times6$	549	$ORB05-7\times7$	506	$ORB09-8\times8$	714	$ABZ7-9\times9$	362
$ORB02-6\times6$	561	$ORB06-7\times7$	628	$ORB10-8\times8$	794	$ABZ8-9\times9$	384
$ORB03-6\times6$	636	$ORB07-7\times7$	293	$ABZ5-8\times8$	985	$ABZ9-9\times9$	374
$ORB04-6\times6$	617	$ORB08-7\times7$	658	$ABZ6-8\times8$	801	$FT10-9\times9$	855
$ORB05-6\times6$	444	$ORB09-7\times7$	634	$ABZ7-8\times8$	335	$LA21-9\times9$	728
$ORB06-6\times6$	533	$ORB10-7\times7$	639	$ABZ8-8\times8$	357	$LA24-9\times9$	744
$ORB07-6\times6$	258	$ABZ5-7\times7$	874	$ABZ9-8\times8$	334	$LA25-9\times9$	797
$ORB08-6\times6$	504	$ABZ6-7\times7$	683	$FT10-8\times8$	737	$LA27-9\times9$	751
$ORB09-6\times6$	540	$ABZ7-7\times7$	297	$LA21-8\times8$	657	$LA29-9\times9$	758
$ORB10-6\times6$	593	$ABZ8-7\times7$	311	$LA24-8\times8$	680	$LA38-9\times9$	801
$ABZ5-6\times6$	744	$ABZ9-7\times7$	278	$LA25-8\times8$	724	$LA40-9\times9$	698
$ABZ6-6\times6$	615	$FT10-7\times7$	653	$LA27-8\times8$	669	$YN1-9\times9$	394
$ABZ7-6\times6$	222	$LA21-7\times7$	618	$LA29-8\times8$	688	$YN2-9\times9$	403
$ABZ8-6\times6$	261	$LA24-7\times7$	604	$LA38-8\times8$	738	$YN3-9\times9$	412
$ABZ9-6\times6$	235	$LA25-7\times7$	627	$LA40-8\times8$	664	$YN4-9\times9$	412
$FT10-6\times6$	594	$LA27-7\times7$	564	$YN1-8\times8$	358	$ABZ7-10\times10$	379
$LA21-6\times6$	528	$LA29-7\times7$	593	$YN2-8\times8$	369	$ABZ8-10\times10$	399
$LA24-6\times6$	543	$LA38-7\times7$	624	$YN3-8\times8$	343	$ABZ9-10\times10$	418
$LA25-6\times6$	532	$LA40-7\times7$	596	$YN4-8\times8$	379	$LA21-10\times10$	847
$LA27-6\times6$	495	$YN1-7\times7$	327	$LA16-9\times9$	780	$LA24-10\times10$	804
$LA29-6\times6$	541	$YN2-7\times7$	334	$LA17-9\times9$	700	$LA25-10\times10$	846
$LA38-6\times6$	572	$YN3-7\times7$	305	$LA18-9\times9$	772	$LA27-10\times10$	805
$LA40-6\times6$	497	$YN4-7\times7$	316	$LA19-9\times9$	766	$LA29-10\times10$	803
$YN1-6\times6$	288	$LA16-8\times8$	717	$LA20-9\times9$	805	$LA38-10\times10$	857
$YN2-6\times6$	304	$LA17-8\times8$	638	$ORB01-9\times9$	886	$LA40-10\times10$	**750, P**
$YN3-6\times6$	278	$LA18-8\times8$	677	$ORB02-9\times9$	793	$YN1-10\times10$	440
$YN4-6\times6$	269	$LA19-8\times8$	691	$ORB03-9\times9$	902	$YN2-10\times10$	463
$LA16-7\times7$	679	$LA20-8\times8$	684	$ORB04-9\times9$	901	$YN3-10\times10$	454
$LA17-7\times7$	601	$ORB01-8\times8$	787	$ORB05-9\times9$	831	$YN4-10\times10$	466
$LA18-7\times7$	567	$ORB02-8\times8$	726	$ORB06-9\times9$	853		
$LA19-7\times7$	640	$ORB03-8\times8$	809	$ORB07-9\times9$	356		

Capítulo 5

JOB SHOP SCHEDULING CON MINIMIZACIÓN DEL FLUJO TOTAL

5.1. Introducción

En este capítulo abordamos el problema Job Shop Scheduling con minimización del tiempo de flujo total, denotado como $J||\sum C_i$. La aproximación que proponemos consiste en generalizar las ideas desarrolladas en el capítulo anterior para el mismo problema con minimización del makespan. Como ya hemos indicado en capítulos anteriores, prácticamente la totalidad de las propuestas que se pueden encontrar en la literatura para resolver el problema de minimización del makespan se basan en el concepto de camino crítico. Se han hecho también algunos intentos de generalizar estos métodos a otras funciones objetivo, pero en la práctica no resultan tan eficientes. Un ejemplo es el método que propone Kreipl en [47] para minimizar el tardiness ponderado. Se trata de un método de búsqueda local que utiliza una estrategia de vecindad que a su vez generaliza a otras propuestas debidas a E. Taillard en [84] y Dell' Amico y Trubian en [24]. Estas estrategias se basan en calcular un camino crítico y a continuación proponer cambios en el sentido de algunos de los arcos de este camino. Estos cambios se aceptan siempre y cuando cumplan una serie de propiedades que tienen que ver con la factibilidad de la solución resultante, así como con la posibilidad de mejora de esta solución. La generalización que se propone en [47] requiere redefinir el concepto de camino crítico, de modo que para la minimización del tardiness en realidad hay que hablar de varios caminos críticos, al menos tantos como el número de trabajos que exceden su due date en una solución dada. De este modo, el número de vecinos de cada solución es muy grande, y esto se traduce en que la eficiencia del método es mucho menor que los métodos análogos que se utilizan para minimizar el makespan. Otra aproximación similar es presentada por Essafi en [28]; en este caso el método de búsqueda local se combina con un algoritmo genético. Los resultados experimentales son algo mejores que los que se presentan en [47].

Otro ejemplo claro lo tenemos con el algoritmo de Brucker [12]. El espacio de búsqueda que recorre no es completo cuando se trata de minimizar otra función objetivo como el tardiness o el

flujo total. Además, si se intenta generalizar para estas funciones objetivo, en principio habría que considerar varios caminos críticos, como en el caso anterior, y el factor de ramificación sería mucho mayor. De hecho, hasta donde nosotros conocemos, no existe en la literatura ningún intento de generalizar el algoritmo de Brucker para resolver el problema JSS con funciones objetivo diferentes del makespan. Aunque los autores sí generalizaron este algoritmo para resolver el problema JSS con tiempos de setup y minimización del makespan [14].

Dado que se trata de generalizar el método propuesto en el capítulo 4, consideraremos el espacio de búsqueda de las planificaciones activas. A diferencia del espacio que recorre el algoritmo de Brucker, este espacio es completo o dominante, para las funciones objetivo regulares, como el tiempo de flujo total y el tardiness. Como estrategia de búsqueda utilizaremos también el algoritmo A^*. La principal contribución de este capítulo es afrontar la resolución del problema $J||\sum C_i$, versión que está poco estudiada en la literatura, mediante una estrategia exacta. Más concretamente, diseñamos diversas estrategias heurísticas y formalizamos el método de poda por dominancia, introducido en el capítulo anterior, al problema $J||\sum C_i$. Como veremos en el estudio experimental, el resultado es un método competitivo con los métodos actuales, en particular con el método propuesto en [47]. Este método se puede utilizar para minimizar el tiempo de flujo total, ya que esta función objetivo es un caso particular del tardiness ponderado cuando todos los pesos son iguales y los due dates son 0. Para ello utilizaremos la implementación que está disponible en el prototipo LEKIN®(http://www.stern.nyu.edu/om/software/lekin/index.htm).

Este capítulo está organizado del siguiente modo. En la sección 5.2 indicamos cómo se formula el problema y la forma de abordarlo con el algoritmo A^*; para no resultar demasiado repetitivos simplemente indicamos los cambios con respecto al problema del capítulo anterior. En la sección 5.3 se describen las diferentes estrategias heurísticas que hemos diseñado para guiar la búsqueda del algoritmo A^* en la resolución del problema. En la sección 5.4 se indica el modo de generalizar el método de poda por dominancia cuando se trata de minimizar el tiempo de flujo total. En la sección 5.5 se muestran los resultados del estudio experimental que hemos llevado a cabo para comparar los heurísticos propuestos, para analizar el comportamiento del método de poda, y para establecer comparaciones con otros métodos. Finalmente, en la sección 5.6 se resumen las principales conclusiones y aportaciones del capítulo.

5.2. Formulación del Problema $J||\sum C_i$

El problema Job Shop Scheduling con minimización del Tiempo de Flujo Total se conoce en la literatura como $J||\sum C_i$ y tiene una formulación análoga al problema $J||C_{max}$. De hecho la única diferencia entre ambos está en la función de evaluación. El problema consiste en planificar un conjunto de N trabajos $\{J_1, \ldots, J_N\}$ sobre un conjunto de M recursos o máquinas físicas $\{R_1, \ldots, R_M\}$. Cada trabajo J_i consta de un conjunto de tareas u operaciones $\{\theta_{i1}, \ldots, \theta_{iM}\}$ que han de ser planificadas secuencialmente. Cada tarea θ_{il} requiere una única máquina $R_{\theta il}$, tiene una duración fijada $p_{\theta il}$ y un tiempo de inicio $st_{\theta il}$ a determinar. Esta versión del problema también esta sometida a las restricciones de precedencia, capacidad y no-expulsión. El objetivo es encontrar una planificación factible tal que la suma del tiempo de fin de todos los trabajos, es decir el flujo total, sea mínimo.

Algoritmo 5.1 SUC(estado n). Problema $J||\sum C_i$: Algoritmo para expandir el estado n

Requiere: Estado n del problema JSS
Produce: Estados sucesores del estado n

1. $A = \{v \in US(n);\ PJ_v \in SC(n)\}$;
2. $v = \arg\min\{r_u + p_u;\ u \in A\}$;
3. $B = \{w \in A;\ R_w = R_v \text{ y } r_w < r_v + p_v\}$;

para cada $w \in B$ **hacer**

4. $SC(n') = SC(n) \cup \{w\}$ y $US(n') = US(n) \backslash \{w\}$;
5. $G_{n'} = G_n \cup \{w \to v;\ v \in US(n'),\ R_v = R_w\}$;
   {Se planifica w en el estado n′ con un tiempo de inicio r_w}
6. $c(n, n') = r_w + p_w - (r_{PJ_w} + p_{PJ_w})$;
7. Añadir n' a los sucesores;

fin para

8. Retornar sucesores;

Las instancias de esta versión del problema se representan también, como se ha indicado en el capítulo 4, mediante un grafo dirigido. Siendo una planificación factible un subgrafo sin ciclos que nos indica el orden de procesamiento de las tareas en cada máquina. El tiempo de completud del trabajo J_i, denotado como C_i, es el coste del camino más largo desde el nodo *inicial* al nodo *final*, restringido a pasar por la última tarea del trabajo J_i justo antes del nodo *final*. Para una planificación factible, las cabezas y colas se definen del mismo modo que para la versión con minimización del makespan, es decir:

$$\begin{aligned} r_v &= \text{máx}\{\text{máx}_{w \in P(v)}(r_w + p_w), r_{PJ_v} + p_{PJ_v}\} \\ q_v &= \text{máx}\{\text{máx}_{w \in S(v)}(p_w + q_w), p_{SJ_v} + q_{SJ_v}\} \end{aligned} \tag{5.1}$$

El Algoritmo 5.1 muestra la operación de expansión que genera el conjunto de planificaciones activas completo cuando es aplicado sucesivamente desde el estado inicial (planificación vacía). Este algoritmo es prácticamente igual al Algoritmo 4.1 propuesto en el capítulo 4; la única diferencia está en el paso 6, es decir en el cálculo del coste de pasar de un estado a otro.

Recordemos también la notación empleada: O denota el conjunto de tareas u operaciones de que consta un problema; n_1 y n_2 son dos estados del espacio de búsqueda, en n_1, O se puede obtener como la unión disjunta $SC(n_1) \cup US(n_1)$, es decir, la unión de tareas planificadas y no planificadas. $D(n_1) = |SC(n_1)|$ es la profundidad, en el espacio de búsqueda, del nodo n_1. Dado $O' \subseteq O$, $\boldsymbol{r}_{n_1}(O')$ es el vector de las cabezas de las tareas de O' en el estado n_1. $\boldsymbol{r}_{n_1}(O') \leq \boldsymbol{r}_{n_2}(O')$ si y sólo si para cada tarea $v \in O'$, $r_v(n_1) \leq r_v(n_2)$, siendo $r_v(n_1)$ y $r_v(n_2)$ las cabezas de la tarea v en los estados n_1 y n_2 respectivamente. Análogamente, $\boldsymbol{q}_{n_1}(O')$ es el vector de las colas de las tareas de O' en el estado n_1. Dado que ahora se trata de minimizar el tiempo de flujo total, el coste de pasar de un estado n a otro n', $c(n, n')$, se obtiene calculando la diferencia $r_w + p_w - (r_{PJ_w} + p_{PJ_w})$, donde $r_w + p_w$ es el tiempo de finalización de la tarea w planificada al pasar de n a n' y $r_{PJ_w} + p_{PJ_w}$ es el tiempo de fin de la tarea anterior a w en el trabajo.

5.3. Estrategias Heurísticas para el Problema $J||\sum C_i$

En esta sección se describen los diferentes heurísticos que hemos diseñado para guiar la búsqueda en el espacio de estados. Todos estos heurísticos se obtienen por relajación de alguna de las restricciones del problema.

Utilizaremos una notación similar a la empleada en el capítulo anterior para la descripción de las estrategias heurísticas. Así, el problema residual representado por un estado n viene dado por las tareas no planificadas en n junto con sus cabezas y colas, es decir

$$P(n) = (US(n),\ \mathbf{r}_n(US(n)), \mathbf{p}_n(US(n)), \mathbf{q}_n(US(n))). \tag{5.2}$$

En general, en un estado n hay un número de trabajos con todas sus tareas planificadas y otros con tareas aún sin planificar, estos subconjuntos de J se denotan respectivamente como:

$$\begin{gathered} J_{US}(n) = \{J_i \in J; \exists j, 1 \leq j \leq M, \theta_{ij} \in US(n)\} \\ J_{SC}(n) = J \backslash J_{US}(n). \end{gathered} \tag{5.3}$$

El valor de $g(n)$ viene dado por la suma de los tiempos de completud de las últimas tareas planificadas de cada trabajo. De acuerdo con la notación anterior, este valor se expresa como:

$$g(n) = \sum_{J_i \in J_{SC}(n)} (r_{\theta_{iM}} + p_{\theta_{iM}}) + \sum_{J_i \in J_{US}(n)} (r_{\theta_{ik_i}} + p_{\theta_{ik_i}}) \tag{5.4}$$

donde k_i es el índice de la última tarea planificada del trabajo $J_i \in J_{US}(n)$, es decir

$$k_i = (j; \theta_{ij} \in SC(n), \theta_{ij+1} \in US(n)). \tag{5.5}$$

En el estado n, consideramos a su vez el conjunto de trabajos $J_{US}(n)$ como la unión disjunta $J_m \cup J_{\overline{m}}$, donde J_m es el subconjunto de trabajos con alguna operación en $US(n)$ que requiere la máquina m y $J_{\overline{m}}$ es el subconjunto de trabajos con operaciones en $US(n)$ y que no requieren la máquina m, es decir

$$\begin{gathered} J_m = \{J_i \in J; \exists j, 1 \leq j \leq M,\ t.q.\ \theta_{ij} \in US(n)\ \ y\ R_{\theta_{ij}} = m\} \\ J_{\overline{m}} = \{J_i \in J; \forall j, 1 \leq j \leq M, R_{\theta_{ij}} \neq m\ si\ \theta_{ij} \in US(n)\}. \end{gathered} \tag{5.6}$$

Para obtener modelos relajados del problema $J||\sum C_i$, en primer lugar se divide el problema original $P(n)$ en dos subproblemas $P(n)|_{J_m}$ y $P(n)|_{J_{\overline{m}}}$ definidos del siguiente modo. Sean $US(n)|_{J_m}$ y $US(n)|_{J_{\overline{m}}}$ los conjuntos de tareas no planificadas de J_m y $J_{\overline{m}}$ respectivamente, es decir

$$\begin{gathered} US(n)|_{J_m} = \{\theta \in US(n); \theta \in J_i, J_i \in J_m\} \\ US(n)|_{J_{\overline{m}}} = \{\theta \in US(n); \theta \in J_{\overline{m}}, J_i \in J_{\overline{m}}\}. \end{gathered} \tag{5.7}$$

De forma análoga al problema $P(n)$, los subproblemas $P(n)|_{J_m}$ y $P(n)|_{J_{\overline{m}}}$ se definen como

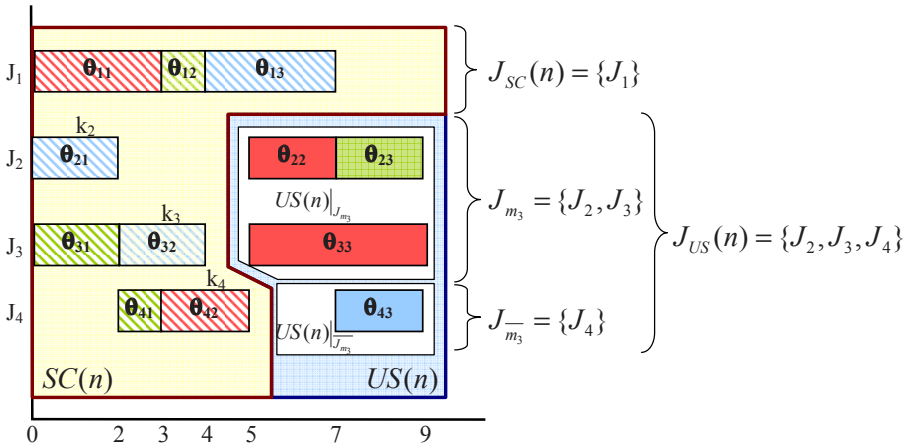
$$P(n)|_{J_m} = (US(n)|_{J_m}, \mathbf{r}_n(US(n)|_{J_m}), \mathbf{p}_n(US(n)|_{J_m}), \mathbf{q}_n(US(n)|_{J_m}))$$
$$P(n)|_{J_{\overline{m}}} = (US(n)|_{J_{\overline{m}}}, \mathbf{r}_n(US(n)|_{J_{\overline{m}}}), \mathbf{p}_n(US(n)|_{J_{\overline{m}}}), \mathbf{q}_n(US(n)|_{J_{\overline{m}}})). \tag{5.8}$$

Una vez realizada esta partición del problema original, se diseñan relajaciones para cada uno de estos subproblemas. La cota inferior de la solución del problema original se calcula como la suma del coste óptimo, o una cota inferior, de ambas relajaciones del problema. Utilizaremos también la notación $P(n)|_m$ para referirnos al problema en el que solamente se consideran las tareas no planificadas que requieren a la máquina m, por supuesto con sus cabezas, duraciones y colas en el estado n. Es decir:

$$P(n)|_m = (US(n)|_m, \mathbf{r}_n(US(n)|_m), \mathbf{p}_n(US(n)|_m), \mathbf{q}_n(US(n)|_m)). \tag{5.9}$$

Donde $US(n)|_m$ denota el conjunto de tareas no planificadas en el estado n que requieren a la máquina m. El problema $P(n)|_m$ se conoce en la literatura como el problema de secuenciamiento de una máquina (OMS) con cabezas y colas.

La figura 5.1 trata de clarificar mediante un ejemplo la notación descrita anteriormente. Para ello, empleamos un estado intermedio de un problema con 4 trabajos y 3 máquinas, en el que hay 8 tareas planificadas ($SC(n) = \{\theta_{11}, \theta_{12}, \theta_{13}, \theta_{21}, \theta_{31}, \theta_{32}, \theta_{41}, \theta_{42}\}$) y 4 sin planificar ($US(n) = \{\theta_{22}, \theta_{23}, \theta_{33}, \theta_{43}\}$). Por tanto, en este estado hay 1 trabajo completamente planificado ($J_{SC}(n) = J_1$) y 3 sin terminar de planificar ($J_{US}(n) = J_2, J_3, J_4$). Si nos fijamos en una máquina con tareas



Índices de las últimas tareas planificadas de los trabajos $J_{US}(n)$: $\mathbf{k_2} = 1$; $\mathbf{k_3} = 2$; $\mathbf{k_4} = 2$;

g(n) = 18;

Para la máquina $\mathbf{m_3}$

$$J_{m_3} = \{J_2, J_3\};\ US(n)\big|_{J_{m_3}} = \{\theta_{22}, \theta_{23}, \theta_{33}\}$$
$$J_{\overline{m_3}} = \{J_4\};\quad US(n)\big|_{J_{\overline{m_3}}} = \{\theta_{43}\};$$

De igual modo para la máquina $\mathbf{m_1}$

$$J_{m_1} = \{J_2\};\ US(n)\big|_{J_{m_1}} = \{\theta_{22}, \theta_{23}\}$$
$$J_{\overline{m_1}} = \{J_3, J_4\};\ US(n)\big|_{J_{\overline{m_1}}} = \{\theta_{33}, \theta_{43}\};$$

Figura 5.1: Esquema de Notación aplicada a un problema con 4 trabajos y 3 máquinas

sin planificar, por ejemplo la máquina m_3, podemos distinguir entre los trabajos con tareas no planificadas, aquellos que tienen tareas que emplean la máquina m_3 ($J_{m_3} = \{J_2, J_3\}$) junto con sus tareas ($US(n)|_{J_{m_3}} = \{\theta_{22}, \theta_{23}, \theta_{33}\}$), de los que no las emplean ($J_{\overline{m_3}} = \{J_4\}$) junto con sus tareas ($US(n)|_{J_{\overline{m_3}}} = \{\theta_{43}\}$).

Para ilustrar la forma en que se aplican los diferentes heurísticos, utilizaremos el estado intermedio del problema $FT06$ que se representa en la Figura 5.2 mediante un gráfico de Gantt. Como se puede observar en este estado hay 8 tareas planificadas, es decir, con un tiempo de inicio ya asignado, concretamente $SC(n) = \{\theta_{11}, \theta_{12}, \theta_{31}, \theta_{32}, \theta_{41}, \theta_{42}, \theta_{61}, \theta_{62}\}$. Mientras que para el resto de tareas ($US(n)$) solamente se conoce una cota inferior de su tiempo de inicio, que viene dada por su cabeza en el estado n. Por tanto, el problema $P(n)$ vendrá definido por las tareas pertenecientes a $US(n)$ junto con sus cabezas, duraciones y colas. En este ejemplo $J_{US}(n) = \{J_1, J_2, J_3, J_4, J_5, J_6\}$, las últimas tareas planificadas de los trabajos $J_{US}(n)$ son $\{\theta_{12}, \theta_{32}, \theta_{42}, \theta_{62}\}$, $J_{SC}(n) = \{\oslash\}$, y $g(n) = 37$. Si consideramos, por ejemplo, la máquina m_1, para ella $J_{m_1} = \{J_2, J_3, J_5, J_6\}$ y $J_{\overline{m_1}} = \{J_1, J_3\}$, siendo $US(n)|_{J_{m_1}}$ y $US(n)|_{J_{\overline{m_1}}}$ respectivamente, las tareas sin planificar pertenecientes a estos conjuntos. De forma análoga, para la máquina m_1 el problema original $P(n)$ se divide en los subproblemas $P(n)|_{J_{m_1}}$ y $P(n)|_{J_{\overline{m_1}}}$ que estarán constituidos respectivamente por las tareas $US(n)|_{J_{m_1}}$ y $US(n)|_{J_{\overline{m_1}}}$, junto con sus cabezas, duraciones y colas. El problema $P(n)|_m$, problema OMS, para esta máquina m_1, estará constituido por las tareas no planificadas que emplean la máquina m_1, $US(n)|_{m_1} = \{\theta_{25}, \theta_{34}, \theta_{55}, \theta_{64}\}$, junto con sus cabezas, duraciones y colas.

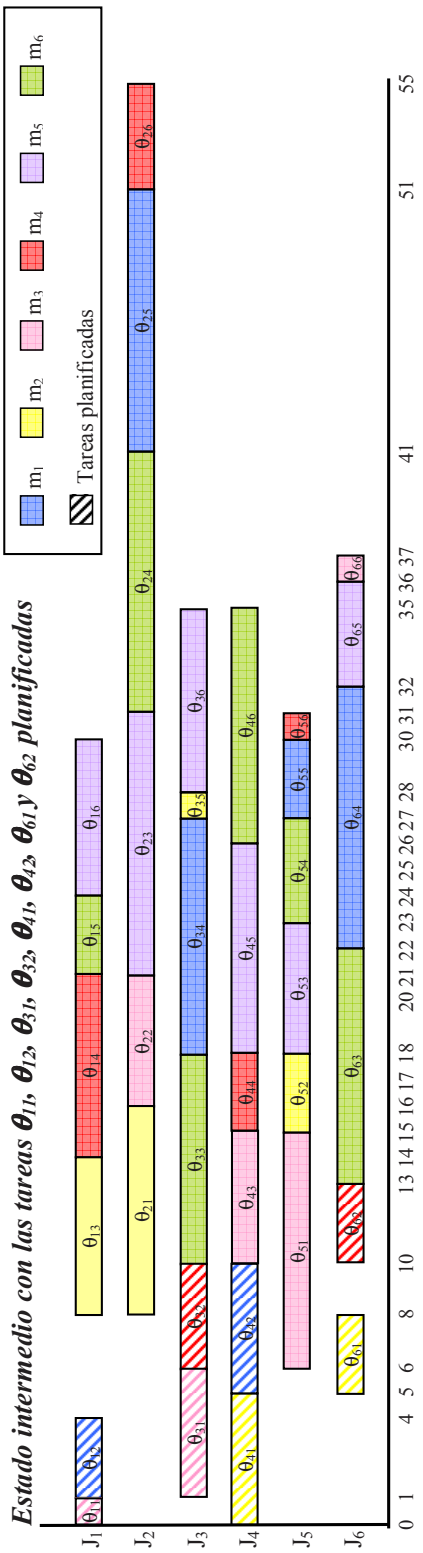


Figura 5.2: Gráfica de Gantt para un estado intermedio del problema $FT06$

5.3.1. Heurístico h_1

Esta estimación heurística se obtiene a partir de una sencilla simplificación del problema, resoluble en un tiempo polinomial, que consiste en relajar todas las restricciones de capacidad que afectan a cada par de tareas de $US(n)$. De acuerdo con esta relajación, las tareas no planificadas pueden solapar sus intervalos de procesamiento aunque requieran la misma máquina. En consecuencia la planificación óptima se consigue planificando cada tarea en el instante indicado por su cabeza. Así, la estimación del coste del problema $P(n)$ se calcula simplemente sumando el tiempo de completud más temprano para todos los trabajos de $J_{US(n)}$, donde el tiempo de completud más temprano de un trabajo es la suma de la cabeza de su última tarea y su tiempo de procesamiento. De este modo obtenemos una cota inferior de $f^*(n)$. Por lo que, para obtener una cota inferior de $h^*(n)$ es necesario restar el valor de $g(n)$. Esta estimación heurística se calcula del siguiente modo:

$$\begin{aligned} h_1(n) &= F_1(P(n)) + \textstyle\sum_{J_i \in J_{SC}(n)} (r_{\theta_{iM}} + p_{\theta_{iM}}) - g(n); \\ F_1(P(n)) &= \textstyle\sum_{J_i \in J_m \cup J_{\overline{m}}} (r_{\theta_{iM}} + p_{\theta_{iM}}) \end{aligned} \tag{5.10}$$

La gráfica de la Figura 5.3 muestra la solución óptima del problema relajado correspondiente al estado intermedio del problema $FT06$. En ella se puede observar que al realizar la relajación de las restricciones de capacidad, el cálculo del heurístico queda reducido a la suma de los tiempos de fin de las últimas tareas de cada trabajo, descontando el valor de la g para obtener la estimación heurística h_1.

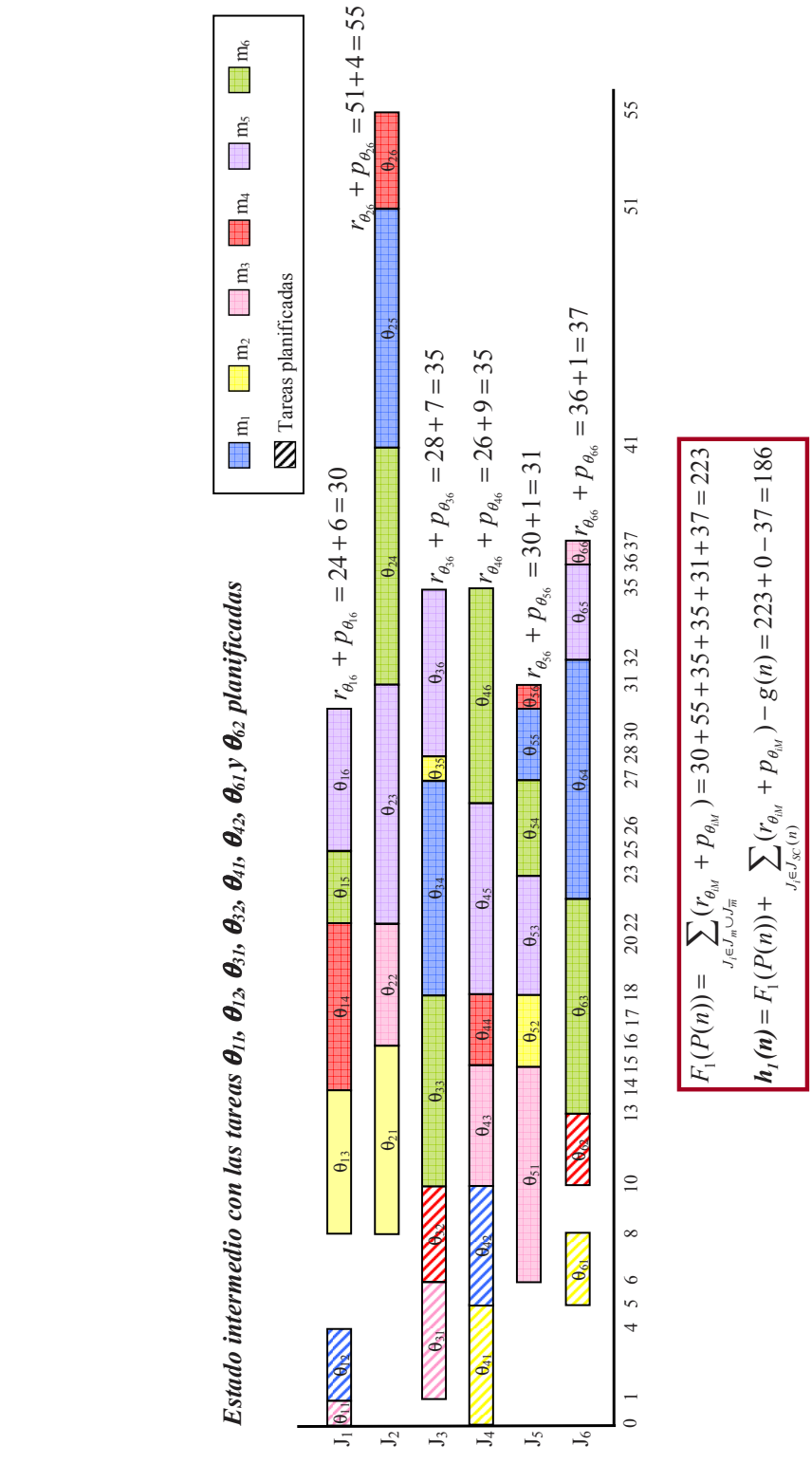


Figura 5.3: Valor del heurístico h_1 para un estado intermedio del problema $FT06$

5.3.2. Heurístico h_2

Este heurístico se inspira en el heurístico diseñado para la versión del problema con minimización del makespan propuesto en [16, 17] y explicado de forma más detallada en la sección 4.5.3. Para calcular una cota inferior del tiempo de flujo total de los trabajos en el subproblema $P(n)|_{J_m}$, comenzamos relajando las restricciones de capacidad que afectan a las tareas en $US(n)$ que no requieren la máquina m. En otras palabras, consideramos el problema simplificado $P(n)|_m$. La estimación para el problema $P(n)|_{J_{\overline{m}}}$ se hace de forma similar al heurístico h_1. El problema $P(n)|_m$ con minimización del flujo total continúa siendo *NP-duro*, por lo que se realiza una nueva simplificación: se relaja la restricción de no expulsión en la máquina m. En [10], Blazewicz y sus colaboradores describen el cálculo de una solución óptima para este problema que se obtiene aplicando la regla "menor tiempo de proceso remanente"(SRPT). Esta cota se basa en el resultado que se muestra a continuación.

Proposición 5.3.1. *Sean u y v dos tareas de $US(n)|_m$ tales que $r_u \leq r_v$ y $p_u \leq p_v$, entonces existe una planificación óptima en la que tarea v comienza tras el fin de la tarea u.*

Así, una solución óptima al problema resultante de esta segunda relajación, se obtiene realizando una simple modificación a la planificación de Jackson (Jackson's Preemptive Schedule ó JPS) [16, 17] para la minimización del makespan. Esta modificación afecta solamente a las prioridades empleadas para la construcción de la misma. En lugar de considerar la prioridad de las tareas en relación directa con los valores de sus colas, las prioridades se asignan ahora en relación inversa con sus tiempos de procesamiento remanente (a menor tiempo remanente mayor prioridad). La planificación resultante se calcula siguiendo el siguiente algoritmo. Comenzando en el instante de tiempo t dado por la menor cabeza de entre todas las tareas, consideramos las tareas pertenecientes al conjunto $D = \{u/r_u \leq t \wedge p'_u > 0\}$, formado por aquellas tareas aún no terminadas en el instante t (p'_u denota el tiempo de procesamiento remanente de una tarea u en el instante t). Se elige v, la tarea con menor tiempo de procesamiento remanente. Según la proposición 5.3.1 es óptimo planificar esta tarea desde el instante t hasta el instante t' dado por el mínimo entre $t + p'_v$ y el instante en el que una nueva tarea (es decir una tarea no incluida en D) está disponible. El instante t se incrementa hasta t' y el proceso continúa hasta que todas las operaciones se hayan completado. Como se ha visto en capítulos anteriores, Carlier y Pinson probaron en [16, 17] que el cálculo de la planificación JPS tiene una complejidad de $O(k \times \log_2(k))$, siendo k el número de tareas.

La planificación JPS modificada del problema $P(n)|_m$ proporciona, sumando los tiempos de fin y las colas de todas las tareas en $US(n)|_m$, una cota inferior del mejor tiempo de flujo total que podemos alcanzar desde el estado n para los trabajos J_m. Una vez calculado esta cota, se calcula el valor máximo entre esta cota y la suma de los tiempos de fin de los trabajos J_m. Para obtener una cota inferior de $f^*(n)$ es necesario sumar a este valor los tiempos de fin del resto de los trabajos. Quedándonos con el mayor de esos valores de entre todas las máquinas obtenemos una cota inferior de $f^*(n)$. Por lo tanto, para obtener una cota inferior de $h^*(n)$, se debe descontar el valor de $g(n)$.

Así pues calculamos el heurístico h_2 de la siguiente manera

$$h_2(n) = F_2(P(n)) + \sum_{J_i \in J_{SC}(n)}(r_{\theta_{iM}} + p_{\theta_{iM}}) - g(n);$$

$$F_2(P(n)) = \text{máx}_{m \in R}\{\text{máx}(\sum_{J_i \in J_m}(C_{\theta_i{}^m} + q_{\theta_i{}^m}), \sum_{J_i \in J_m}(r_{\theta_{iM}} + p_{\theta_{iM}})) \ + \sum_{J_i \in J_{\overline{m}}}(r_{\theta_{iM}} + p_{\theta_{iM}})\}. \tag{5.11}$$

donde ${\theta_i}^m$ es la tarea del trabajo J_i que requiere la máquina m y $C_{{\theta_i}^m}$ su tiempo de completud en la planificación de Jackson (JPS) calculada según el algoritmo propuesto anteriormente. El heurístico h_2 es consistente ya que se obtiene como el coste de la solución óptima de una versión relajada del problema original [62].

La gráfica de la Figura 5.4 muestra la solución óptima del problema relajado correspondiente al estado de la Figura 5.2. En primer lugar se muestra el cálculo de la solución óptima para el problema relajado correspondiente a la máquina m_5. Se calcula la solución del problema relajado construyendo la planificación JPS para esa máquina. Una vez que se tiene esto, se calcula una cota inferior (LB_{m_5}) de la solución del problema residual $P(n)$ sumando las estimaciones calculadas para los conjuntos de trabajos J_{m_5} y $J_{\overline{m_5}}$. Lo mismo se haría para el resto de las máquinas con tareas sin planificar. Sumando al máximo de todos estos valores el tiempo de fin de los trabajos en J_{SC}, y descontando el valor de la función g, tenemos el valor que proporciona el heurístico h_2.

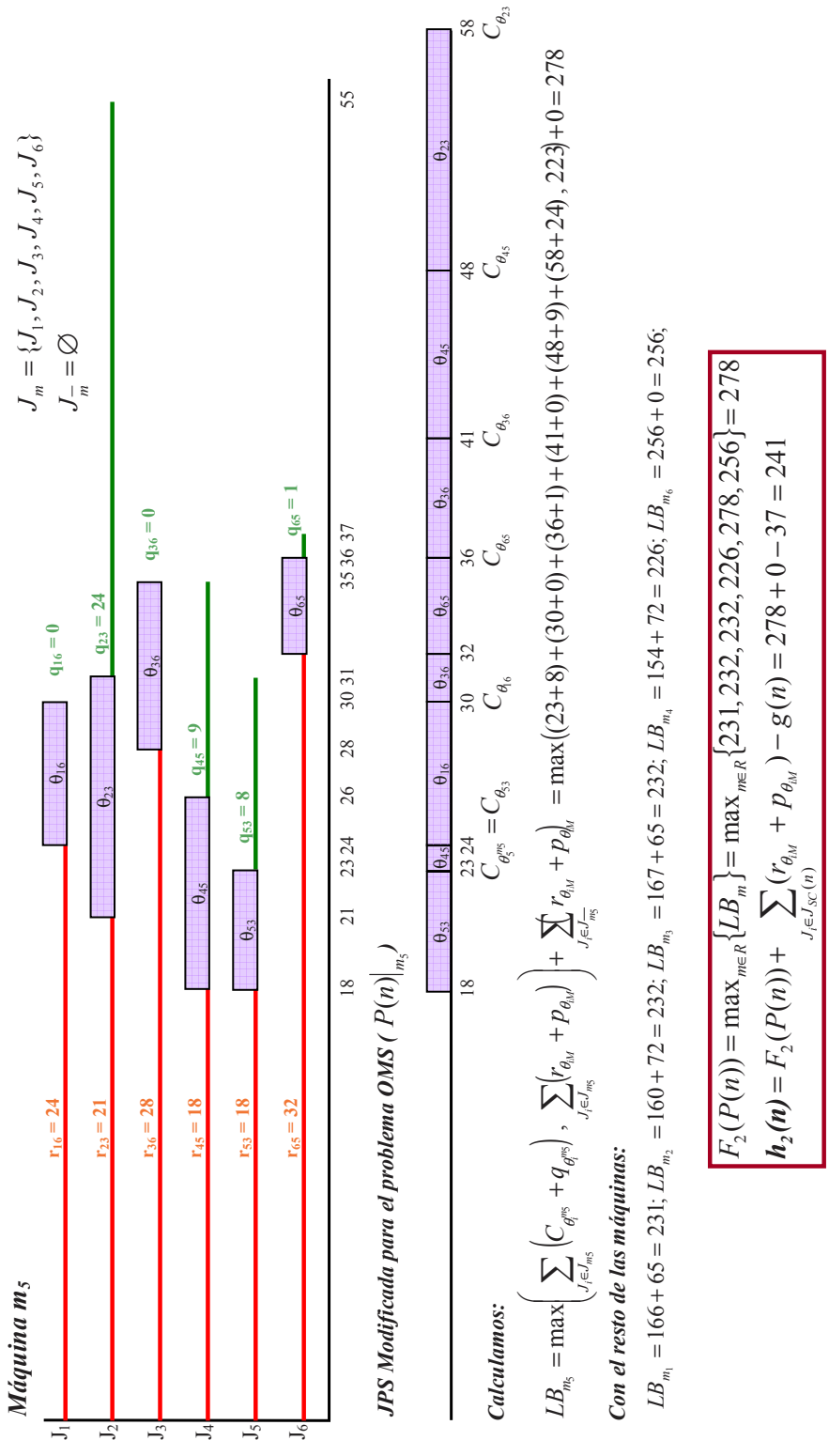


Figura 5.4: Valor del heurístico h_2 para un estado intermedio del problema $FT06$

5.3.3. Heurístico h_3

Para diseñar este heurístico se comienza, como en h_2, relajando en el problema $P(n)|_{J_m}$ las restricciones de capacidad que afectan a cada par de tareas no planificadas en el estado n, $US(n)$; pero se mantienen las correspondientes a las tareas no planificadas que requieren a una máquina m. La diferencia con el heurístico h_2 es que se mantienen, en el modelo relajado, las restricciones de capacidad que afectan a una tarea de $SC(n)$ y a otra tarea de $US(n)$. De este modo el problema relajado es un problema diferente al que denotamos por $P'(n)|_m$. Se trata ahora del problema OMS con cabezas, due dates (tiempos máximos de fin sin incurrir en penalización) y minimización del tardiness (tiempo de retardo de todos los trabajos). El tardiness de una tarea u es la penalización por terminar en un tiempo posterior a su due date y se define como $T_u = \max(0, C_u - d_u)$, donde C_u es el tiempo en el que la tarea u es completada y d_u su due date. Este problema lo denotaremos como

$$P'(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m),\ \mathbf{p}_n(US(n)|_m),\ \mathbf{d}_n(US(n)|_m)), \tag{5.12}$$

donde $\mathbf{d}_n(US(n)|_m)$ denota los due dates de las operaciones, calculadas como: $d_{\theta ij} = r_{\theta iM} + p_{\theta iM} - q_{\theta ij}$ para cada tarea θ_{ij} en $US(n)|_m$.

Para el problema $P(n)|_{J_{\overline{m}}}$ se calcula la misma estimación que en los heurísticos h_1 y h_2.

El problema $P'(n)|_m$ con minimización del tardiness o tiempo total de retardo es *NP-duro* [70]. Incluso relajando la restricción de no expulsión, el problema resultante continúa siendo *NP-duro*. En [5], P. Baptiste y sus colaboradores introducen una nueva cota inferior para el problema con expulsión. El cálculo de esta cota se basa en dos resultados que se muestran a continuación.

Proposición 5.3.2. *Sean u y v dos tareas tales que $r_u \leq r_v$, $p_u \leq p_v$ y $d_u \leq d_v$, entonces existe una planificación óptima en la que la tarea v comienza tras el fin de la tarea u.*

Proposición 5.3.3. *Sean u y v dos tareas tales que $r_u \leq r_v$, $p_u \leq p_v$ y $d_u > d_v$. Intercambiar d_u y d_v no incrementa el tardiness total óptimo.*

Estos resultados permiten calcular una cota inferior aplicando el siguiente algoritmo. Comenzando en el instante de tiempo t dado por la menor cabeza de entre todas las tareas, consideramos las tareas pertenecientes al conjunto $D = \{u / r_u \leq t \wedge p'_u > 0\}$, formado por aquellas tareas aún no terminadas en el instante t (p'_u denota el tiempo de procesamiento remanente de una tarea u en el instante t). Se eligen u, la tarea con menor tiempo de procesamiento remanente, y v la tarea con el menor due date. Si se cumple que $d_u = d_v$, de acuerdo a la proposición 5.3.2 es óptimo planificar esta tarea desde el instante t hasta el instante t', instante dado por el mínimo entre $t + p'_u$ y el instante en el que una nueva tarea (es decir una tarea no incluida en D) está disponible. Si no se cumple $d_u = d_v$, entonces según la proposición 5.3.3, los due dates de las tareas u y v son intercambiados. De este modo la tarea u, tendrá el menor tiempo de procesamiento remanente y el menor due date, teniendo la nueva instancia del problema un tardiness óptimo menor o igual que el original. Por lo tanto, es óptimo planificar u desde el instante t hasta el instante t'. El instante t se incrementa hasta t' y el proceso continúa hasta que todas las operaciones sean completadas. Este algoritmo tiene una complejidad de $O(k \log k)$, siendo k el número de tareas.

Así pues, calculamos el heurístico denominado h_3, del siguiente modo

$$\begin{aligned} h_3(n) &= F_3(P(n)) + \textstyle\sum_{J_i \in J_{SC}(n)} (r_{\theta_{iM}} + p_{\theta_{iM}}) - g(n); \\ F_3(P(n)) &= \text{máx}_{m \in R} \{ \textstyle\sum_{J_i \in J_m} (T_{\theta_i{}^m} + r_{\theta_{iM}} + p_{\theta_{iM}}) \ + \sum_{J_i \in J_{\overline{m}}} (r_{\theta_{iM}} + p_{\theta_{iM}}) \}. \end{aligned} \tag{5.13}$$

donde $\theta_i{}^m$ es la tarea del trabajo J_i que requiere la máquina m y $T_{\theta_i{}^m}$ su tardiness en la planificación con expulsión calculada por el algoritmo anterior para el problema $P'(n)|_m$. El heurístico h_3 no se obtiene de una solución óptima del problema relajado sino de una cota inferior, por lo tanto es admisible pero puede no ser consistente.

La gráfica de la Figura 5.5 muestra una cota inferior de la solución óptima del problema relajado correspondiente al estado intermedio del problema $FT06$, que se muestra a su vez en la Figura 5.2. En la Figura 5.5 se muestra en primer lugar el cálculo de una cota inferior de la solución óptima para el problema OMS correspondiente a la máquina m_5. Como se ve este problema está definido por las cabezas, duraciones y due dates de las tareas que requieren la máquina 5 y están sin planificar (los due dates de cada tarea se muestran con una línea vertical de color azul). En primer lugar se calcula una cota inferior de la solución del problema relajado construyendo la planificación JPS para esa máquina (en ella se muestran los instantes en los que se han intercambiado los due dates de las tareas para lograr que la tarea planificada tenga menor tiempo remanente y menor due date). Una vez que se tiene la planificación, se calculan los tardiness de las tareas ($T_{\theta_i{}^m}$). Con ellos se calcula una cota inferior (LB_{m_5}) de la solución del problema residual $P(n)$ sumando las estimaciones calculadas para los conjuntos de trabajos J_{m_5} y $J_{\overline{m_5}}$. Los mismo se haría para el resto de las máquinas con tareas sin planificar. El máximo de todos estos valores será una cota inferior de la solución del problema $P(n)$. Si a este valor le sumamos el tiempo de fin de los trabajos en J_{SC}, y descontando el valor de la función g, tenemos el valor que proporciona el heurístico h_3.

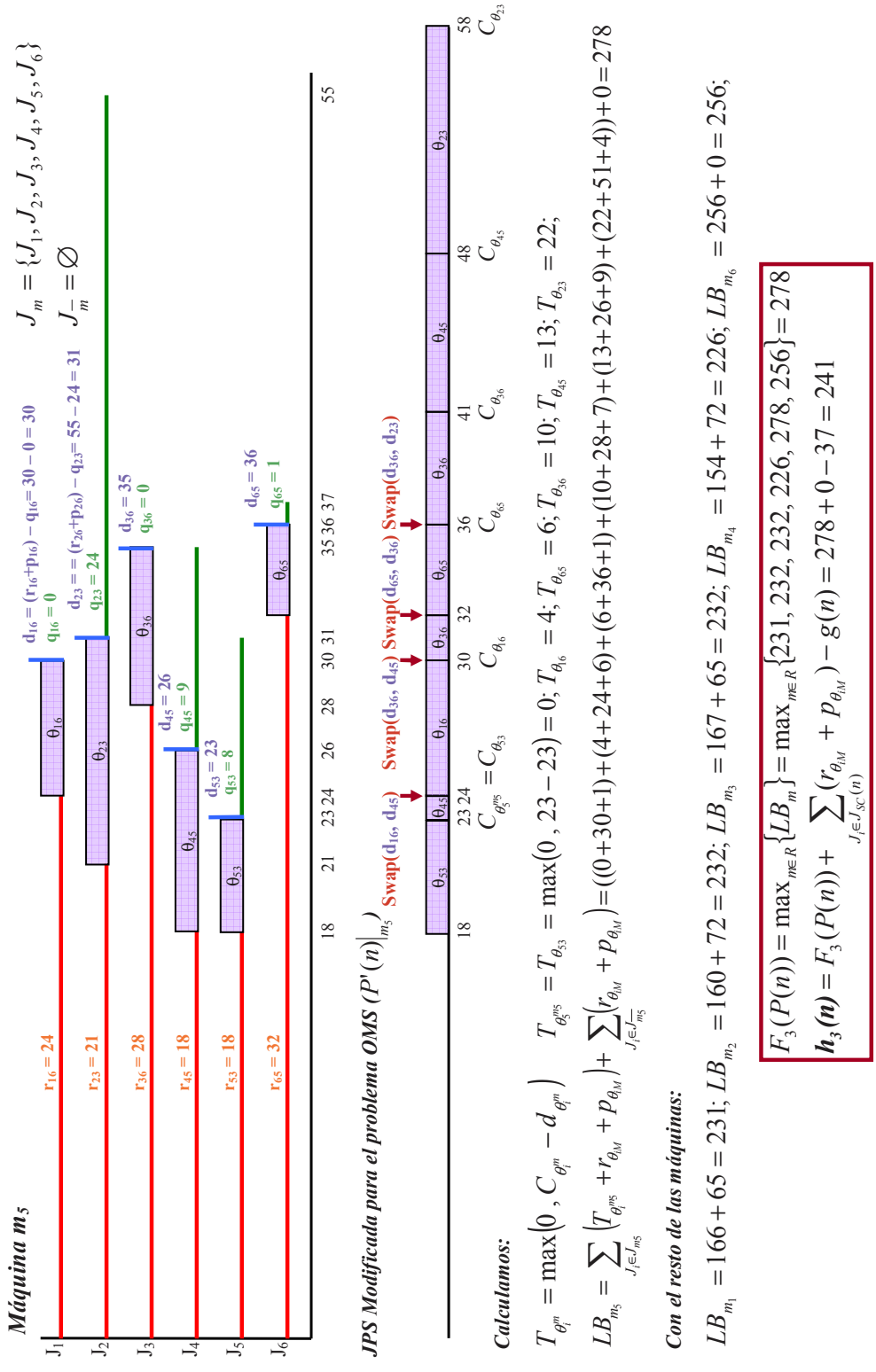


Figura 5.5: Valor del heurístico h_3 para un estado intermedio del problema $FT06$

5.3.4. Heurístico h_4

Las estimaciones que hace el heurístico h_3 descrito en la sección anterior se puede mejorar si se emplea el conjunto de reglas de dominancia propuestas por Baptiste y sus colaboradores en [5], reglas que generalizan a las reglas conocidas como reglas de Emmons y propuestas en [27]. Las reglas de Emmons originales se aplican al problema OMS (con minimización del tardiness) y permiten deducir algunas relaciones de precedencia en determinados casos en los que las cabezas de todas las operaciones son iguales y la expulsión no está permitida. La generalización de estas reglas cuando las cabezas son arbitrarias sólo es posible si la restricción de no expulsión se relaja. En este caso se denominan reglas de Emmons generalizadas [5] y pueden emplearse para ajustar la cota inferior en el problema con expulsión. La aplicación de las reglas de Emmons generalizadas requiere un deadline (tiempo límite de fin) δ_u para cada tarea u. El valor inicial de δ_u puede establecerse como el tiempo de completud de una planificación activa. Es fácil ver que todas las planificaciones activas tienen el mismo tiempo de completud C_{max}.

Para obtener el valor de C_{max}, se puede construir una planificación en la que los trabajos se planifican en orden no decreciente de sus tiempos de inicio. Una vez hecho esto el valor de δ_u se puede ajustar teniendo en cuenta que cada operación u no puede terminar después de un tiempo $C_{max} - \Sigma_{v \in A_u} p_v$, donde A_u es un conjunto de operaciones que se ha determinado que tienen que empezar después de que se complete la tarea u. De igual manera, el tiempo de inicio r'_u de una tarea u puede ser inicializado como r_i y ser posteriormente ajustado a $\max(r'_u, C_{max}(B_u))$, dónde B_u es el conjunto de tareas que tienen que terminar antes de que la tarea u pueda empezar y $C_{max}(B_u)$ es el tiempo de fin de una planificación activa de las tareas de B_u. Las reglas de Emmons generalizadas se enuncian en las tres proposiciones siguientes[5]:

Proposición 5.3.4. *(Generalización de la Regla 1 de Emmons) Sea S una planificación y u y v dos tareas tales que $r_u \leq r_v$, $d_u \leq d_v$ y $\delta_u \leq \max(r_v + p_v, d_v)$, entonces existe una planificación S' en la que la tarea v comienza tras el fin de la tarea u y el tardiness de S' es menor o igual que el tardiness de S.*

Proposición 5.3.5. *(Generalización de la Regla 2 de Emmons) Sea S una planificación y u y v dos tareas tales que $r_u \leq r_v$, $p_u \leq p_v$ y $d_u \leq d_v + p_v$, entonces existe una planificación S' en la que la tarea v comienza tras el fin de la tarea u y el tardiness de S' es menor o igual que el tardiness de S.*

Proposición 5.3.6. *(Generalización de la Regla de 3 de Emmons) Sea S una planificación y u y v dos tareas tales que $r_u \leq r_v$, $\delta_u \leq p_v$, entonces existe una planificación S' en la que la tarea v comienza tras el fin de la tarea u y el tardiness de S' es menor o igual que el tardiness de S.*

Estas reglas se aplican en cada paso del algoritmo descrito en la Sección 5.3.3 para poder descartar algunas de las tareas que pueden ser planificadas a continuación (en el mismo instante de tiempo) en la construcción de la planificación, es decir, antes de aplicar la regla $SRPT$. Por ejemplo, si en un instante de tiempo t hay dos tareas u y v que pueden ser planificadas, es decir dos tareas que tienen el mismo tiempo de inicio $r'_u = r'_v = t$, tales que $p'_u \leq p'_v$ y $d_u \leq d_v + p'_v$, entonces como consecuencia de la generalización de la regla de Emmons número 1, el tiempo remanente de

la tarea u puede ser planificado antes del inicio de la tarea v, y los valores r_v, B_v, δ_v y A_u se deben ajustar en consecuencia. Un razonamiento análogo se puede aplicar cuando se siguen las reglas de Emmons generalizadas 2 y 3. Como se indica en [5] esta mejora de la cota inferior se calcula con complejidad $O(k^4)$. Aunque, en la práctica, la aplicación de las reglas de Emmons generalizadas se realiza en una cantidad de tiempo razonable.

La gráfica de la Figura 5.6 muestra una cota inferior de la solución óptima del problema relajado correspondiente al estado de la Figura 5.2. En esta figura se muestra en primer lugar el cálculo de una cota inferior de la solución óptima para el problema $P'(n)|_{m_5}$. Como se puede observar este problema está definido por las cabezas, las duraciones y los due dates de las tareas que requieren la máquina 5 y están aún sin planificar (los due dates de cada tarea se muestran con una línea vertical de color azul). En primer lugar se calcula una cota inferior de la solución del problema relajado. Para ello se construye la planificación JPS para esa máquina (en ella se muestran tanto los instantes en los que se han intercambiado los due dates como los instantes en los que las reglas de Emmons han determinado que una tarea ha de ir antes que otra). Una vez que se tiene esta planificación, se calculan los tardiness de las tareas ($T'_{\theta_i m}$). Con ellos se calcula una cota inferior (LB_{m_5}) de la solución del problema residual $P(n)$ sumando las estimaciones calculadas para los conjuntos de trabajos J_{m_5} y $J_{\overline{m_5}}$. Esto mismo se hace para el resto de máquinas con tareas sin planificar. Sumando al máximo de todos estos valores el tiempo de fin de los trabajos en J_{SC} tenemos una cota inferior de la solución del problema residual. Finalmente se resta el valor de la g y así tenemos la estimación heurística h_4.

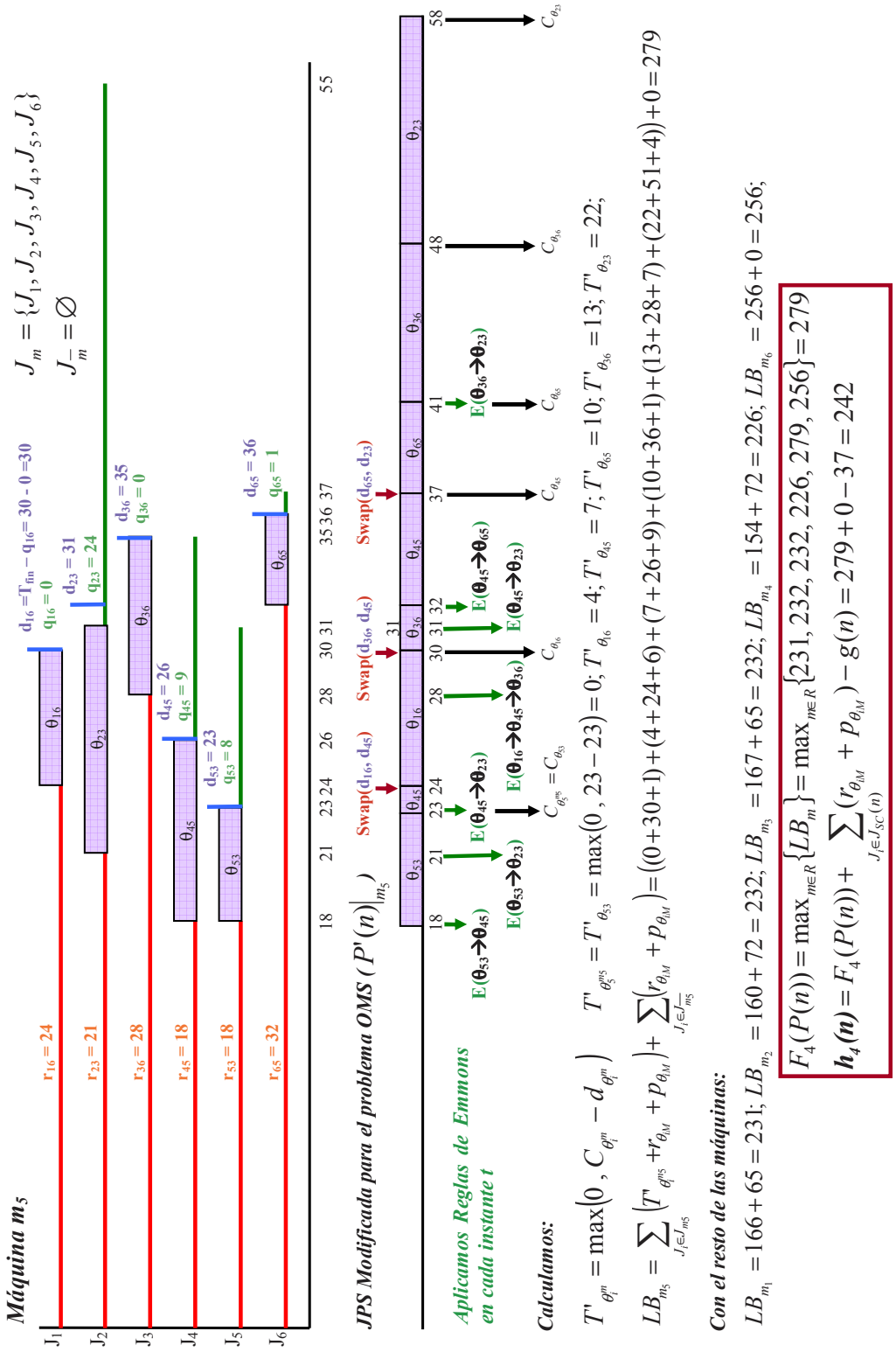


Figura 5.6: Valor del heurístico h_4 para un estado intermedio del problema $FT06$

5.3.5. Mejora de los Heurísticos

Todos estos heurísticos anteriores se basan en el cálculo de cotas inferiores independientes para cada una de las partes o subproblemas en que se divide el problema asociado a cada estado n, es decir $P(n)|_{J_m}$ y $P(n)|_{J\overline{m}}$ para cada máquina m con tareas sin planificar en n. La cota inferior del subproblema $P(n)|_{J_m}$ se obtiene mediante una relajación no demasiado fuerte que requiere calcular una planificación tipo JPS y aplicar una serie de reglas de decisión. Sin embargo, la cota inferior del problema $P(n)|_{J\overline{m}}$ se obtiene realizando una relajación mucho más fuerte que requiere un cálculo más simple, la suma de las cabezas y los tiempos de procesamiento de las últimas operaciones de cada trabajo, es decir es cómo si para este subproblema se calculase el heurístico h_1.

La posibilidad que planteamos ahora es obtener una mejor cota inferior para el problema $P(n)|_{J\overline{m}}$. Para ello, podemos considerar al problema $P(n)|_{J\overline{m}}$ como una instancia con la misma estructura que el problema $P(n)$ original, pero con un tamaño menor. Así, para obtener una cota inferior de este problema podríamos aplicar recursivamente el mismo algoritmo.

En la descripción de los cuatro heurísticos anteriores, las funciones $F_i(P(n))$, $1 \leq i \leq 4$, calculan una cota inferior de la suma de los tiempos de finalización de los trabajos que tienen aun tareas sin planificar en el estado n. Definimos ahora la siguiente familia de funciones, para referirnos a las estimaciones que hacen estos heurísticos para el subproblema $P(n)|_{J_m}$.

$$\begin{aligned} H_1(P(n)|_{J_m}) &= \textstyle\sum_{J_i \in J_m}(r_{\theta_{iM}} + p_{\theta_{iM}}) \\ H_2(P(n)|_{J_m}) &= \textstyle\sum_{J_i \in J_m}(C_{\theta_i^m} + q_{\theta_i^m}) \\ H_3(P(n)|_{J_m}) &= \textstyle\sum_{J_i \in J_m}(T_{\theta_i^m} + r_{\theta_{iM}} + p_{\theta_{iM}}) \\ H_4(P(n)|_{J_m}) &= \textstyle\sum_{J_i \in J_m}(T'_{\theta_i^m} + r_{\theta_{iM}} + p_{\theta_{iM}}), \end{aligned} \tag{5.14}$$

donde $T'_{\theta_i^m}$ denota el tardiness de la tarea θ_i^m cuando se calcula aplicando al algoritmo las reglas de Emmons generalizadas (tal y como se hace en el cálculo del heurístico h_4). Las funciones $F_i(P(n))$, $1 \leq i \leq 4$, se pueden reescribir del siguiente modo (teniendo en cuenta que la función H_1 es aplicable también al subproblema $P(n)|_{J\overline{m}}$).

$$F_i(P(n)) = \max_{m \in R}\{H_i(P(n)|_{J_m}) + H_1(P(n)|_{J_{\overline{m}}})\}. \tag{5.15}$$

La idea de la mejora de los heurísticos es utilizar para el problema $P(n)|_{J\overline{m}}$ una estimación mejor que la que proporciona la función H_1. Para ello se pueden definir las funciones F_i de forma recursiva como

$$F_i(P(n)) = \max_{m \in R}\{H_i(P(n)|_{J_m}) + F_i(P(n)|_{J_{\overline{m}}})\}. \tag{5.16}$$

con $F_i(P) = 0$, si P es un problema nulo.

Por supuesto, el cálculo de las nuevas estimaciones será más costoso. Además es posible que solamente tenga un efecto importante en la mejora de los heurísticos para instancias del problema a partir de un cierto tamaño. El motivo es que en las sucesivas llamadas recursivas los subproblemas

serán cada vez más pequeños, de modo que prácticamente todas las estimaciones darán valores iguales o muy parecidos. En cualquier caso, estas cuestiones requerirán un estudio experimental.

5.4. Propiedades de Dominancia

En esta sección formalizamos el método de poda por dominancia, introducido en el capítulo anterior, para su aplicación al problema $J||\sum C_i$. Este método, como ya se ha comentado, busca la reducción del espacio de búsqueda aplicando una serie de reglas que permiten establecer relaciones de dominancia entre estados que no son sucesores del mismo nodo. Recordemos que la idea principal se basa en ver que la mejor solución alcanzable desde un estado no es mejor que la mejor solución alcanzable desde otro.

El mismo ejemplo simple que utilizamos en el capítulo anterior, nos sirve ahora también para ilustrar la idea de la relación de dominancia entre dos estados. Este ejemplo se muestra en la Figura 5.7. Cuando la función objetivo es el tiempo de flujo total, también es claro que la mejor solución alcanzable desde el estado de la Figura 5.7a no puede ser mejor que la mejor solución alcanzable desde el estado de la Figura 5.7b. En consecuencia, se puede prescindir de aquel estado con la seguridad de que el algoritmo de búsqueda sigue siendo completo.

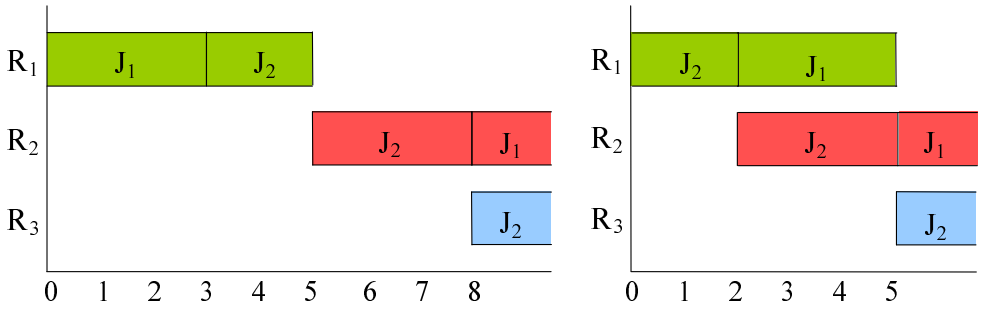


Figura 5.7: Planificación parcial de dos estados de la búsqueda, el estado b) domina al estado a)

Recordemos también cómo hemos definido formalmente la dominancia entre dos estados en el capítulo anterior.

Definición 5.1. *Dados dos estados n_1 y n_2, tales que $n_1 \notin P^*_{s\text{-}n_2}$ y $n_2 \notin P^*_{s\text{-}n_1}$, n_1 domina a n_2 si y sólo si $f^*(n_1) \leq f^*(n_2)$.*

Como hemos visto en el capítulo anterior, las condiciones de dominancia dependen de la naturaleza del problema, en particular de la función objetivo, y también de la función heurística h. A continuación veremos cómo se pueden formular estas condiciones para el problema $J||\sum C_i$ considerando los heurísticos h_1 y h_2. En este caso restringimos desde el principio los resultados para el caso en que los dos nodos n_1 y n_2 tengan las mismas tareas planificadas. Así para simplificar las expresiones, denotaremos por SC y US a los conjuntos de operaciones planificadas y no planificadas en los dos estados respectivamente. Análogamente J_{SC} y J_{US} representan los trabajos con tareas

planificadas y sin planificar en los dos estados. Asimismo, denotaremos por $r_v(n)$ y $q_v(n)$ a la cabeza y a la cola respectivamente de la tarea v en el estado n,

Teorema 5.1. *Sean n_1 y n_2 dos estados tales que $SC(n_1) = SC(n_2)$. Si*

$$\boldsymbol{r}_{n_1}(US) \leq \boldsymbol{r}_{n_2}(US) \tag{5.17}$$

y

$$\sum_{J_i \in J_{SC}} r_{\theta_{iM}}(n_1) \leq \sum_{J_i \in J_{SC}} r_{\theta_{iM}}(n_2) \tag{5.18}$$

entonces se cumplen las siguientes condiciones:

1. $D(n_1) = D(n_2)$.
2. $f(n_1) \leq f(n_2)$.
3. n_1 *domina a* n_2.

Demostración. La condición 1 es evidente, ya que la profundidad de un nodo es el número de tareas planificadas en el nodo, por lo que si ambos estados tienen las mismas tareas planificadas entonces $D(n_1) = D(n_2) = |SC|$.

La condición 2 depende de la estimación heurística y se puede probar fácilmente para los heurísticos h_1 y h_2. Estos dos heurísticos son consistentes, a diferencia de los heurísticos h_3 y h_4. De las expresiones (5.17) y (5.18), se tiene que $\sum_{J_i \in J_{SC} \cup J_{US}} (r_{\theta_{iM}}(n_1) + p_{\theta_{iM}}) \leq \sum_{J_i \in J_{SC} \cup J_{US}} (r_{\theta_{iM}}(n_2) + p_{\theta_{iM}})$. Por lo tanto la condición 2 se cumple para h_1. Para probar esta condición para el heurístico h_2, lo único que tenemos que hacer es observar que si $v \in US$, entonces $q_v(n_1) = q_v(n_2)$ y $r_v(n_1) \leq r_v(n_2)$. Partiendo de esta observación, es fácil ver que una planificación con expulsión PS para el problema relajado $P(n_2)|_m$ es también una planificación con expulsión para el problema $P(n_1)|_m$. Sin embargo, para este problema, la planificación con expulsión PS puede ser no activa ya que algunas tareas podrían comenzar antes sin necesidad de retrasar ninguna otra tarea. Por tanto, moviendo hacia la izquierda las tareas teniendo en cuenta las cabezas con menor valor en el problema $P(n_1)|_m$, la planificación con expulsión PS puede ser transformada en una nueva planificación con expulsión PS' para $P(n_1)|_m$ que es activa y tiene un menor tiempo de flujo total que la planificación PS. Así, teniendo en cuenta este hecho y la expresión (5.18) podemos decir que $f(n_1) \leq f(n_2)$ se cumple para el heurístico h_2.

Por último, la condición 3 se puede probar mediante un razonamiento análogo al utilizado para probar la condición 2 en el heurístico h_2. En este caso se cumple que $f^*(n_1) \leq f^*(n_2)$, por lo tanto n_1 domina a n_2.

□

5.4.1. Reglas de Dominancia

El resultado anterior tiene un aspecto algo diferente a los resultados que hemos establecido en el capítulo anterior para el problema $J||C_{max}$. No obstante, nos permiten definir unas reglas de dominancia que son prácticamente iguales.

Las reglas de dominancia para el problema $J||\sum C_i$ son las siguientes:

1. Cada vez que un nodo n es seleccionado por el algoritmo A^* para ser expandido, n se compara con todo nodo n' en $ABIERTA$ con $f(n) = f(n')$ y $D(n) = D(n')$. Si alguno de los nodos es dominado por el otro, se poda. En el caso de que ambos se dominen mutuamente, se poda el nodo n', es decir el nodo que se encuentra en $ABIERTA$.

2. Si el nodo n no es podado en el paso 1, se compara con aquellos nodos n' que se encuentran en $CERRADA$ y que cumplen que $SC(n') = SC(n)$ ($f(n) \geq f(n')$ siempre que h sea consistente). Si n' domina a n, entonces el nodo n se poda.

La eficiencia de la aplicación de estas reglas se basa también en la implementación de la tablas $ABIERTA$ y $CERRADA$. Debemos destacar aquí, que para los heurísticos h_3 y h_4 la condición 2 del teorema 5.1 no se cumple. Esto hace que al aplicar las reglas de dominancia de igual forma que se hacía en el capítulo anterior (4), no se comprueben posibles situaciones de poda. Concretamente, en la tabla $ABIERTA$ se pierde la comprobación de la poda en las situaciones en las que $f(n) < f(n')$ y n' domina a n. Con respecto a las comprobaciones en la tabla $CERRADA$, son las mismas que para los heurísticos h_1 y h_2, es decir hay que comparar el nodo n con todos los nodos en $CERRADA$ que tienen sus mismas tareas planificadas, con independencia de los valores de f.

5.5. Estudio Experimental

En esta sección vamos a mostrar los diferentes experimentos realizados para el problema $J||\sum C_i$ en condiciones de optimalidad. Comenzaremos analizando el comportamiento de los diferentes heurísticos propuestos. A continuación estudiaremos la técnica de poda descrita en la sección 5.4. Por otro lado, al igual que en el capítulo anterior, trataremos de poner de manifiesto cuál es el tamaño límite de problemas que el algoritmo A^* es capaz de resolver y la influencia de la técnica de poda tanto en el espacio de memoria como en el tiempo de ejecución del algoritmo.

Para realizar los experimentos hemos utilizado los mismos problemas que en el capítulo anterior, recordemos que se trata de problemas seleccionados del repositorio *OR-library* [6], repositorio con instancias de problemas JSS de diferente tamaño y dificultad. Concretamente hemos experimentado con las siguientes instancias: $FT06$ (6×6), $LA01 - 15$ (10×5, 15×5, 20×5, 10×10), $ORB01 - 10$ (10×10), $ABZ05 - 06$ (10×10). En este capítulo, también hemos considerado versiones reducidas de los problemas, incluidos problemas de las baterías YN y *Selectos* (problemas identificados como difíciles [2]): $FT20$ (20×5), $FT10$ (10×10), $ABZ07 - 09$ (20×15, 20×20), $LA21 - 25$ (15×10), $LA27 - 29$ (20×10), $LA38 - 40$ (15×15) y $YN1 - 4$ (20×20). Además, también hemos considerado el conjunto de instancias propuestas por Sadeh y Fox en [74]. Recordemos que es un conjunto de 60 instancias de tamaño 10×5 organizadas en 6 grupos de 10 problemas cada uno caracterizados por los parámetros: BK y RG

La máquina dónde se realiza la experimentación es la misma empleada en la experimentación del capítulo anterior, recordemos que se trata de una máquina con un procesador Intel Core 2 Duo a 2,13 GHz y con 7,6 Gb. de RAM, siendo los experimentos ejecutados sobre Ubuntu $V8{,}04$. El

tiempo límite establecido para la resolución de los problemas también es en este caso de una hora. Cuando los problemas no se resuelven, bien por que se agota la memoria disponible, o bien por que se supera el tiempo límite establecido, se muestran en negrita los valores alcanzados hasta ese momento.

5.5.1. Comparación de Heurísticos

Como se ha visto no todos los heurísticos propuestos se obtienen a partir de soluciones óptimas de problemas relajados, por lo que no todos son consistentes, aunque sí admisibles. En esta sección vamos a estudiar su comportamiento, para ello utilizaremos problemas seleccionados de la *OR-library*. Inicialmente experimentaremos con el problema $FT06$, instancia de tamaño 6×6, del conjunto de problemas FT y con el conjunto de problemas LA del 01 al 15, cuyos tamaños son de 10×5, 15×5 y 20×5 respectivamente.

La Tabla 5.1 muestra los resultados de aplicar los heurísticos diseñados a la instancia del problema $FT06$. Al igual que ocurría en el capítulo anterior, para esta versión del problema sigue tratándose de una instancia muy pequeña que es resuelta incluso con el heurístico trivial $h_0(n) = 0, \forall n$. Como se puede observar, para esta instancia, los heurísticos h_2, h_3 y h_4 expanden todos ellos el mismo número de nodos, siendo este número mucho menor que para los heurísticos h_0 y h_1. Esto pone de manifiesto que estos heurísticos están mejor informados por lo que orientan la búsqueda hacia caminos más prometedores que el heurístico h_1, consiguiendo reducir de forma considerable el espacio que es necesario explorar para alcanzar una solución óptima. Estos resultados muestran que la búsqueda sin el empleo de un heurístico resulta una tarea dura incluso para un problema pequeño como es el $FT06$. La similitud en los resultados obtenidos para los heurísticos h_2, h_3 y h_4, se debe a que el cálculo de estos heurísticos es muy parecido, aunque no son estrictamente comparables ya que aún basándose los tres en la construcción de una planificación JPS, en el caso del h_2 esta planificación permite calcular la solución óptima del problema relajado mientras que en el caso de h_3 y h_4 sólo nos da una cota inferior de la solución del problema relajado, por lo que como se ha visto en su descripción aunque los tres son admisibles, h_3 y h_4 no son consistentes. Además, h_3 y h_4 no son estrictamente comparables con h_1 y h_2.

La Tabla 5.2 muestra los resultados sobre las instancias LA01-15. En primer lugar, cabe destacar que ninguno de los 4 heurísticos es capaz de resolver las instancias $LA06 - 20$ y que además los heurísticos h_0 y h_1 no son capaces de resolver ninguna de las instancias $LA01 - 05$. El resto de heurísticos sí es capaz de encontrar solución para los problemas $LA01$, $LA03$ y $LA04$. Como se

Tabla 5.1: Resultados para el problema $FT06$ resuelto con los diferentes heurísticos

Heurístico	Gen.	Exp.	T.(s)
h_0	7477914	4866614	400
h_1	16931	10614	1
h_2	184	111	0
h_3	184	111	0
h_4	184	111	0

Tabla 5.2: Resultados para los problemas $LA01$-15 resuelto con los diferentes heurísticos. Los valores en negrita indican que la memoria se ha agotado sin alcanzar una solución

	h_2			h_3			h_4		
Inst.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA01$	1118136	475522	129	1109048	471492	138	1105430	470037	227
$LA02$	**3625492**	**1454493**	**411**	**3622825**	**1451824**	**444**	**3703408**	**1485722**	**725**
$LA03$	392860	154791	47	390090	153589	48	389175	153266	79
$LA04$	855500	345005	96	844107	339748	102	843025	339377	166
$LA05$	**3570165**	**1399726**	**406**	**3569995**	**1399554**	**435**	**3570789**	**1400345**	**740**

puede observar en la tabla, a medida que los heurísticos están más informados, el número de estados expandidos para alcanzar la solución es menor, sin embargo, y debido al incremento de la complejidad en el cálculo de los mismos, el tiempo es mayor. Los resultados experimentales nos muestran que a medida que el heurístico está mejor informado la profundidad a la que se llega en la búsqueda, para aquellos problemas no resueltos, es mayor o igual, como cabe esperar. Como se puede observar, en cuanto al número de nodos expandidos, los heurísticos h_3 y h_4 expanden menos nodos que el heurístico h_2, aunque no son estrictamente comparables; el heurístico h_4 mejora un poco a h_3 aunque la aplicación de las reglas de Emmons no garantiza estrictamente una mejora. En cuanto al tiempo empleado para alcanzar una solución, el heurístico h_2 es mejor, aunque muy parecido al h_3, sin embargo, el h_4 debido al coste de tiempo que supone la aplicación de las reglas de Emmons se aleja de los tiempos de h_2 y h_3. Los resultados de la Tabla 5.2 ponen de manifiesto que ante esta batería de problemas el mejor heurístico es el h_2, porque resuelve los mismos problemas que los heurísticos h_3 y h_4, en menor tiempo. Sin embargo, y teniendo en cuenta únicamente estos problemas no es posible realizar una afirmación categórica, pues es posible que ante otra batería de problemas no suceda lo mismo.

5.5.2. Análisis de la Técnica de Poda

En esta sección se describen los resultados obtenidos con las reglas de poda descritas en la sección 5.4.1. Para ello se ha hecho un estudio experimental similar al realizado en el capítulo anterior.

Para analizar la eficiencia de la técnica de poda, hemos experimentado en primer lugar con el problema $FT06$ y con los problemas $LA01-15$, utilizando podas por dominancia tanto en la tabla $ABIERTA$ como en $CERRADA$. La Tabla 5.3 recoge los resultados de los experimentos con el problema $FT06$. Como se puede observar la técnica de poda por dominancia produce una reducción considerable del número de nodos expandidos para alcanzar una solución, concretamente cuando no se emplea heurístico, es decir cuando se usa el heurístico h_0, la reducción es grandísima, siendo también considerable en el resto de heurísticos. Aunque se trata de una instancia muy pequeña, este experimento permite comprobar que el efecto de la poda por dominancia está en relación inversa con el grado de información del heurístico.

La Tabla 5.4, muestra los resultados de aplicar la técnica de poda por dominancia a las instancias $LA01$-15. Los resultados de estos experimentos muestran que aún empleando la técnica de poda por

Tabla 5.3: Resultados para el problema $FT06$ resuelto con los diferentes heurísticos y empleando la técnica de poda

	No poda			Poda		
Heurístico	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
h_0	7477914	4866614	400	69929	46594	7
h_1	16931	10614	1	5584	3541	0
h_2	184	111	0	165	100	0
h_3	184	111	0	165	100	0
h_4	184	111	0	165	100	0

Tabla 5.4: Resultados para los problemas $LA01$-05 resuelto con los diferentes heurísticos. La negrita indica que la memoria se ha agotado sin alcanzar una solución

	h_2			h_3			h_4		
Inst.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
No Poda									
$LA01$	1118136	475522	129	1109048	471492	138	1105430	470037	227
$LA02$	**3625492**	**1454493**	**411**	**3622825**	**1451824**	**444**	**3703408**	**1485722**	**725**
$LA03$	392860	154791	47	390090	153589	48	389175	153266	79
$LA04$	855500	345005	96	844107	339748	102	843025	339377	166
$LA05$	**3570165**	**1399726**	**406**	**3569995**	**1399554**	**435**	**3570789**	**1400345**	**740**
Poda									
$LA01$	248935	107955	35	246576	106896	37	246145	106711	58
$LA02$	520356	217751	80	516490	215975	84	515840	215751	123
$LA03$	79883	32118	10	79239	31832	12	79403	31886	19
$LA04$	136928	57558	19	135262	56746	20	135074	56678	31
$LA05$	412277	174320	68	410076	173443	70	408693	172827	102

dominancia, al igual que ocurría sin el empleo de dicha técnica, ninguno de los heurísticos es capaz de resolver las instancias $LA06-20$ y que además los heurísticos h_0 y h_1 no son capaces de resolver ninguna de las instancias $LA01-05$. Como se puede ver en la Tabla 5.4, la técnica de poda por dominancia permite al algoritmo A^* resolver todos los problemas $LA01-05$ para los heurísticos h_2, h_3 y h_4 mientras que sin ella los problemas $LA02$ y $LA05$ no son resueltos.

Además, como se puede ver, la técnica de poda por dominancia consigue reducir considerablemente tanto el número de nodos expandidos como el tiempo empleado para alcanzar la solución. Esto nos hace ver que la técnica de poda por dominancia también es una técnica efectiva de reducción del espacio de búsqueda para el problema JSS con minimización del flujo total. Por otro lado, y en lo que se refiere a la comparación de los heurísticos se puede ver que su comportamiento relativo es el mismo con y sin poda por dominancia. En la Tabla 5.4 se pone de manifiesto que, en general, a medida que los heurísticos están más informados el número de nodos expandido es ligeramente menor, tanto con poda como sin ella; sin embargo, el tiempo empleado crece debido al incremento en su complejidad del cálculo, sobre todo para el heurísticos h_4 (heurístico h_3 que emplea las reglas de Emmons). Con respecto al número de nodos expandidos cuando se emplea la

poda, existe una excepción con el problema $LA03$, para el heurístico h_4. Aunque es de esperar que el heurístico h_4 esté mejor informado que el heurístico h_3, no se puede probar de forma estricta que $h_4(n) > h_3(n), \forall n$, por lo que este comportamiento es compatible con la definición de los heurísticos.

En vista de los resultados que la Tabla 5.4 nos muestra, no podemos decir que el empleo de las reglas de Emmons con el heurístico h_3 (heuristico h_4) proporcione buenos resultados para esta batería de problemas (el número de nodos expandidos en general es menor, pero el tiempo mayor), esto puede ser debido a que el tamaño de estos problemas no es demasiado grande para que el efecto de las reglas de Emmons sea visible. En cuanto a los heurísticos h_2 y h_3 podemos decir que en cuanto a número de nodos expandidos es mejor el h_3 pero en cuanto a tiempo de ejecución el h_2, sin embargo las diferencias entre ambos son muy pequeñas por lo que, con los resultados para estos problemas, no se puede decir que uno sea claramente mejor que el otro.

Los resultados obtenidos hasta el momento, en comparación con sus análogos en la versión del problema $J||C_{max}$, parecen indicar que esta versión del problema $J||\sum C_i$ es mucho más difícil. Aún tratándose de versiones diferentes del problema, ambas emplean estimaciones heurísticas basadas en relajaciones del problema similares, pero para la versión $J||C_{max}$ estas instancias son fáciles, por lo que todas son resueltas óptimamente incluso sin emplear la técnica de poda. Este indicio será analizado también en el resto de experimentos que mostramos a continuación.

5.5.3. Eficiencia de la Técnica de Poda en condiciones de Optimalidad

Los experimentos de esta sección tienen el mismo objetivo que sus análogos en el capítulo anterior. En primer lugar, permiten determinar el tamaño límite de problemas que somos capaces de resolver en condiciones de optimalidad y en comparación con otros métodos. Además permiten comprobar el efecto que la técnica de poda por dominancia produce tanto en el espacio de búsqueda como en el tiempo de ejecución. Estos experimentos también sirven para comparar de forma más exhaustiva los tres heurísticos mejor informados. Al igual que en el capítulo anterior, en estos experimentos utilizamos problemas de las baterías FT, LA, ORB, ABZ, YN y $Selectos$, y sus versiones reducidas; así como 60 problemas propuestos por Sadeh y Fox en [74].

Todos los problemas se resuelven con los heurísticos h_2, h_3 y h_4, tratando de ver si alguno de ellos presenta un comportamiento mejor que los demás. Los resultados de los experimentos se recogen en una serie de tablas incluidas en el Anexo 5.8, en las que se muestran el número de nodos generados, expandidos y el tiempo de ejecución empleado para resolver cada uno de los problemas. Además, al igual que en el capítulo anterior, hemos incluido datos estadísticos como son la media y la desviación típica.

Como se ha visto en las tablas de resultados de las secciones anteriores 5.3 y 5.4 tanto el problema $FT06$ como los problemas $LA(\text{del } 01-05)$ se resuelven óptimamente, mientras que los problemas LA $(\text{del } 06-20)$ no se resuelven ni siquiera empleando la técnica de poda. Por ello, y con el fin de ver tanto el efecto de la poda como el limite de tamaño que somos capaces de resolver óptimamente, al igual que hicimos en el capítulo anterior, se redujeron algunos de los problemas de las baterías empleadas para convertirlos en problemas más pequeños. El método utilizado para la reducción de los problemas es el mismo que el capítulo anterior: a partir de un problema de tamaño $n \times m$ para

obtener uno de tamaño menor se reduce el número de trabajos y de máquinas borrando los trabajos de mayor número y las máquinas de mayor número.

Al igual que en el capítulo anterior, consideramos instancias desde tamaño 6×6 hasta tamaño 10×10. Todos los problemas se resolvieron con los heurísticos h_2, h_3 y h_4 empleando la técnica de poda y sin ella, los resultados de estos experimentos se recogen de forma exhaustiva en las tablas del Anexo 5.8, secciones 5.8.1, 5.8.2 y 5.8.3.

Para poder ver el efecto de la poda, a medida que el tamaño de los problemas crece, se realizaron una serie de tablas resumen en las que para cada heurístico y tamaño de problema se puede ver en primer lugar el número de instancias resueltas, y en segundo lugar las medias y las desviaciones típicas del número de nodos generados, expandidos y del tiempo de ejecución para aquellos problemas resueltos con y sin la técnica de poda.

Instancias de Tamaño 6×6 a 9×9

La Tabla 5.5, muestra un resumen de los resultados de los experimentos realizados con los problemas desde un tamaño 6×6 hasta a un tamaño 9×9.

Veamos en primer lugar el efecto de la poda a medida que el tamaño del problema crece. Como se observa, estos resultados muestran que, al igual que ocurría con la versión del problema $J||C_{max}$, la eficacia de la técnica de poda es mayor a medida que el tamaño de los problemas crece, es decir su efectividad es mayor cuanto mayor es el espacio de búsqueda. Para los problemas de tamaño 6×6 la reducción en media del número de nodos expandidos no es muy significativa, siendo las medias de los tiempos similares. En los problemas de tamaño 7×7 comienza a apreciarse el efecto de la poda, siendo la diferencia de los tiempos medios aún no demasiado elevada. Es, al igual que ocurría en la versión $J||C_{max}$, en los problemas de tamaño 8×8 en los que se aprecia un salto en cuanto a la reducción del número de nodos expandidos y del tiempo de ejecución, resolviéndose con poda un problema que no se resuelve sin poda (Tablas 5.20, 5.33 y 5.46). En los problemas de tamaño 9×9 el efecto de la poda es mucho mayor y hace que se resuelvan 16 problemas para los que sin poda se agota la memoria sin alcanzar solución (Tablas 5.21, 5.34 y 5.47). Estos resultados nos hacen pensar que el límite de problemas que podemos resolver de forma óptima va a estar próximo al umbral de tamaño 10×10.

En cuanto a los heurísticos, podemos observar que para estos tamaños de problema su comportamiento es similar, ya que los tres resuelven los mismos problemas. En general podemos decir que el heurístico h_4 es el que expande menor número de nodos, pero con una diferencia pequeña con respecto a los heurísticos h_2 y h_3. Sin embargo, es el heurístico h_2 el que requiere menos tiempo en alcanzar una solución, siendo los tiempos entre los heurísticos h_2 y h_3 parecidos, pero disparándose en el heurístico h_4. Esto se debe a que el cálculo del heurístico h_4 es mucho más costoso que el de los otros dos.

Hacemos ahora un pequeño inciso para poner de manifiesto que, de nuevo, la versión del problema $J||\sum C_i$ parece ser mucho más difícil que la $J||C_{max}$. Hasta los problemas de tamaño 8×8 somos capaces de resolver los mismos problemas, aunque expandiendo mucha mayor cantidad de nodos para el problema $J||\sum C_i$. Sin embargo, para el tamaño 9×9, en la versión $J||C_{max}$ sin poda

lográbamos resolver 8 problemas más de los que se resuelven ahora. Esto nos lleva a pensar que para esta versión del problema el límite de problemas a resolver va a estar por debajo del tamaño umbral 10×10.

Tabla 5.5: Resultados resumen para problemas recortados de las baterías *LA*, *ORB*, *ABZ* y *Selectos*, con los heurísticos h_2, h_3 y h_4, realizando podas

No Poda									
	h_2			h_3			h_4		
6×6, *Resueltos* 32/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	1194,56	705,34	0,09	1192,06	703,88	0,09	1192,00	703,84	0,19
Desviación	934,73	532,47	0,29	934,19	532,07	0,29	934,13	532,04	0,39
7×7, *Resueltos* 32/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	26215,78	15025,75	3,06	26163,25	14989,16	3,19	26152,31	14983,47	4,66
Desviación	71527,27	39848,97	8,12	71359,53	39728,62	8,46	71319,81	39708,86	12,93
8×8, *Resueltos* 31/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	158019,00	92138,06	22,68	157786,52	91991,06	24,19	157721,90	91955,48	35,45
Desviación	154376,88	92314,80	22,21	154172,15	92180,82	23,60	154123,80	92151,94	35,20
9×9, *Resueltos* 16/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	1680652,88	971419,50	271,13	1678095,38	969856,69	289,88	1677182,19	969326,00	434,06
Desviación	708006,25	399947,91	112,32	707053,82	399410,95	118,34	706561,67	399086,68	174,67
Poda									
	h_2			h_3			h_4		
6×6, *Resueltos* 32/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	704,06	416,78	0,06	702,75	415,97	0,03	702,75	415,97	0,16
Desviación	402,03	228,27	0,24	401,40	227,86	0,17	401,40	227,86	0,36
7×7, *Resueltos* 32/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	5705,47	3302,69	0,69	5696,28	3296,47	0,88	5695,16	3295,84	1,16
Desviación	6760,53	3722,54	1,04	6746,49	3713,26	1,02	6742,06	3711,09	1,56
8×8, *Resueltos* 32/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	30986,84	18038,58	5,26	30937,52	18007,42	5,61	30926,42	18000,84	7,97
Desviación	20768,93	12513,98	3,55	20733,36	12491,24	3,93	20729,40	12488,26	5,60
9×9, *Resueltos* 32/32									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	223243,06	130202,69	45,94	222934,25	130003,56	48,63	222930,56	130001,00	69,81
Desviación	97005,11	56057,24	21,16	96887,53	55985,82	22,35	96884,90	55984,58	30,59

Instancias de Tamaño 10×10

Veamos ahora que ocurre para los problemas que se encuentran en el tamaño umbral. Como se observa en la Tabla 5.6, para esta versión del problema no somos capaces de resolver ningún problema sin recortar de este tamaño, sólo logramos resolver tres instancias recortadas de la batería *Selectos*. La técnica de poda permite resolver estas tres instancias, no resueltas sin poda, expandiendo muchos menos nodos de los que se llegaron a expandir sin poda, antes de agotar la memoria, y alcanzando una solución en un tiempo menor. Por otro lado, tal y como se ve en las Tablas 5.30, 5.43 y 5.56 del Anexo, para los problemas no resueltos, con poda se llega a una profundidad mayor, siendo también, en la mayoría de los casos, la cota inferior alcanzada mejor.

En cuanto a los heurísticos, tiene un comportamiento similar al de los otros tamaños pero con pequeños matices. El heurístico h_2 es el que expande mayor número de nodos, empleando menor tiempo en alcanzar una solución. El heurístico h_3 expande un número un poco menor de nodos pero emplea más tiempo en alcanzar una solución. Por último el heurístico h_4 expande un mayor número de nodos que h_3 para los primeros problemas y expande menos nodos para el tercero, siendo el tiempo en todos los casos mucho mayor.

Estos resultados siguen poniendo de manifiesto que esta versión del problema $J||\sum C_i$ es más difícil de resolver que la versión $J||C_{max}$. Ante heurísticos diseñados con relajaciones similares, el número de nodos expandidos para alcanzar una solución del problema $J||\sum C_i$, es mucho mayor que para el problema $J||C_{max}$. Además como se ve en los resultados de las Tablas 5.22, 5.35 y 5.48, para esta versión del problema sólo resolvemos tres instancias, mientras que para la misma batería de problemas, en la versión $J||C_{max}$, somos capaces de resolver tres instancias de problemas que no son recortados ($ABZ6$, $ORB07$ y $ORB10$) y que sin poda no es posible resolver, además de todas las recortadas excepto una ($LA40 - 10 \times 10$).(Ver Tabla 4.16).

Tabla 5.6: Resultados resumen para los únicos problemas resueltos de tamaño 10×10, problemas recortados de las *Selectos*, con los heurísticos h_2, h_3 y h_4 y realizando podas

No Poda									
	h_2			h_3			h_4		
Inst.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$ABZ7 - 10 \times 10$	**2639236**	**1538304**	**524**	**2638952**	**1538017**	**566**	**2638953**	**1538023**	**913**
$ABZ9 - 10 \times 10$	**2422071**	**1320912**	**489**	**2421717**	**1320556**	**532**	**2421749**	**1320576**	**881**
$LA38 - 10 \times 10$	**2461384**	**1360366**	**498**	**2461209**	**1360193**	**546**	**2461574**	**1360541**	**900**
Poda									
	h_2			h_3			h_4		
Inst.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$ABZ7 - 10 \times 10$	552246	326967	160	551879	326732	169	551935	326758	242
$ABZ9 - 10 \times 10$	1050968	581996	382	1049546	581117	402	1049924	581325	554
$LA38 - 10 \times 10$	937039	536287	265	934964	534999	279	934613	534787	406

Instancias de Sadeh y Fox

En el siguiente grupo de experimentos consideramos el conjunto de instancias propuestas por Sadeh y Fox. Como se puede ver tanto en las Tablas resumen 5.7 y 5.7, como en las Tablas de la 5.23 a la 5.28, utilizando la técnica de poda se resuelven óptimamente todas las instancias, mientras que sin ella hay 5 instancias que no son resueltas. De esos resultados podemos concluir que como se observa en la Tabla 5.7, al igual que ocurría en la versión $J||C_{max}$ del problema, los problemas con 2 cuellos de botella son más difíciles de resolver que los que solo tienen uno, ya que es en estos grupos en los que sin poda no se resuelve alguna instancia y con poda se resuelven expandiendo un elevado número de nodos. En las Tablas 5.7 y 5.8 también se ve que el parámetro RG no es significativo desde el punto de vista de la dificultad del problema.

La técnica de poda produce para estos problemas el efecto deseado en todos los grupos de problemas, siendo la reducción tanto del espacio de búsqueda como del tiempo de ejecución considerable en todos ellos. En cuanto al los heurísticos h_2, h_3 y h_4, siguen el mismo comportamiento observado

Tabla 5.7: Resultados resumen para los problemas de *Sadeh* con $BK = 1$, agrupados por grupos según su parámetros RG, con los heurísticos h_2, h_3 y h_4, realizando podas

No Poda									
	h_2			h_3			h_4		
$RG = 0,0$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	58957,90	26890,80	6,70	58150,80	26626,20	7,10	57504,20	26323,30	10,70
Desviación	74684,40	34776,77	8,71	74346,85	34629,16	9,30	74719,62	34805,67	14,20
$RG = 0,1$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	73069,30	31343,00	8,50	72872,10	31266,50	9,10	72859,00	31260,60	15,70
Desviación	134215,79	58040,91	15,59	134158,82	58000,63	17,03	134148,53	57996,34	28,98
$RG = 0,2$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	11069,20	4968,90	1,30	11007,10	4942,80	1,20	11007,10	4942,80	2,00
Desviación	20318,53	8807,98	2,41	20250,24	8781,80	2,40	20250,24	8781,80	3,85
Poda									
	h_2			h_3			h_4		
$RG = 0,0$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	6028,80	2559,00	0,80	5895,60	2510,10	0,90	5893,20	2509,30	1,40
Desviación	3149,32	1340,39	0,60	3056,69	1304,57	0,54	3057,38	1305,02	0,92
$RG = 0,1$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	9335,60	3971,00	1,40	9298,10	3956,50	1,40	9295,80	3955,40	2,40
Desviación	10808,93	4652,67	1,69	10836,35	4662,44	1,91	10835,63	4662,02	3,04
$RG = 0,2$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	2045,90	906,00	0,30	2027,50	898,50	0,30	2027,50	898,50	0,20
Desviación	3261,49	1436,01	0,64	3230,10	1423,88	0,64	3230,10	1423,88	0,60

para las demás baterías de problemas, el h_2 resuelve los mismos problemas que los otros dos en menos tiempo, aunque expende unos pocos nodos más.

Con respecto a la dificultad de esta versión del problema frente a la versión con minimización del makespan, es claro que estos experimentos ponen de manifiesto la dificultad de la versión con minimización del flujo total. Para la versión $J||C_{max}$, los problemas contemplados en esta sección son instancias pequeñas que se resuelven óptimamente sin emplear y empleando la técnica de poda, expandiendo en muchos de ellos un número igual a 51 nodos. Para la versión tratada en este capítulo, sin poda hay 5 problemas que no se resuelven antes de agotar la memoria, y con poda, aún resolviéndose todos, el número de nodos expandidos para alcanzar la solución es mucho mayor en todos los casos.

Tabla 5.8: Resultados resumen para los problemas de *Sadeh* con $BK = 2$, agrupados por grupos según su parámetros RG, con los heurísticos h_2, h_3 y h_4, realizando podas

No Poda									
	h_2			h_3			h_4		
$RG = 0,0$, *Resueltos* 8/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	805375,25	395059,25	91,38	803212,50	394294,38	96,25	802716,13	394074,63	149,25
Desviación	833755,96	424844,18	96,15	831518,06	424153,01	101,34	830883,04	423871,17	158,00
$RG = 0,1$, *Resueltos* 8/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	509160,63	222872,88	56,63	508742,00	222688,38	60,63	508547,50	222625,13	98,00
Desviación	823087,70	341292,00	91,57	822929,69	341244,39	98,46	822500,82	341108,74	163,38
$RG = 0,2$, *Resueltos* 9/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	542520,56	260655,44	58,67	542170,78	260502,33	62,00	541994,44	260423,22	90,44
Desviación	1270740,71	606225,50	136,57	1270532,44	606147,49	144,59	1270080,96	605946,12	210,87
Poda									
	h_2			h_3			h_4		
$RG = 0,0$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	49907,00	22270,75	8,25	49650,13	22167,25	8,63	49628,25	22160,13	12,88
Desviación	53584,11	24935,85	10,40	53523,40	24911,32	10,93	53487,57	24900,40	15,19
$RG = 0,1$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	37804,88	16171,38	5,88	37740,88	16144,13	6,38	37754,63	16151,50	10,00
Desviación	46286,71	19273,89	7,88	46256,12	19266,25	8,17	46283,20	19279,68	12,86
$RG = 0,2$, *Resueltos* 10/10									
	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
Media	23093,22	10645,44	3,44	23050,56	10631,33	3,56	23050,00	10631,67	5,44
Desviación	33636,79	15348,45	5,19	35640,23	16269,99	5,79	33608,07	15343,12	8,22

5.5.4. Comparación con otros Métodos

Este grupo de experimentos tiene como objetivo comparar nuestra versión del algoritmo A^*, con la técnica de poda, con otros métodos. Concretamente haremos un estudio comparativo con un algoritmo genético y con un algoritmo de búsqueda local.

Para este grupo de experimentos se utilizan las instancias de la batería LA de tamaño 10×5 ($LA01 - 05$), así como los problemas reducidos de tamaños 8×8 y 9×9 que se obtienen de los problemas de tamaño 10×10 de las baterías: LA, ORB, ABZ y $Selectos$. Como hemos visto en experimentos anteriores, no somos capaces de resolver ninguna de las instancias de tamaño 10×10 sin recortar, este es el motivo por el que empleamos problemas reducidos para estos experimentos. Elegimos sólo las reducciones de los problemas que tienen tamaño 10×10 porque en la práctica son más difíciles que los que se obtienen de problemas de mayor tamaño. En el último conjunto de experimentos, consideramos el conjunto de instancias propuestas por Sadeh y Fox en [74]. Recordemos que se trata de un conjunto de 60 instancias de tamaño 10×5 organizadas por los parámetros BK y RG, en 6 grupos de 10 problemas cada uno.

Las versiones tanto del algoritmo genético como de la búsqueda local son ejecutadas en la misma máquina siendo, en este caso, el sistema operativo Windows XP Profesional. La relación de tiempos entre ambos sistemas operativos nos da que el sistema operativo Ubuntu es $2,2$ veces más rápido. Así, todos los tiempos que se tomen de referencia desde los experimentos realizados en Ubuntu serán multiplicados por $2,2$ para de este modo dar tanto al algoritmo genético como a la búsqueda local las mismas condiciones de tiempo en el sistema operativo Windows XP. Las Tablas 5.9, 5.10 y

Tabla 5.9: Tiempos empleados por el algoritmo A^* con poda y empleando h_2 en el SO Ubuntu y trasladados al SO Windows, para los problemas $LA01 - 05$.

Instancias	T.(s) Ubuntu	T.(s) Windows
$LA01$	35	77
$LA02$	80	176
$LA03$	10	22
$LA04$	19	41,8
$LA05$	68	149,6
Medias	42,4	93,28

Tabla 5.10: Medias de los tiempos empleados por el algoritmo A^* con poda y empleando h_2 en el SO Ubuntu y trasladados al SO Windows, para los grupos de problemas de $Sadeh$

Grupo Instancias	T. medio (s) Ubuntu	T. medio(s) Windows
$BK = 1, RG = 0, 0$	0,80	1,76
$BK = 2, RG = 0, 0$	8,25	18,15
$BK = 1, RG = 0, 1$	1,40	3,08
$BK = 2, RG = 0, 1$	5,88	12,93
$BK = 1, RG = 0, 2$	0,30	0,66
$BK = 2, RG = 0, 2$	3,44	7,58
Medias	3,34	7,36

5.11, recogen los tiempos de ejecución del algoritmo A^* en el sistema operativo Ubuntu para estos problemas, así como su transformación al sistema operativo Windows y la media de los mismos.

Ambos métodos se compararan con el algoritmo A^*, empleando el heurístico h_2 (mejor heurístico según el estudio experimental) y la técnica de poda por dominancia, dándole un tiempo límite de 3600 segundos. Cuando la mejor solución alcanzada por estos métodos no es la óptima se le pone delante un *.

Tabla 5.11: Tiempos empleados por el algoritmo A^* con poda y empleando h_2 en el *SO* Ubuntu y trasladados al *SO* Windows, para los problemas de tamaños 8×8 y 9×9.

8×8			9×9		
Instancias	T.(s) Ubuntu	T.(s) Windows	Instancias	T.(s) Ubuntu	T.(s) Windows
$LA16 - 8 \times 8$	4	8,8	$LA16 - 9 \times 9$	38	83,6
$LA17 - 8 \times 8$	6	13,2	$LA17 - 9 \times 9$	116	255,2
$LA18 - 8 \times 8$	3	6,6	$LA18 - 9 \times 9$	34	74,8
$LA19 - 8 \times 8$	3	6,6	$LA19 - 9 \times 9$	28	61,6
$LA20 - 8 \times 8$	5	11	$LA20 - 9 \times 9$	110	242
$ORB01 - 8 \times 8$	4	8,8	$ORB01 - 9 \times 9$	166	365,2
$ORB02 - 8 \times 8$	5	11	$ORB02 - 9 \times 9$	92	202,4
$ORB03 - 8 \times 8$	10	22	$ORB03 - 9 \times 9$	110	242
$ORB04 - 8 \times 8$	4	8,8	$ORB04 - 9 \times 9$	273	600,6
$ORB05 - 8 \times 8$	3	6,6	$ORB05 - 9 \times 9$	16	35,2
$ORB06 - 8 \times 8$	11	24,2	$ORB06 - 9 \times 9$	208	457,6
$ORB07 - 8 \times 8$	9	19,8	$ORB07 - 9 \times 9$	155	341
$ORB08 - 8 \times 8$	40	88	$ORB08 - 9 \times 9$	772	1698,4
$ORB09 - 8 \times 8$	20	44	$ORB09 - 9 \times 9$	38	83,6
$ORB10 - 8 \times 8$	1	2,2	$ORB10 - 9 \times 9$	106	233,2
$ABZ5 - 8 \times 8$	3	6,6	$ABZ5 - 9 \times 9$	39	85,8
$ABZ6 - 8 \times 8$	4	8,8	$ABZ6 - 9 \times 9$	29	63,8
$FT10 - 8 \times 8$	4	8,8	$FT10 - 9 \times 9$	72	158,4
Medias	7,72	16,98	Medias	133,44	293,57

Algoritmo Genético

El estudio experimental realizado hasta el momento pone de manifiesto que instancias fáciles de resolver en la versión del problema $J||C_{max}$, son difíciles en la versión tratada en este capítulo, $J||\sum C_i$. Para ver este efecto se comparará la técnica propuesta en este capítulo con un algoritmo genético, similar al propuesto para la minimización del makespan en [32], descrito en la sección 2.4.2. De hecho la única diferencia con este algoritmo se encuentra en la función de evaluación.

Recordemos las principales características de este algoritmo genético. El esquema de codificación se basa en permutaciones con repetición, como propuso Bierwirth en [8]. La función de evaluación se basa en el algoritmo de Giffler y Thomson (ver sección 2.4.2). La población inicial se genera aleatoriamente. Los operadores genéticos son el cruce del orden de los trabajos (JOX) y la mutación mediante el intercambio aleatorio de genes. La estrategia evolutiva consiste en seleccionar todos los cromosomas en cada generación y organizarlos aleatoriamente en parejas. Después se aplica a cada

pareja de cromosomas los operadores de cruce y mutación según las probabilidades de cruce y mutación. Finalmente se aplica el operador de selección por torneo a cada pareja de padres y a sus descendientes para quedarse con dos cromosomas que pasan a la siguiente generación. El tamaño de la población se fija con un parámetro, siendo el criterio de parada del algoritmo genético un número de generaciones dado, también, por un parámetro.

Todos los experimentos con el algoritmo genético se ejecutarán 20 veces para cada instancia, siendo este configurado con probabilidades de cruce y mutación de $0,8$ y $0,2$ respectivamente. Las tablas de estos experimentos recogen para cada instancia su solución óptima, la mejor solución alcanzada por el algoritmo genético, la media de las mejores soluciones alcanzadas en las 20 pruebas y el número de veces que se alcanzó la mejor solución.

En primer lugar resolvemos los problemas $LA01-05$ con el algoritmo genético. La configuración elegida para el algoritmo genético hará que este tarde aproximadamente de media en cada prueba la media de los tiempos que tarda el algoritmo A^* en resolver estos problemas, con la correspondiente traslación de este tiempo al sistema operativo Windows XP. Concretamente se configura el algoritmo genético con una población de 900 cromosomas y 3000 generaciones ($/0,8/0,2/900/3000/20/$). Esta configuración hará que se tenga un tiempo medio de $88,5$ segundos por prueba, tiempo muy similar a 93 segundos, media de ejecutar estos problemas con el algoritmo A^*. Los resultados de estos experimentos se recogen en la Tabla 5.12; para cada problema se muestran la solución óptima, la mejor solución, la media de las soluciones alcanzadas y el número de veces que se alcanzó la mejor solución. Como se observa para esta versión del problema con minimización del flujo total, el algoritmo genético no siempre encuentra la solución óptima, siendo las mejores soluciones de las 20 ejecuciones, en media muy diferentes a las óptimas. Sin embargo, para la minimización del makespan, el algoritmo alcanza la solución óptima en todas las ejecuciones (20) y para todas las instancias; siendo incluso capaz de alcanzar estas soluciones con menor número de generaciones y menos individuos en la población. Estos resultados dejan claro que la versión del problema con minimización del flujo total, independientemente del método de resolución empleado, es mucho mas difícil que la versión con minimización del makespan.

A pesar de no poder hacer una comparación estricta, en estos resultados se ve claramente que el algoritmo A^* empleando la técnica de poda es, para esta versión del problema, mejor método que el algoritmo genético; encuentra siempre la solución óptima, mientras que el algoritmo genético para las instancias $LA01$ y $LA05$ no es capaz de encontrarlas en ninguna de las 20 ejecuciones. Además, el algoritmo A^* emplea para ello un tiempo mucho menor (Tabla 5.9).

Tabla 5.12: Resultados con el AG sobre el subconjunto de instancias $LA01 - LA05$. Configuración del AG /0,8/0,2/300/2000/20/

Instancias	Opt.	Mejor	Media	Veces
$LA01$	4832	*4843	4850,4	16
$LA02$	4459	4459	4481,6	10
$LA03$	4151	4151	4167,9	5
$LA04$	4259	4259	4326,5	3
$LA05$	4072	*4082	4090,6	3

Tabla 5.13: Resultados con el AG sobre los subconjuntos de instancias recortadas de tamaños 8×8 y 9×9.

8×8, $AG(/0,8/0,2/400/750/20/)$					9×9, $AG(/0,8/0,2/1000/6000/20/)$				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
$LA16-8\times8$	4600	4600	4679	3	$LA16-9\times9$	5724	*5790	5794,3	19
$LA17-8\times8$	4366	4366	4408	1	$LA17-9\times9$	5390	5390	5430	5
$LA18-8\times8$	4690	*4732	4794	6	$LA18-9\times9$	5770	5770	5852	2
$LA19-8\times8$	4612	4612	4617,5	15	$LA19-9\times9$	5891	*5930	5946,4	8
$LA20-8\times8$	4616	4616	4659	6	$LA20-9\times9$	5915	*5951	5978	1
$ORB01-8\times8$	4743	4743	4786	4	$ORB01-9\times9$	6367	*6411	6425	9
$ORB02-8\times8$	4678	4678	4689	7	$ORB02-9\times9$	5867	*5900	5947	1
$ORB03-8\times8$	4925	4925	4927	18	$ORB03-9\times9$	6310	6310	6312	18
$ORB04-8\times8$	5081	5081	5188	9	$ORB04-9\times9$	6661	*6679	6770	1
$ORB05-8\times8$	4191	4191	4213	12	$ORB05-9\times9$	5605	5605	5688,3	6
$ORB06-8\times8$	4673	4673	4747,5	3	$ORB06-9\times9$	6106	6106	6109,2	18
$ORB07-8\times8$	2124	*2158	2160	18	$ORB07-9\times9$	2668	2668	2668,5	17
$ORB08-8\times8$	4749	4749	4792	4	$ORB08-9\times9$	5656	*5668	5671	19
$ORB09-8\times8$	4590	*4599	4610	3	$ORB09-9\times9$	6013	*6026	6052	1
$ORB10-8\times8$	4959	4959	4977,4	12	$ORB10-9\times9$	6328	6328	6334	6
$ABZ5-8\times8$	6818	*6850	6871	1	$ABZ5-9\times9$	8586	8586	8586	20
$ABZ6-8\times8$	4900	*4974	4974	20	$ABZ6-9\times9$	6524	6524	6524	20
$FT10-8\times8$	4559	4559	4573,4	2	$FT10-9\times9$	5982	5982	6007	14

Veamos que ocurre con los problemas cercanos al tamaño umbral, problemas de tamaño 8×8 y 9×9. En este caso se configura el algoritmo genético con una población de 400 cromosomas y 750 generaciones en el caso de los de tamaño 8×8 y con una población de 1000 cromosomas y 6000 generaciones para los de tamaño 9×9. Estas configuraciones harán que se tenga en media un tiempo de 17 segundos por prueba para los de tamaño 8×8 y 294 segundos por prueba para los de tamaño 9×9, medias de ejecutar estos problemas con el A*. Como se observa en la Tabla 5.13, el comportamiento es similar a lo que ocurría para los problemas $LA01-05$, hay 5 instancias de tamaño 8×8 y 8 de tamaño 9×9 que el algoritmo genético no resuelve nunca, para el resto de las instancias, el algoritmo genético alcanza la solución óptima en alguna de las 20 pruebas. Además, las medias de las soluciones encontradas están lejanas a la solución óptima en la mayoría de los casos. Claramente el algoritmo A^* también se comporta mejor para estos problemas, ya que los resuelve todos de forma óptima, empleando para ello un tiempo menor.

En el último conjunto de experimentos, consideramos el conjunto de instancias propuestas por Sadeh y Fox en [74]. Como se vio en los resultados de experimentos anteriores el algoritmo A^* con poda es capaz de resolver todas las instancias en menor número de nodos expandidos y tiempo. Sin poda hay 5 instancias que no es capaz de resolver. Todas las instancias que no se resolvieron sin poda se encuentran en los grupos de problemas con dos cuellos de botella $BK = 2$, por tanto, claramente estos problemas son más difíciles ya que, además, los cuellos de botella contribuyen a aumentar el factor de ramificación del espacio de búsqueda. Para las instancias de *Sadeh*, a pesar de ser del mismo tamaño que las instancias $LA01-05$, el algoritmo A^* necesita menos tiempo para alcanzar

una solución. El motivo de esto es que las máquinas, a parte de los cuellos de botella, se distribuyen uniformemente (a diferencia de las instancias LA en las que la distribución no es uniforme) a lo largo de las tareas de los trabajos, por lo que el factor de ramificación es menor. A pesar de ser por tanto instancias más fáciles de resolver que las LA se ha configurado el algoritmo genético de igual forma que se configuró para las instancias $LA01-05$ $(/0,8/0,2/300/2000/20/)$, siendo el tiempo medio de cada prueba $88,5$ segundos. Los resultados de estos experimentos con el algoritmo genético se recogen en las Tabla 5.14, en la que se muestra las instancias de *Sadeh* agrupadas por

Tabla 5.14: Resultados con el AG sobre los subconjuntos de instancias de *Sadeh*. Configuración del AG /0,8/0,2/300/2000/20/

$BK=1, RG=0,0$, *Resueltos* 10/10					$BK=2, RG=0,0$, *Resueltos* 8/10				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
*Sadeh*01	857	857	857	20	*Sadeh*11	988	*998	998	20
*Sadeh*02	850	850	850,2	19	*Sadeh*12	997	997	997,4	12
*Sadeh*03	911	911	911	20	*Sadeh*13	949	949	951,2	2
*Sadeh*04	832	832	832,3	19	*Sadeh*14	867	*870	871,2	14
*Sadeh*05	813	813	813,1	19	*Sadeh*15	853	853	854	16
*Sadeh*06	755	755	755	20	*Sadeh*16	866	866	866	20
*Sadeh*07	883	883	883,3	19	*Sadeh*17	945	945	945,7	7
*Sadeh*08	868	868	868	20	*Sadeh*18	914	*916	917	17
*Sadeh*09	766	766	766	20	*Sadeh*19	980	980	994	7
*Sadeh*10	863	863	863	20	*Sadeh*20	879	879	879	20
$BK=1, RG=0,1$, *Resueltos* 10/10					$BK=2, RG=0,1$, *Resueltos* 10/10				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
*Sadeh*21	869	869	869	20	*Sadeh*31	1004	1004	1004	20
*Sadeh*22	913	913	913	20	*Sadeh*32	1048	1048	1048	20
*Sadeh*23	911	911	911	20	*Sadeh*33	961	961	962,2	11
*Sadeh*24	871	871	871	20	*Sadeh*34	884	884	884	20
*Sadeh*25	837	837	837	20	*Sadeh*35	972	972	975	2
*Sadeh*26	774	774	775	10	*Sadeh*36	956	956	956	20
*Sadeh*27	906	906	906	20	*Sadeh*37	976	976	976	20
*Sadeh*28	960	960	961,8	11	*Sadeh*38	986	986	992,3	4
*Sadeh*29	804	804	804	20	*Sadeh*39	1006	1006	1006	20
*Sadeh*30	891	891	891	20	*Sadeh*40	943	943	947	5
$BK=1, RG=0,2$, *Resueltos* 8/10					$BK=2, RG=0,2$, *Resueltos* 9/10				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
*Sadeh*41	912	912	912	20	*Sadeh*51	1038	*1039	1039	20
*Sadeh*42	950	950	950,8	13	*Sadeh*52	1075	1075	1075	20
*Sadeh*43	921	921	921	20	*Sadeh*53	979	979	979	20
*Sadeh*44	924	*941	941,2	10	*Sadeh*54	950	950	951,7	11
*Sadeh*45	878	*881	881	20	*Sadeh*55	1026	1026	1026	20
*Sadeh*46	784	784	784,2	19	*Sadeh*56	1058	1058	1058	20
*Sadeh*47	932	932	932	20	*Sadeh*57	1039	1039	1039	20
*Sadeh*48	1011	1011	1014	6	*Sadeh*58	1034	1034	1039,3	7
*Sadeh*49	864	864	866,6	8	*Sadeh*59	1032	1032	1038	2
*Sadeh*50	902	902	902	20	*Sadeh*60	985	985	985,5	18

sus parámetros BK y RG. Como se ve 6 instancias no son resueltas nunca óptimamente, mientras que 31 instancias son resueltas optimamente en las 20 pruebas. Para el resto de instancias, 23, el algoritmo genético alcanza la solución óptima al menos una vez de las 20 pruebas. A la vista de estos experimentos se puede decir que el algoritmo A^* es un método muy competitivo con el algoritmo genético para la versión del problema con minimización del flujo total, ya que es capaz de resolver todas las instancias empleando para todas ellas un tiempo mucho menor que el empleado por el algoritmo genético.

Búsqueda Local

En esta sección nos proponemos comparar nuestro algoritmo A^* con una búsqueda local denominada camino aleatorio de paso largo ($LargeStepRandomWalk$) y propuesta por Kreipl en [47]. En principio, esta búsqueda local está ideada para el problema JSS con minimización del tardiness ponderado. No obstante, este problema se reduce al $J||\sum C_i$ si consideramos que todos los trabajos tienen el mismo peso y que todos los due dates tienen valor 0. En este estudio experimental hemos empleado una implementación de este heurístico, para el problema $J||\sum C_i$, incluido en la herramienta $LEKIN$® ([66])que está disponible en la siguiente dirección: *http://www.stern.nyu.edu/om/software/lekin/index.htm*.

Esta búsqueda local consiste en sucesivas fases de intensificación de fases, denominadas pequeños pasos, y de fases de diversificación de fases, o pasos largos. Los pasos largos emplean un algoritmo de metropolis y por lo tanto aceptan soluciones peores para poder escapar de óptimos locales, mientras que el paso corto emplea un método de mínimo gradiente por lo que siempre alcanza un óptimo local. La estructura de vecindad propuesta en [47] es similar a las estructuras de vecindad definidas por Taillard en [85] y por Dell'Amico y Trubian en [24] para la minimización del makespan y basadas en cambios en el camino crítico. Sin embargo, a diferencia del caso de minimización del makespan donde sólo hay un camino crítico, para la minimización del flujo total puede haber tantos caminos críticos como número de trabajos. Por lo tanto, en general el número de vecinos es mucho mayor y en la práctica el método no es tan eficiente como su aplicación a la minimización del makespan.

Todos los experimentos con esta búsqueda local se ejecutarán 20 veces para cada instancia. Los resultados obtenidos de estos experimentos se recogen en una serie de tablas en las que se muestra para cada instancia su solución óptima, la mejor solución alcanzada por la búsqueda local, la media de las mejores soluciones alcanzadas en las 20 ejecuciones y el número de veces que se alcanzó la mejor solución.

Tabla 5.15: Resultados con el algoritmo BL sobre las instancias $LA01 - LA05$

Instancias	Opt.	Mejor	Media	Veces
$LA01$	4832	4832	4832,9	1
$LA02$	4459	*4479	4483,2	4
$LA03$	4151	4151	4151,0	20
$LA04$	4259	4259	4268,8	2
$LA05$	4072	4072	4095,0	2

Comenzamos en primer lugar resolviendo las instancias $LA01-05$, mediante la búsqueda local. La Tabla 5.15 recoge los resultados obtenidos para estas instancias con un tiempo límite de 93 segundos. Este es el tiempo medio empleado por el algoritmo A^* para resolver dichas instancias. Como se puede observar, todas las instancias menos la $LA02$ son resueltas de forma óptima en alguna de las ejecuciones. Sin embargo, sólo una se resuelve óptimamente siempre, la $LA03$, mientras que para el resto el número de veces en que se resuelven es pequeño, estando la media de las soluciones alejada de la solución óptima. Estos resultados muestran que el algoritmo A^* empleando la técnica de poda es, para esta versión del problema, mejor método que la búsqueda local; ya que encuentra siempre la solución óptima empleando para ello un tiempo mucho menor (Tabla 5.9).

Veamos ahora que ocurre con los problemas cercanos al tamaño umbral, problemas de tamaño 8×8 y 9×9. En este caso se le da a la búsqueda local un tiempo de 17 segundos por ejecución para los de tamaño 8×8 y 294 segundos para los de tamaño 9×9. Como se observa en la Tabla 5.16, ocurre algo parecido a lo que ocurría para las instancias $LA01-05$. Hay 2 instancias de tamaño 8×8 ($LA17-8\times8$ y $ABZ5-8\times8$) que no son resueltas nunca, mientras que 11 instancias son resueltas siempre en las 20 ejecuciones. Para el resto de instancias la búsqueda local encuentra la solución óptima al menos en una de las 20 ejecuciones. En el caso de las instancias de tamaño 9×9, hay 4 ($LA17-9\times9$, $LA20-9\times9$, $ORB08-9\times9$ y $ORB10-9\times9$) que la búsqueda local no resuelve nunca, 9 que son resueltas siempre en las 20 ejecuciones, y el resto que son resueltas al menos una vez. Para ambos tamaños, se puede observar que las medias de las soluciones encontradas, para las instancias no resueltas, están lejanas a la solución óptima. Claramente el algoritmo A^* también se

Tabla 5.16: Resultados con el algoritmo BL sobre los subconjuntos de instancias recortadas de tamaños 8×8 y 9×9.

8×8, $BL(17s)$					9×9, $BL(294s)$				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
$LA16-8\times8$	4600	4600	4606,45	3	$LA16-9\times9$	5724	5724	5739,55	6
$LA17-8\times8$	4366	*4379	4379	20	$LA17-9\times9$	5390	*5396	5403,5	5
$LA18-8\times8$	4690	4690	4704,7	13	$LA18-9\times9$	5770	5770	5770	20
$LA19-8\times8$	4612	4612	4612	20	$LA19-9\times9$	5891	5891	5891	20
$LA20-8\times8$	4616	4616	4616	20	$LA20-9\times9$	5915	*5934	5935,2	12
$ORB01-8\times8$	4743	4743	4743	20	$ORB01-9\times9$	6367	6367	6378,5	5
$ORB02-8\times8$	4678	4678	4678	20	$ORB02-9\times9$	5867	5867	5867,85	3
$ORB03-8\times8$	4925	4925	4925	20	$ORB03-9\times9$	6310	6310	6310	20
$ORB04-8\times8$	5081	5081	5081	20	$ORB04-9\times9$	6661	6661	6661	20
$ORB05-8\times8$	4191	4191	4192,5	5	$ORB05-9\times9$	5605	5605	5605	20
$ORB06-8\times8$	4673	4673	4673	20	$ORB06-9\times9$	6106	6106	6106	20
$ORB07-8\times8$	2124	2124	2124	20	$ORB07-9\times9$	2668	2668	2668	20
$ORB08-8\times8$	4749	4749	4759,9	6	$ORB08-9\times9$	5656	*5668	5693,3	2
$ORB09-8\times8$	4590	4590	4590	20	$ORB09-9\times9$	6013	6013	6013,75	18
$ORB10-8\times8$	4959	4959	4959	20	$ORB10-9\times9$	6328	6333	6332,75	1
$ABZ5-8\times8$	6818	*6839	6891	4	$ABZ5-9\times9$	8586	8586	8586	20
$ABZ6-8\times8$	4900	4900	4922,5	2	$ABZ6-9\times9$	6524	6524	6524,6	14
$FT10-8\times8$	4559	4559	4559	20	$FT10-9\times9$	5982	5982	5982	20

comporta mejor para estos problemas, ya que los resuelve todos de forma óptima, empleando para ello un tiempo menor.

Por último, resolvemos con la búsqueda local las 60 instancias propuestas por Sadeh y Fox en [74]. Al igual que hicimos con el algoritmo genético emplearemos como tiempo límite 93 segundos, media empleada para los problemas $LA01-05$ de su mismo tamaño, a pesar de ser su tiempo medio de resolución bastante inferior a este valor. La Tabla 5.17 recoge los resultados de estos experimentos para los problemas de *Sadeh* agrupados por sus parámetros BK y RG. Como se

Tabla 5.17: Resultados con el algoritmo BL sobre los subconjuntos de instancias de *Sadeh*

$BK=1, RG=0,0$, *Resueltos* 7/10					$BK=2, RG=0,0$, *Resueltos* 9/10				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
*Sadeh*01	857	*858	860,1	5	*Sadeh*11	988	988	988,55	13
*Sadeh*02	850	*855	857,4	1	*Sadeh*12	997	997	997,15	18
*Sadeh*03	911	911	911	20	*Sadeh*13	949	*953	961,65	1
*Sadeh*04	832	*833	834,6	3	*Sadeh*14	867	*869	870,55	2
*Sadeh*05	813	813	813,5	10	*Sadeh*15	853	853	853,9	11
*Sadeh*06	755	755	755,85	5	*Sadeh*16	866	866	866,1	19
*Sadeh*07	883	883	883,45	11	*Sadeh*17	945	945	945,85	4
*Sadeh*08	868	868	868,45	13	*Sadeh*18	914	914	916	5
*Sadeh*09	766	766	766	20	*Sadeh*19	980	980	980,65	7
*Sadeh*10	863	863	863,9	6	*Sadeh*20	879	879	879	20
$BK=1, RG=0,1$, *Resueltos* 8/10					$BK=2, RG=0,1$, *Resueltos* 9/10				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
*Sadeh*21	869	869	869	20	*Sadeh*31	1004	1004	1004,2	16
*Sadeh*22	913	913	913,25	15	*Sadeh*32	1048	1048	1049,35	11
*Sadeh*23	911	911	911	20	*Sadeh*33	961	*968	970,85	6
*Sadeh*24	871	*873	872,8	16	*Sadeh*34	884	884	884	20
*Sadeh*25	837	*838	838	20	*Sadeh*35	972	972	972	20
*Sadeh*26	774	774	774	20	*Sadeh*36	956	956	957,05	9
*Sadeh*27	906	906	906	20	*Sadeh*37	976	976	976,1	19
*Sadeh*28	960	960	960,5	10	*Sadeh*38	986	986	986	20
*Sadeh*29	804	804	804,55	9	*Sadeh*39	1006	1006	1007	10
*Sadeh*30	891	891	891,95	1	*Sadeh*40	943	943	943,65	8
$BK=1, RG=0,2$, *Resueltos* 8/10					$BK=2, RG=0,2$, *Resueltos* 8/10				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
*Sadeh*41	912	912	912	20	*Sadeh*51	1038	1038	1038	20
*Sadeh*42	950	950	950	20	*Sadeh*52	1075	1075	1075	20
*Sadeh*43	921	921	921	20	*Sadeh*53	979	979	981,7	1
*Sadeh*44	924	924	927,25	3	*Sadeh*54	950	*952	955,95	1
*Sadeh*45	878	878	881,2	2	*Sadeh*55	1026	1026	1026	20
*Sadeh*46	784	784	784	20	*Sadeh*56	1058	1058	1059,05	9
*Sadeh*47	932	932	933,35	5	*Sadeh*57	1039	*1040	1041,25	7
*Sadeh*48	1011	*1012	1012,1	18	*Sadeh*58	1034	1034	1034	20
*Sadeh*49	864	864	865,05	7	*Sadeh*59	1032	1032	1032	20
*Sadeh*50	902	*903	903,75	5	*Sadeh*60	985	985	986,85	7

ve, hay 12 instancias que no son resueltas en ninguna de las 20 ejecuciones, 19 instancias que son resueltas siempre y el resto que se resuelven alguna vez de las 20 ejecuciones. Como se observa en los problemas de *Sadeh*, las medias de las soluciones son más cercanas a la solución óptima, esto es debido a que estas instancias son más fáciles de resolver. Hay que tener en cuenta que para todos estos experimentos, el tiempo límite es un margen muy superior a la media de tiempo que tarda el algoritmo A^* para los mismos problemas, por lo tanto se le está dando a la búsqueda local una ventaja importante.

Para todas las instancias contempladas en estos experimentos, el algoritmo A^* alcanza la solución óptima, mientras que la búsqueda local no es capaz de resolver las mismas instancias aún dándole un tiempo mucho mayor. Por supuesto, la búsqueda local, al igual que el algoritmo genético, es capaz de proporcionar una solución no óptima, cosa que esta versión del algoritmo A^* no hace. En capítulos posteriores se mostrarán versiones del algoritmo A^* que permitirán el cálculo de cotas superiores de la solución.

Si comparamos la búsqueda local con el algoritmo genético, podemos decir que en general tienen un comportamiento similar. Para los problemas $LA01-05$ y los problemas de tamaños 8×8 y 9×9 se comporta mejor la búsqueda local pues sólo deja sin resolver óptimamente 7 problemas, mientras que el genético deja 15. Sin embargo, para los problemas de *Sadeh* es mejor el algoritmo genético que deja sin resolver óptimamente 6, mientras que la búsqueda local deja 12.

5.6. Conclusiones

En este capítulo hemos formulado el problema JSS con minimización del flujo total, $J||\sum C_i$, en el marco de la búsqueda en espacios de estados, en concreto en el marco del algoritmo A^*. Para ello, al igual que en la versión $J||C_{max}$, hemos utilizado el espacio de búsqueda de las planificaciones activas, y hemos diseñado una serie de heurísticos admisibles y en algunos casos monótonos, basados en distintas relajaciones del problema. La principal aportación de este capítulo es ver que la idea de poda por dominancia, que permite reducir tanto el espacio de búsqueda como el tiempo de ejecución, es aplicable a diferentes versiones del problema. Para ello en este capítulo la hemos formulado y descrito para aplicarla a la versión del problema con minimización del flujo total. Por último, hemos presentado los resultados de un estudio experimental en el que, al igual que en el capítulo anterior, se comparan los heurísticos, se analiza la eficiencia de distintas versiones de la técnica de poda por dominancia y se muestra el umbral del tamaño de las instancias que el método permite resolver.

Los resultados experimentales muestran que el mejor de los heurísticos para esta versión del problema y los tamaños de problemas que podemos abordar, es el heurístico h_2 (basado en el cálculo de planificaciones tipo JPS en las que la prioridad es el tiempo remanente). Este heurístico, a pesar de expandir algunos nodos más, resuelve los mismos problemas que los heurísticos h_3 y h_4 empleando menor tiempo de ejecución.

Hemos comprobado que el límite del tamaño de los problemas que podemos resolver óptimamente, gracias al empleo de la técnica de poda por dominancia, es para esta versión del problema un poco menor que para la versión con minimización del makespan, aunque se resuelve alguna instancia del umbral de tamaño que deja de considerarse pequeño, es decir 10×10, el límite en función de

los resultados experimentales está en este caso mucho más cerca del tamaño 9×9. Además, no se resuelven los problemas $LA06 - 15$ que sí se resuelven cuando se minimiza el makespan.

Hemos visto que nuestra aproximación con el algoritmo A^*, empleando la técnica de poda, tiene un comportamiento mejor en comparación tanto con un algoritmo genético como con la búsqueda local propuesta en [47]. A pesar de dar ventaja de tiempo a estos métodos, no fueron capaces de resolver todas las instancias, mientras que el algoritmo A^* con la técnica de poda resuelve todas las instancias empleando menor tiempo.

Las ideas más importantes presentadas en este capítulo se recogen en [82, 79, 80]

En el capítulo siguiente proponemos un estudio experimental que nos permita por un lado, incorporar junto a la técnica de poda, en ambas versiones del problema, el cálculo de cotas superiores, con el objetivo de tratar de reducir el espacio de búsqueda; y por otro, relajar la restricción de optimalidad de la solución empleando tanto la versión híbrida del algoritmo $G\&T$ (sección 2.4.1) como la ponderación de la función heurística (Algoritmos ε-admisibles, sección 3.4.2).

5.7. Resumen de Notación

- $J = \{J_1, \ldots, J_N\}$, Conjunto de N trabajos.
- $R = \{R_1, \ldots, R_M\}$, Conjunto de M recursos o máquinas físicas.
- $J_i = \{\theta_{i1}, \ldots, \theta_{iM}\}$, Conjunto de tareas u operaciones de que consta el trabajo J_i.
- $R_{\theta_{il}}$, Máquina que requiere la tarea θ_{il}.
- $p_{\theta_{il}}$, Duración de la tarea θ_{il}.
- $r_{\theta_{il}}$, Cabeza de la tarea θ_{il}.
- $q_{\theta_{il}}$, Cola de la tarea θ_{il}.
- $st_{\theta_{il}}$, Tiempo de inicio de la tarea θ_{il}.
- PJ_v, Predecesor de la tarea v en la secuencia de su trabajo.
- SJ_v, Sucesor de la tarea v en la secuencia de su trabajo.
- $SC(n)$, Tareas planificadas en el estado n.
- $US(n)$, Tareas sin planificar en el estado n.
- $P(n) = (US(n),\ \mathbf{r}_n(US(n)), \mathbf{p}_n(US(n)), \mathbf{q}_n(US(n)))$, Problema residual representado por las tareas sin planificar en el estado n, sus cabezas, duraciones y colas.
- $J_{US}(n) = \{J_i \in J; \exists j, 1 \leq j \leq M, \theta_{ij} \in US(n)\}$, Trabajos con tareas sin planificar en el estado n.
- $J_{SC}(n) = J \backslash J_{US}(n)$, Trabajos con todas sus tareas planificadas en el estado n.
- $g(n) = \sum_{J_i \in J_{SC}(n)} (r_{\theta_{iM}} + p_{\theta_{iM}}) + \sum_{J_i \in J_{US}(n)} (r_{\theta_{ik_i}} + p_{\theta_{ik_i}})$, Valor $g(n)$: suma de los tiempos de completud de las últimas tareas planificadas de cada trabajo.
- $k_i = (j; \theta_{ij} \in SC(n), \theta_{ij+1} \in US(n))$, índice de la última tarea planificada del trabajo $J_i \in J_{US}(n)$.
- $J_m = \{J_i \in J; \exists j, 1 \leq j \leq M,\ t.q.\ \theta_{ij} \in US(n)\ \ y\ R_{\theta_{ij}} = m\}$, Subconjunto de trabajos con alguna operación en $US(n)$ que requiere la máquina m.
- $J_{\overline{m}} = \{J_i \in J; \forall j, 1 \leq j \leq M, R_{\theta_{ij}} \neq m\ si\ \theta_{ij} \in US(n)\}$, Subconjunto de trabajos con operaciones en $US(n)$ y que no requieren la máquina m.
- $US(n)|_{J_m} = \{\theta \in US(n); \theta \in J_m\}$, Subconjunto de tareas no planificadas de J_m.
- $US(n)|_{J_{\overline{m}}} = \{\theta \in US(n); \theta \in J_{\overline{m}}\}$, Subconjunto de tareas no planificadas de $J_{\overline{m}}$.
- $P(n)|_{J_m} = (US(n)|_{J_m}, \mathbf{r}_n(US(n)|_{J_m}), \mathbf{p}_n(US(n)|_{J_m}), \mathbf{q}_n(US(n)|_{J_m}))$, Subproblema de $P(n)$, formado por las tareas $US(n)|_{J_m}$, sus cabezas, duraciones y colas.

- $P(n)|_{J_{\overline{m}}} = (US(n)|_{J_{\overline{m}}}, \mathbf{r}_n(US(n)|_{J_{\overline{m}}}), \mathbf{p}_n(US(n)|_{J_{\overline{m}}}), \mathbf{q}_n(US(n)|_{J_{\overline{m}}}))$, Subproblema de $P(n)$, formado por las tareas $US(n)|_{J_{\overline{m}}}$, sus cabezas, duraciones y colas.
- $US(n)|_m$, Conjunto de tareas no planificadas en el estado n que requieren a la máquina m.
- $P(n)|_m = (US(n)|_m, \mathbf{r}_n(US(n)|_m), \mathbf{p}_n(US(n)|_m), \mathbf{q}_n(US(n)|_m))$, Problema OMS formado por las tareas $US(n)|_m$, sus cabezas, duraciones y colas.
- θ_i^m, Tarea del trabajo J_i que requiere la máquina m.
- $C_{\theta_i{}^m}$, Tiempo de completud de la tarea θ_i^m en la planificación con expulsión calculada para el problema $P(n)|_m$.
- $T_u = \text{máx}(0, C_u - d_u)$, Tardiness de la tarea u, donde C_u es el tiempo en el que la tarea u es completada y d_u su due date.
- $P'(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m),\ \mathbf{p}_n(US(n)|_m),\ \mathbf{d}_n(US(n)|_m))$, Problema OMS con cabezas, due dates (tiempos máximos de fin sin incurrir en penalización) y minimización del tardiness (tiempo de retardo de todos los trabajos).
- $T_{\theta_i{}^m}$, Tardiness de la tarea θ_i^m en la planificación con expulsión calculada para el problema $P'(n)|_m$.
- C_{max}, Tiempo de completud de una planificación activa.
- A_u, Conjunto de operaciones que se ha determinado que tienen que empezar después de que se complete la tarea u.
- $\delta_u = C_{max} - \Sigma_{u \in A_u} p_u$, Deadline (tiempo límite de fin) para la tarea u.
- B_u, Conjunto de tareas que tienen que terminar antes de que la tarea u pueda empezar.
- $C_{max}(B_u)$, Tiempo de fin de una planificación activa de las tareas de B_u.
- $r'_u = \text{máx}(r'_u, C_{max}(B_u))$, Tiempo de inicio de una tarea u ajustado.
- $T'_{\theta_i^m}$, Tardiness de la tarea θ_i^m en la planificación con expulsión calculada empleando las reglas de Emmons generalizadas para el problema $P'(n)|_m$.

5.8. Anexo. Tablas

5.8.1. Heurístico h_2

Tabla 5.18: Resultados con h_2 para una serie de problemas de tamaño 6×6: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 6 \times 6$	2656	508	296	0	432	255	0
$LA17 - 6 \times 6$	2392	3499	1904	0	1382	743	0
$LA18 - 6 \times 6$	2393	1106	677	0	614	377	0
$LA19 - 6 \times 6$	2617	730	450	0	408	257	0
$LA20 - 6 \times 6$	2374	483	309	0	392	250	0
$ORB01 - 6 \times 6$	2574	1362	824	0	1019	614	1
$ORB02 - 6 \times 6$	2778	1778	1080	0	1125	668	0
$ORB03 - 6 \times 6$	2759	591	352	0	377	219	0
$ORB04 - 6 \times 6$	2894	1304	757	0	747	431	0
$ORB05 - 6 \times 6$	2236	503	301	0	390	237	0
$ORB06 - 6 \times 6$	2730	2096	1210	0	1288	739	0
$ORB07 - 6 \times 6$	1215	946	551	0	860	499	0
$ORB08 - 6 \times 6$	2355	3962	2236	1	1591	883	0
$ORB09 - 6 \times 6$	2312	383	239	0	328	203	0
$ORB10 - 6 \times 6$	2889	624	391	0	431	273	0
$ABZ5 - 6 \times 6$	3767	659	377	0	492	281	0
$ABZ6 - 6 \times 6$	2971	368	223	0	263	162	0
$ABZ7 - 6 \times 6$	1109	315	193	0	306	188	0
$ABZ8 - 6 \times 6$	1365	703	427	0	429	266	0
$ABZ9 - 6 \times 6$	1164	249	166	0	249	166	0
$FT10 - 6 \times 6$	2544	1352	763	1	503	293	0
$LA21 - 6 \times 6$	2613	506	304	0	397	243	0
$LA24 - 6 \times 6$	2514	733	454	0	555	339	1
$LA25 - 6 \times 6$	2598	2127	1234	0	1299	762	0
$LA27 - 6 \times 6$	2416	1670	1008	0	1085	653	0
$LA29 - 6 \times 6$	2353	313	195	0	232	144	0
$LA38 - 6 \times 6$	2602	776	455	0	602	360	0
$LA40 - 6 \times 6$	2211	1121	686	0	650	401	0
$YN1 - 6 \times 6$	1418	415	230	0	352	196	0
$YN2 - 6 \times 6$	1545	2903	1771	1	1123	650	0
$YN3 - 6 \times 6$	1436	2041	1203	0	1295	774	0
$YN4 - 6 \times 6$	1382	2100	1305	0	1314	811	0
Media		1194,56	705,34	0,09	704,06	416,78	0,06
Desviación		934,73	532,47	0,29	402,03	228,27	0,24

Tabla 5.19: Resultados con h_2 para una serie de problemas de tamaño 7×7: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16-7\times7$	3554	2929	1722	0	1570	920	1
$LA17-7\times7$	3488	30457	17524	4	7281	4068	1
$LA18-7\times7$	3496	24682	14376	3	5883	3454	0
$LA19-7\times7$	3627	5551	3224	0	2177	1270	0
$LA20-7\times7$	3498	11661	7076	2	5013	3070	0
$ORB01-7\times7$	3589	11534	6743	1	5675	3325	1
$ORB02-7\times7$	3727	16392	10283	2	7235	4470	1
$ORB03-7\times7$	3754	49873	28494	6	11451	6418	2
$ORB04-7\times7$	3803	13633	8081	2	4991	2918	0
$ORB05-7\times7$	2939	7108	4347	1	4215	2551	0
$ORB06-7\times7$	3480	23111	13586	2	7116	4112	1
$ORB07-7\times7$	1747	79222	46048	10	22692	13096	3
$ORB08-7\times7$	3647	415338	231427	47	36525	19930	5
$ORB09-7\times7$	3428	6397	3929	1	3066	1796	0
$ORB10-7\times7$	3624	1686	1060	0	1264	797	0
$ABZ5-7\times7$	5144	4300	2535	0	1687	956	0
$ABZ6-7\times7$	3936	4194	2614	1	2694	1678	0
$ABZ7-7\times7$	1764	10213	5803	1	5022	2868	1
$ABZ8-7\times7$	1886	3453	1930	0	2323	1312	0
$ABZ9-7\times7$	1577	1240	767	1	1108	694	0
$FT10-7\times7$	3420	13929	8059	1	4135	2390	1
$LA21-7\times7$	3499	8675	5193	1	3623	2218	0
$LA24-7\times7$	3442	7156	4206	1	3192	1880	1
$LA25-7\times7$	3581	8855	5058	1	4455	2571	0
$LA27-7\times7$	3219	4788	3062	1	2454	1553	1
$LA29-7\times7$	3383	13600	8127	2	3794	2303	0
$LA38-7\times7$	3407	2352	1428	0	1568	958	1
$LA40-7\times7$	3128	10149	6426	1	2684	1647	0
$YN1-7\times7$	1947	12991	7810	2	4831	2859	1
$YN2-7\times7$	1988	19017	11309	2	5783	3397	0
$YN3-7\times7$	1883	9275	5603	1	4320	2626	1
$YN4-7\times7$	1883	5144	2974	1	2748	1581	0
	Media	26215,78	15025,75	3,06	5705,47	3302,69	0,69
	Desviación	71527,27	39848,97	8,12	6760,53	3722,54	1,04

Tabla 5.20: Resultados con h_2 para una serie de problemas de tamaño 8×8: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16-8\times8$	4600	88212	51836	14	23071	13591	4
$LA17-8\times8$	4366	377200	219058	52	38998	22712	6
$LA18-8\times8$	4690	97407	57048	13	15607	9248	3
$LA19-8\times8$	4612	84918	50181	12	19803	11783	3
$LA20-8\times8$	4616	130574	75250	18	28934	17096	5
$ORB01-8\times8$	4743	101092	56133	15	25032	13500	4
$ORB02-8\times8$	4678	74660	44477	11	29813	17744	5
$ORB03-8\times8$	4925	609635	351815	90	56924	31195	10
$ORB04-8\times8$	5081	77563	43995	11	21175	11890	4
$ORB05-8\times8$	4191	62884	37372	9	17313	10348	3
$ORB06-8\times8$	4673	390831	219299	57	63480	35278	11
$ORB07-8\times8$	2124	230587	132772	34	47802	27470	9
$ORB08-8\times8$	4749	**3724438**	**2019639**	**540**	226924	121175	40
$ORB09-8\times8$	4590	670462	420077	95	118489	73503	20
$ORB10-8\times8$	4959	5861	3602	1	3670	2245	1
$ABZ5-8\times8$	6818	84979	47297	12	18889	10386	3
$ABZ6-8\times8$	4900	99476	61469	14	22603	13832	4
$ABZ7-8\times8$	2257	127357	69570	18	30269	16959	5
$ABZ8-8\times8$	2496	114989	63073	17	31448	17475	5
$ABZ9-8\times8$	2213	40064	23499	6	17241	10117	3
$FT10-8\times8$	4559	153099	91812	23	25996	14934	4
$LA21-8\times8$	4459	140537	81561	19	29974	17720	5
$LA24-8\times8$	4641	243020	137602	34	44503	25565	7
$LA25-8\times8$	4750	232396	130132	33	52743	30040	9
$LA27-8\times8$	4381	45586	28188	6	14358	8752	2
$LA29-8\times8$	4540	95650	53714	15	19005	10929	3
$LA38-8\times8$	4439	72093	41309	11	31128	18049	6
$LA40-8\times8$	4090	87982	51853	12	29930	18053	5
$YN1-8\times8$	2480	64461	38964	9	17340	10403	3
$YN2-8\times8$	2510	33060	19523	5	11094	6566	2
$YN3-8\times8$	2459	67641	40324	9	23712	14125	4
$YN4-8\times8$	2618	194313	113475	28	30248	17688	5
	Media	158019,00	92138,06	22,68	30986,84	18038,58	5,26
	Desviación	154376,88	92314,80	22,21	20768,93	12513,98	3,55

Tabla 5.21: Resultados con h_2 para una serie de problemas de tamaño 9×9: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 9 \times 9$	5724	1826477	1038587	301	188554	109407	38
$LA17 - 9 \times 9$	5390	**3181614**	**1797729**	**516**	511390	292217	116
$LA18 - 9 \times 9$	5770	1214536	685150	199	164228	94906	34
$LA19 - 9 \times 9$	5891	1443075	868511	231	138468	83634	28
$LA20 - 9 \times 9$	5915	**3207575**	**1851608**	**510**	521255	305809	110
$ORB01 - 9 \times 9$	6367	**3057495**	**1701337**	**528**	747039	422160	166
$ORB02 - 9 \times 9$	5867	**3028495**	**1672149**	**493**	442811	253923	92
$ORB03 - 9 \times 9$	6310	**2797314**	**1441036**	**493**	491935	259311	110
$ORB04 - 9 \times 9$	6661	**3317823**	**1961773**	**569**	1132648	660501	273
$ORB05 - 9 \times 9$	5605	301189	177225	49	82378	48079	16
$ORB06 - 9 \times 9$	6106	**2832347**	**1475981**	**489**	861588	467739	208
$ORB07 - 9 \times 9$	2668	**3241463**	**1885454**	**548**	593738	346444	155
$ORB08 - 9 \times 9$	5668	**2547716**	**1191485**	**449**	1765913	909440	772
$ORB09 - 9 \times 9$	6013	926860	566732	156	187332	111130	38
$ORB10 - 9 \times 9$	6328	**3066050**	**1710111**	**514**	488453	277685	106
$ABZ5 - 9 \times 9$	8586	1598526	935658	256	199939	114915	39
$ABZ6 - 9 \times 9$	6524	872162	511966	143	145908	85627	29
$ABZ7 - 9 \times 9$	2767	2244692	1269841	362	329613	190759	72
$ABZ8 - 9 \times 9$	3078	**3206471**	**1850400**	**520**	927825	535069	286
$ABZ9 - 9 \times 9$	2752	848919	477011	139	134357	75743	27
$FT10 - 9 \times 9$	5982	**3164126**	**1807912**	**549**	328803	183795	72
$LA21 - 9 \times 9$	5428	1738142	1038551	281	154049	92998	30
$LA24 - 9 \times 9$	5650	2226751	1325281	356	238216	142497	48
$LA25 - 9 \times 9$	5880	**3045403**	**1689473**	**498**	329322	186841	68
$LA27 - 9 \times 9$	5608	1987140	1152943	324	190610	109979	38
$LA29 - 9 \times 9$	5732	**3210879**	**1854931**	**519**	377399	221186	77
$LA38 - 9 \times 9$	5855	3127964	1782402	501	453546	262083	94
$LA40 - 9 \times 9$	5006	2623898	1447383	419	297779	172451	62
$YN1 - 9 \times 9$	3187	**3264361**	**1908481**	**511**	201631	118289	41
$YN2 - 9 \times 9$	3198	1791556	1025642	285	295710	171328	62
$YN3 - 9 \times 9$	3206	2118559	1239829	336	371202	217707	80
$YN4 - 9 \times 9$	3257	**3181088**	**1825140**	**519**	396240	229621	87
	Media	1680652,88	971419,50	271,13	223243,06	130202,69	45,94
	Desviación	708006,25	399947,91	112,32	97005,11	56057,24	21,16

Tabla 5.22: Resultados con h_2 para los únicos problema resueltos de tamaño 10×10: problemas recortados de las batería *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$ABZ7 - 10 \times 10$	3262	**2639236**	**1538304**	**524**	552246,00	326967,00	160,00
$ABZ9 - 10 \times 10$	3483	**2422071**	**1320912**	**489**	1050968,00	581996,00	382,00
$LA38 - 10 \times 10$	7113	**2461384**	**1360366**	**498**	937039,00	536287,00	265,00

Tabla 5.23: Resultados con h_2 para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*01	857	6517	2780	1	2930	1246	0
*Sadeh*02	850	14109	6092	2	4614	1937	1
*Sadeh*03	911	48104	20814	5	8778	3553	1
*Sadeh*04	832	9983	4580	1	4707	2106	1
*Sadeh*05	813	4579	2064	0	1338	561	0
*Sadeh*06	755	59458	25338	7	9279	3903	1
*Sadeh*07	883	23484	9560	3	3735	1448	1
*Sadeh*08	868	53600	25843	6	6873	3143	1
*Sadeh*09	766	105669	49471	11	5710	2437	0
*Sadeh*10	863	264076	122366	31	12324	5256	2
	Media	58957,90	26890,80	6,70	6028,80	2559,00	0,80
	Desviación	74684,40	34776,77	8,71	3149,32	1340,39	0,60

Tabla 5.24: Resultados con h_2 para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*11	988	**3984890**	**1814402**	**485**	264200	111970	57
*Sadeh*12	997	673885	326113	74	13541	5719	2
*Sadeh*13	949	1521394	744668	170	186410	86132	35
*Sadeh*14	867	123435	53236	14	21267	8416	3
*Sadeh*15	853	753985	327819	86	54474	23168	8
*Sadeh*16	866	2689170	1370760	311	44123	18602	7
*Sadeh*17	945	294580	157280	32	23518	11264	3
*Sadeh*18	914	241296	115871	28	43459	19546	7
*Sadeh*19	980	145257	64727	16	12464	5319	1
*Sadeh*20	879	**3807399**	**1635382**	**424**	158454	65448	28
	Media	805375,25	395059,25	91,38	49907,00	22270,75	8,25
	Desviación	833755,96	424844,18	96,15	53584,11	24935,85	10,40

Tabla 5.25: Resultados con h_2 para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*21	869	6807	2892	1	4027	1711	0
*Sadeh*22	913	204100	83788	23	23099	9520	4
*Sadeh*23	911	5026	2145	1	2871	1215	0
*Sadeh*24	871	6676	3004	0	1882	827	0
*Sadeh*25	837	2400	1001	1	1371	564	1
*Sadeh*26	774	9766	4517	1	3310	1453	0
*Sadeh*27	906	9556	4108	1	3102	1278	1
*Sadeh*28	960	436237	190248	51	35850	15635	5
*Sadeh*29	804	23390	10494	3	6533	2818	1
*Sadeh*30	891	26735	11233	3	11311	4689	2
	Media	73069,30	31343,00	8,50	9335,60	3971,00	1,40
	Desviación	134215,79	58040,91	15,59	10808,93	4652,67	1,69

Tabla 5.26: Resultados con h_2 para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*31	1004	**4161601**	**1991787**	**458**	250357	111745	50
*Sadeh*32	1048	501492	236808	55	14980	5852	2
*Sadeh*33	961	447679	218832	51	92737	42990	15
*Sadeh*34	884	12156	6053	2	3511	1539	1
*Sadeh*35	972	2643002	1101393	294	136496	54343	23
*Sadeh*36	956	109687	54158	12	15205	7130	2
*Sadeh*37	976	103833	56779	11	5216	2584	1
*Sadeh*38	986	107857	49066	12	28524	12807	3
*Sadeh*39	1006	147579	59894	16	5770	2126	0
*Sadeh*40	943	**3815559**	**1644848**	**417**	312843	126036	67
	Media	509160,63	222872,88	56,63	37804,88	16171,38	5,88
	Desviación	823087,70	341292,00	91,57	46286,71	19273,89	7,88

Tabla 5.27: Resultados con h_2 para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*41	912	310	166	0	226	117	0
*Sadeh*42	950	69007	29775	8	11686	5149	2
*Sadeh*43	921	252	115	0	252	115	0
*Sadeh*44	924	1380	574	0	1244	518	0
*Sadeh*45	878	949	429	0	794	353	0
*Sadeh*46	784	640	297	0	464	214	0
*Sadeh*47	932	20798	9832	3	1513	681	0
*Sadeh*48	1011	11349	5758	1	2012	925	1
*Sadeh*49	864	5120	2353	1	1488	643	0
*Sadeh*50	902	887	390	0	780	345	0
	Media	11069,20	4968,90	1,30	2045,90	906,00	0,30
	Desviación	20318,53	8807,98	2,41	3261,49	1436,01	0,64

Tabla 5.28: Resultados con h_2 para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*51	1038	4119688	1966261	443	105088	47120	16
*Sadeh*52	1075	56914	28290	6	2055	873	0
*Sadeh*53	979	425338	212833	47	58846	28511	9
*Sadeh*54	950	16709	8546	2	4744	2268	1
*Sadeh*55	1026	49880	22084	6	5603	2344	1
*Sadeh*56	1058	19265	9538	2	4447	2062	0
*Sadeh*57	1039	147484	76745	17	19089	9215	3
*Sadeh*58	1034	23606	11100	2	5035	2222	1
*Sadeh*59	1032	23801	10502	3	2932	1194	0
*Sadeh*60	985	**3789546**	**1618829**	**402**	100649	44210	15
	Media	542520,56	260655,44	58,67	23093,22	10645,44	3,44
	Desviación	1270740,71	606225,50	136,57	33636,79	15348,45	5,19

Tabla 5.29: Resultados con h_2 para los problemas: $FT06$ y $LA01 - 05$

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$FT06$	265	184	111	0	165	100	0
$LA01$	4832	1118136	475522	129	248935	107955	35
$LA02$	4459	**3625492**	**1454493**	**411**	520356	217751	80
$LA03$	4151	392860	154791	47	79883	32118	10
$LA04$	4259	855500	345005	96	136928	57558	19
$LA05$	4072	**3570165**	**1399726**	**406**	412277	174320	68

Tabla 5.30: Cotas Inferiores (LB) y Máximas profundidades (MP) alcanzadas con h_2 para los problemas no resueltos

	No poda		Poda			No poda		Poda	
Instancia	LB	MP	LB	MP	Instancia	LB	MP	LB	MP
$LA06$	8070	23	8136	25	$ABZ7$	9710	43	9707	43
$LA07$	7726	24	7802	27	$ABZ8$	9830	38	9841	39
$LA08$	7082	22	7252	26	$ABZ9$	9524	44	9536	45
$LA09$	8112	23	8218	26	$FT10$	7102	47	7278	61
$LA10$	8400	27	8448	29	$FT20$	11913	26	12069	30
$LA11$	12699	17	12777	19	$LA21$	11121	45	11213	53
$LA12$	10657	20	10710	24	$LA24$	10440	45	10507	47
$LA13$	11999	18	12052	21	$LA25$	10191	36	10309	45
$LA14$	13521	18	13589	22	$LA27$	16237	27	16281	31
$LA15$	12780	18	12880	21	$LA29$	14839	31	14872	35
$LA16$	7097	59	7295	75	$LA38$	13562	69	13611	72
$LA17$	6356	62	6492	76	$LA40$	14287	67	14322	68
$LA18$	6795	59	6951	84	$YN1$	13215	51	13211	49
$LA19$	6993	62	7086	69	$YN2$	13950	41	13954	44
$LA20$	7082	58	7202	65	$YN3$	13497	62	13519	67
$ABZ5$	10151	60	10312	69	$YN4$	14575	37	14581	38
$ABZ6$	7589	61	7735	75	$ABZ8 - 10 \times 10$	3373	60	3448	72
$ORB01$	7857	59	7962	79	$LA21 - 10 \times 10$	6911	64	7052	84
$ORB02$	6975	59	7076	66	$LA24 - 10 \times 10$	6866	61	6988	69
$ORB03$	7560	46	7724	56	$LA25 - 10 \times 10$	6593	53	6767	69
$ORB04$	7732	66	7827	77	$LA27 - 10 \times 10$	6730	63	6855	73
$ORB05$	6759	58	6878	74	$LA29 - 10 \times 10$	6446	62	6652	80
$ORB06$	7679	50	7843	61	$LA40 - 10 \times 10$	6086	58	6207	68
$ORB07$	3190	53	3252	80	$YN1 - 10 \times 10$	3745	68	3839	86
$ORB08$	6540	35	6748	63	$YN2 - 10 \times 10$	3837	60	3926	74
$ORB09$	7226	61	7299	69	$YN3 - 10 \times 10$	3823	63	3871	70
$ORB10$	7596	55	7736	64	$YN4 - 10 \times 10$	4030	65	4091	71

5.8.2. Heurístico h_3

Tabla 5.31: Resultados con h_3 para una serie de problemas de tamaño 6×6: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 6 \times 6$	2656	508	296	0	432	255	0
$LA17 - 6 \times 6$	2392	3495	1902	0	1380	742	0
$LA18 - 6 \times 6$	2393	1101	673	0	609	373	0
$LA19 - 6 \times 6$	2617	730	450	0	408	257	0
$LA20 - 6 \times 6$	2374	483	309	0	392	250	0
$ORB01 - 6 \times 6$	2574	1362	824	0	1019	614	0
$ORB02 - 6 \times 6$	2778	1777	1079	0	1124	667	0
$ORB03 - 6 \times 6$	2759	591	352	0	377	219	0
$ORB04 - 6 \times 6$	2894	1304	757	0	747	431	0
$ORB05 - 6 \times 6$	2236	503	301	0	390	237	0
$ORB06 - 6 \times 6$	2730	2096	1210	1	1288	739	0
$ORB07 - 6 \times 6$	1215	946	551	0	860	499	0
$ORB08 - 6 \times 6$	2355	3961	2235	0	1591	883	0
$ORB09 - 6 \times 6$	2312	383	239	0	328	203	0
$ORB10 - 6 \times 6$	2889	624	391	0	431	273	0
$ABZ5 - 6 \times 6$	3767	656	376	0	489	280	0
$ABZ6 - 6 \times 6$	2971	368	223	0	263	162	0
$ABZ7 - 6 \times 6$	1109	315	193	0	306	188	0
$ABZ8 - 6 \times 6$	1365	703	427	1	429	266	0
$ABZ9 - 6 \times 6$	1164	249	166	0	249	166	0
$FT10 - 6 \times 6$	2544	1352	763	0	503	293	0
$LA21 - 6 \times 6$	2613	495	300	0	392	241	0
$LA24 - 6 \times 6$	2514	729	452	0	555	339	0
$LA25 - 6 \times 6$	2598	2116	1227	0	1292	757	1
$LA27 - 6 \times 6$	2416	1670	1008	0	1085	653	0
$LA29 - 6 \times 6$	2353	313	195	0	232	144	0
$LA38 - 6 \times 6$	2602	775	454	0	601	359	0
$LA40 - 6 \times 6$	2211	1091	666	0	641	394	0
$YN1 - 6 \times 6$	1418	415	230	1	352	196	0
$YN2 - 6 \times 6$	1545	2903	1771	0	1123	650	0
$YN3 - 6 \times 6$	1436	2039	1202	0	1293	773	0
$YN4 - 6 \times 6$	1382	2093	1302	0	1307	808	0
Media		1192,06	703,88	0,09	702,75	415,97	0,03
Desviación		934,19	532,07	0,29	401,40	227,86	0,17

Tabla 5.32: Resultados con h_3 para una serie de problemas de tamaño 7×7: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16-7\times7$	3554	2929	1722	1	1570	920	1
$LA17-7\times7$	3488	30418	17500	3	7274	4065	1
$LA18-7\times7$	3496	24596	14317	4	5880	3452	1
$LA19-7\times7$	3627	5548	3223	0	2174	1269	0
$LA20-7\times7$	3498	11646	7068	2	5011	3068	1
$ORB01-7\times7$	3589	11522	6737	2	5674	3324	1
$ORB02-7\times7$	3727	16362	10261	2	7219	4458	1
$ORB03-7\times7$	3754	49834	28470	6	11438	6411	2
$ORB04-7\times7$	3803	13614	8068	2	4975	2906	1
$ORB05-7\times7$	2939	7054	4314	1	4193	2538	0
$ORB06-7\times7$	3480	23094	13572	3	7105	4102	2
$ORB07-7\times7$	1747	79099	45966	10	22653	13070	3
$ORB08-7\times7$	3647	414355	230721	49	36445	19877	5
$ORB09-7\times7$	3428	6393	3927	0	3064	1795	1
$ORB10-7\times7$	3624	1681	1057	1	1264	797	0
$ABZ5-7\times7$	5144	4300	2535	0	1687	956	0
$ABZ6-7\times7$	3936	4193	2613	1	2687	1673	0
$ABZ7-7\times7$	1764	10213	5803	1	5022	2868	1
$ABZ8-7\times7$	1886	3453	1930	1	2327	1314	0
$ABZ9-7\times7$	1577	1238	766	0	1106	693	1
$FT10-7\times7$	3420	13924	8055	1	4135	2390	0
$LA21-7\times7$	3499	8642	5172	1	3608	2208	1
$LA24-7\times7$	3442	7133	4190	0	3190	1879	0
$LA25-7\times7$	3581	8840	5048	2	4448	2566	1
$LA27-7\times7$	3219	4779	3055	0	2449	1549	0
$LA29-7\times7$	3383	13513	8070	2	3778	2292	1
$LA38-7\times7$	3407	2346	1423	0	1565	955	0
$LA40-7\times7$	3128	10140	6419	1	2679	1644	0
$YN1-7\times7$	1947	12956	7786	2	4821	2852	1
$YN2-7\times7$	1988	18999	11293	2	5775	3391	1
$YN3-7\times7$	1883	9275	5603	2	4320	2626	1
$YN4-7\times7$	1883	5135	2969	0	2745	1579	0
	Media	26163,25	14989,16	3,19	5696,28	3296,47	0,88
	Desviación	71359,53	39728,62	8,46	6746,49	3713,26	1,02

Tabla 5.33: Resultados con h_3 para una serie de problemas de tamaño 8×8: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16-8\times8$	4600	88015	51698	14	23022	13556	4
$LA17-8\times8$	4366	376164	218430	55	38927	22668	7
$LA18-8\times8$	4690	97274	56958	15	15581	9233	3
$LA19-8\times8$	4612	84840	50139	13	19771	11767	4
$LA20-8\times8$	4616	130229	75048	20	28847	17044	5
$ORB01-8\times8$	4743	100786	55948	16	25001	13483	5
$ORB02-8\times8$	4678	74474	44370	12	29779	17722	5
$ORB03-8\times8$	4925	609313	351614	96	56856	31154	11
$ORB04-8\times8$	5081	77506	43955	13	21142	11869	4
$ORB05-8\times8$	4191	62741	37278	9	17264	10314	3
$ORB06-8\times8$	4673	390455	219056	61	63412	35233	12
$ORB07-8\times8$	2124	230257	132553	37	47697	27403	9
$ORB08-8\times8$	4749	**3723864**	**2019063**	**579**	226722	121036	44
$ORB09-8\times8$	4590	669276	419292	100	118292	73376	22
$ORB10-8\times8$	4959	5861	3602	1	3670	2245	0
$ABZ5-8\times8$	6818	84938	47264	13	18886	10383	3
$ABZ6-8\times8$	4900	99311	61345	15	22569	13805	4
$ABZ7-8\times8$	2257	127303	69520	20	30206	16928	6
$ABZ8-8\times8$	2496	114930	63026	17	31436	17465	5
$ABZ9-8\times8$	2213	40049	23490	6	17235	10114	3
$FT10-8\times8$	4559	153035	91764	24	25988	14928	5
$LA21-8\times8$	4459	140384	81464	21	29942	17697	5
$LA24-8\times8$	4641	242382	137244	37	44354	25478	8
$LA25-8\times8$	4750	232100	129933	35	52652	29982	9
$LA27-8\times8$	4381	45479	28121	7	14340	8739	2
$LA29-8\times8$	4540	95462	53596	14	18960	10898	4
$LA38-8\times8$	4439	71846	41178	11	31032	17992	5
$LA40-8\times8$	4090	87725	51687	13	29901	18030	6
$YN1-8\times8$	2480	64405	38933	9	17325	10393	3
$YN2-8\times8$	2510	33024	19501	5	11089	6563	2
$YN3-8\times8$	2459	67560	40275	10	23658	14092	4
$YN4-8\times8$	2618	194258	113441	31	30229	17676	6
	Media	157786,52	91991,06	24,19	30937,52	18007,42	5,61
	Desviación	154172,15	92180,82	23,60	20733,36	12491,24	3,93

Tabla 5.34: Resultados con h_3 para una serie de problemas de tamaño 9×9: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 9 \times 9$	5724	1822934	1036167	327	188197	109147	41
$LA17 - 9 \times 9$	5390	**3118369**	**1762209**	**531**	506692	289372	113
$LA18 - 9 \times 9$	5770	1213053	684244	213	164111	94832	34
$LA19 - 9 \times 9$	5891	1441043	867147	248	138246	83485	29
$LA20 - 9 \times 9$	5915	**3206393**	**1850419**	**546**	520616	305422	116
$ORB01 - 9 \times 9$	6367	**3057318**	**1701160**	**580**	746135	421573	179
$ORB02 - 9 \times 9$	5867	**3028468**	**1672121**	**527**	441510	253151	97
$ORB03 - 9 \times 9$	6310	**2796950**	**1440675**	**541**	491587	259095	118
$ORB04 - 9 \times 9$	6661	**3317541**	**1961483**	**623**	1131313	659567	289
$ORB05 - 9 \times 9$	5605	300471	176815	52	82206	47972	17
$ORB06 - 9 \times 9$	6106	**2832350**	**1475977**	**541**	860798	467269	221
$ORB07 - 9 \times 9$	2668	**3241386**	**1885379**	**602**	592898	345889	162
$ORB08 - 9 \times 9$	**5656**	**2547424**	**1191201**	**506**	1763736	908184	803
$ORB09 - 9 \times 9$	6013	925732	565944	170	187179	111023	41
$ORB10 - 9 \times 9$	6328	**3066207**	**1710267**	**550**	487075	276921	112
$ABZ5 - 9 \times 9$	8586	1597384	934936	276	199814	114826	42
$ABZ6 - 9 \times 9$	6524	868473	509495	155	145549	85371	31
$ABZ7 - 9 \times 9$	2767	2243067	1268826	388	329414	190634	75
$ABZ8 - 9 \times 9$	3078	**3205826**	**1849749**	**563**	926678	534344	296
$ABZ9 - 9 \times 9$	2752	848491	476780	150	134297	75710	29
$FT10 - 9 \times 9$	5982	**3164000**	**1807790**	**606**	328673	183711	77
$LA21 - 9 \times 9$	5428	1734782	1036484	296	153642	92732	32
$LA24 - 9 \times 9$	5650	2221842	1322528	386	237697	142193	51
$LA25 - 9 \times 9$	5880	**3045877**	**1689949**	**536**	328911	186607	71
$LA27 - 9 \times 9$	5608	1983058	1150542	338	190361	109803	41
$LA29 - 9 \times 9$	5732	**3208280**	**1852333**	**555**	376393	220531	81
$LA38 - 9 \times 9$	5855	3122647	1779325	525	452798	261603	100
$LA40 - 9 \times 9$	5006	2620931	1445671	446	297212	172114	65
$YN1 - 9 \times 9$	3187	**3264492**	**1908612**	**548**	201514	118216	43
$YN2 - 9 \times 9$	3198	1789183	1024328	308	295414	171157	65
$YN3 - 9 \times 9$	3206	2116435	1238475	360	370811	217455	85
$YN4 - 9 \times 9$	3257	**3180911**	**1824965**	**555**	395649	229238	93
	Media	1678095,38	969856,69	289,88	222934,25	130003,56	48,63
	Desviación	707053,82	399410,95	118,34	96887,53	55985,82	22,35

Tabla 5.35: Resultados con h_3 para los únicos problema resueltos de tamaño 10×10: problemas recortados de las batería *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$ABZ7 - 10 \times 10$	3262	**2638952**	**1538017**	**566**	551879,00	326732,00	169,00
$ABZ9 - 10 \times 10$	3483	**2421717**	**1320556**	**532**	1049546,00	581117,00	402,00
$LA38 - 10 \times 10$	7113	**2461209**	**1360193**	**546**	934964,00	534999,00	279,00

Tabla 5.36: Resultados con h_3 para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*01	857	6435	2752	0	2906	1239	1
*Sadeh*02	850	14109	6092	2	4614	1937	1
*Sadeh*03	911	44294	19750	6	8632	3506	1
*Sadeh*04	832	9958	4566	1	4774	2137	1
*Sadeh*05	813	4567	2060	1	1326	557	0
*Sadeh*06	755	58066	24840	7	8973	3790	1
*Sadeh*07	883	23360	9512	3	3700	1432	1
*Sadeh*08	868	53131	25668	6	6843	3132	1
*Sadeh*09	766	104843	49242	12	5168	2240	0
*Sadeh*10	863	262745	121780	33	12020	5131	2
Media		58150,80	26626,20	7,10	5895,60	2510,10	0,90
Desviación		74346,85	34629,16	9,30	3056,69	1304,57	0,54

Tabla 5.37: Resultados con h_3 para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*11	988	**3986148**	**1815645**	**471**	263764	111824	60
*Sadeh*12	997	673867	326101	78	13541	5719	2
*Sadeh*13	949	1517508	743081	178	186002	85963	37
*Sadeh*14	867	119543	51764	15	20593	8143	3
*Sadeh*15	853	753470	327598	92	54406	23144	8
*Sadeh*16	866	2681044	1368293	328	43518	18367	7
*Sadeh*17	945	294008	157032	34	23326	11182	4
*Sadeh*18	914	241140	115807	28	43389	19516	6
*Sadeh*19	980	145120	64679	17	12426	5304	2
*Sadeh*20	879	**3808911**	**1636885**	**449**	157661	65207	29
Media		803212,50	394294,38	96,25	49650,13	22167,25	8,63
Desviación		831518,06	424153,01	101,34	53523,40	24911,32	10,93

Tabla 5.38: Resultados con h_3 para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*21	869	6692	2846	0	3984	1697	0
*Sadeh*22	913	203738	83654	26	23041	9509	4
*Sadeh*23	911	4550	2012	0	2715	1158	0
*Sadeh*24	871	6602	2963	1	1851	811	1
*Sadeh*25	837	2376	991	0	1359	559	0
*Sadeh*26	774	9607	4456	2	3214	1414	0
*Sadeh*27	906	9489	4076	1	3037	1247	1
*Sadeh*28	960	435939	190071	55	35894	15642	6
*Sadeh*29	804	23475	10550	2	6543	2829	1
	Media	72872,10	31266,50	9,10	9298,10	3956,50	1,40
	Desviación	134158,82	58000,63	17,03	10836,35	4662,44	1,91

Tabla 5.39: Resultados con h_3 para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*31	1004	**4162554**	**1992740**	**480**	250072	111641	54
*Sadeh*32	1048	501492	236808	59	14980	5852	2
*Sadeh*33	961	446725	218424	53	92641	42956	16
*Sadeh*34	884	11954	5947	1	3467	1519	1
*Sadeh*35	972	2642182	1101090	316	136371	54305	24
*Sadeh*36	956	108591	53641	13	15063	7066	2
*Sadeh*37	976	103683	56721	11	5191	2574	1
*Sadeh*38	986	107730	48982	14	28444	12755	4
*Sadeh*39	1006	147579	59894	18	5770	2126	1
*Sadeh*40	943	**3816617**	**1645906**	**448**	311946	125795	70
	Media	508742,00	222688,38	60,63	37740,88	16144,13	6,38
	Desviación	822929,69	341244,39	98,46	46256,12	19266,25	8,17

Tabla 5.40: Resultados con h_3 para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*41	912	310	166	0	226	117	0
*Sadeh*42	950	68795	29698	8	11571	5104	2
*Sadeh*43	921	252	115	0	252	115	0
*Sadeh*44	924	1376	572	0	1240	516	0
*Sadeh*45	878	949	429	0	794	353	1
*Sadeh*46	784	608	283	0	430	199	0
*Sadeh*47	932	20526	9708	2	1506	678	0
*Sadeh*48	1011	11349	5758	2	2012	925	0
*Sadeh*49	864	5033	2316	0	1478	640	0
*Sadeh*50	902	873	383	0	766	338	0
	Media	11007,10	4942,80	1,20	2027,50	898,50	0,30
	Desviación	20250,24	8781,80	2,40	3230,10	1423,88	0,64

Tabla 5.41: Resultados con h_3 para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*51	1038	4118798	1965912	469	104966	47091	17
*Sadeh*52	1075	56914	28290	7	2051	871	0
*Sadeh*53	979	424840	212602	49	58753	28471	9
*Sadeh*54	950	16456	8419	2	4659	2227	1
*Sadeh*55	1026	49880	22084	6	5603	2344	1
*Sadeh*56	1058	19157	9495	3	4390	2040	0
*Sadeh*57	1039	146089	76119	17	19064	9221	3
*Sadeh*58	1034	23606	11100	3	5037	2223	1
*Sadeh*59	1032	23797	10500	2	2932	1194	0
*Sadeh*60	985	**3790067**	**1619340**	**423**	100554	44189	16
	Media	542170,78	260502,33	62,00	23050,56	10631,33	3,56
	Desviación	1270532,44	606147,49	144,59	35640,23	16269,99	5,79

Tabla 5.42: Resultados con h_3 para los problemas: $FT06$ y $LA01 - 05$

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$FT06$	265	184	111	0	165	100	0
$LA01$	4832	1109048	471492	138	246576	106896	37
$LA02$	4459	**3622825**	**1451824**	**444**	516490	215975	84
$LA03$	4151	390090	153589	48	79239	31832	12
$LA04$	4259	844107	339748	102	135262	56746	20
$LA05$	4072	**3569995**	**1399554**	**435**	410076	173443	70

Tabla 5.43: Cotas Inferiores (LB) y Máximas profundidades (MP) alcanzadas con h_3 para los problemas no resueltos

	No poda		Poda			No poda		Poda	
Instancia	LB	MP	LB	MP	Instancia	LB	MP	LB	MP
$LA06$	8071	23	8137	25	$ABZ7$	9710	43	9707	43
$LA07$	7727	24	7803	27	$ABZ8$	9830	38	9841	39
$LA08$	7083	22	7253	26	$ABZ9$	9524	44	9536	45
$LA09$	8113	23	8221	26	$FT10$	7102	47	7278	61
$LA10$	8401	27	8449	29	$FT20$	11913	26	12071	30
$LA11$	12700	17	12780	19	$LA21$	11122	45	11213	53
$LA12$	10659	20	10713	24	$LA24$	10440	45	10508	47
$LA13$	11999	18	12052	21	$LA25$	10191	36	10309	45
$LA14$	13522	18	13591	22	$LA27$	16237	27	16281	31
$LA15$	12782	19	12885	21	$LA29$	14839	31	14873	35
$LA16$	7098	59	7295	75	$LA38$	13563	69	13612	72
$LA17$	6356	62	6493	76	$LA40$	14287	67	14322	68
$LA18$	6795	59	6951	84	$YN1$	13215	51	13211	49
$LA19$	6993	62	7086	69	$YN2$	13950	41	13954	44
$LA20$	7082	58	7202	65	$YN3$	13497	62	13519	67
$ABZ5$	10151	60	10310	69	$YN4$	14575	37	14582	38
$ABZ6$	7589	61	7735	75	$ABZ8 - 10 \times 10$	3373	60	3448	72
$ORB01$	7857	59	7962	79	$LA21 - 10 \times 10$	6911	64	7053	84
$ORB02$	6975	59	7076	66	$LA24 - 10 \times 10$	6866	61	6988	69
$ORB03$	7560	46	7724	56	$LA25 - 10 \times 10$	6593	53	6767	69
$ORB04$	7732	66	7827	77	$LA27 - 10 \times 10$	6731	63	6855	73
$ORB05$	6759	58	6878	74	$LA29 - 10 \times 10$	6447	62	6652	80
$ORB06$	7679	50	7843	61	$LA40 - 10 \times 10$	6087	58	6207	68
$ORB07$	3190	53	3252	80	$YN1 - 10 \times 10$	3745	68	3839	86
$ORB08$	6540	35	6748	63	$YN2 - 10 \times 10$	3837	60	3926	74
$ORB09$	7226	61	7299	69	$YN3 - 10 \times 10$	3823	63	3871	70
$ORB10$	7597	55	7737	64	$YN4 - 10 \times 10$	4030	65	4091	71

5.8.3. Heurístico h_4

Tabla 5.44: Resultados con h_4 para una serie de problemas de tamaño 6×6: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*FT*06 *LA*16 – 6×6	2656	508	296	0	432	255	1
*LA*17 – 6×6	2392	3495	1902	1	1380	742	0
*LA*18 – 6×6	2393	1101	673	0	609	373	0
*LA*19 – 6×6	2617	730	450	0	408	257	0
*LA*20 – 6×6	2374	483	309	0	392	250	0
*ORB*01 – 6×6	2574	1362	824	0	1019	614	1
*ORB*02 – 6×6	2778	1777	1079	0	1124	667	0
*ORB*03 – 6×6	2759	591	352	1	377	219	0
*ORB*04 – 6×6	2894	1304	757	0	747	431	0
*ORB*05 – 6×6	2236	503	301	0	390	237	0
*ORB*06 – 6×6	2730	2094	1209	0	1288	739	0
*ORB*07 – 6×6	1215	946	551	0	860	499	0
*ORB*08 – 6×6	2355	3961	2235	1	1591	883	1
*ORB*09 – 6×6	2312	383	239	0	328	203	0
*ORB*10 – 6×6	2889	624	391	0	431	273	0
*ABZ*5 – 6×6	3767	656	376	0	489	280	0
*ABZ*6 – 6×6	2971	368	223	0	263	162	0
*ABZ*7 – 6×6	1109	315	193	0	306	188	0
*ABZ*8 – 6×6	1365	703	427	0	429	266	0
*ABZ*9 – 6×6	1164	249	166	1	249	166	0
*FT*10 – 6×6	2544	1352	763	0	503	293	0
*LA*21 – 6×6	2613	495	300	0	392	241	0
*LA*24 – 6×6	2514	729	452	0	555	339	0
*LA*25 – 6×6	2598	2116	1227	0	1292	757	1
*LA*27 – 6×6	2416	1670	1008	0	1085	653	0
*LA*29 – 6×6	2353	313	195	0	232	144	0
*LA*38 – 6×6	2602	775	454	1	601	359	0
*LA*40 – 6×6	2211	1091	666	0	641	394	0
*YN*1 – 6×6	1418	415	230	0	352	196	0
*YN*2 – 6×6	1545	2903	1771	0	1123	650	0
*YN*3 – 6×6	1436	2039	1202	0	1293	773	0
*YN*4 – 6×6	1382	2093	1302	1	1307	808	1
Media		1192,00	703,84	0,19	702,75	415,97	0,16
Desviación		934,13	532,04	0,39	401,40	227,86	0,36

Tabla 5.45: Resultados con h_4 para una serie de problemas de tamaño 7×7: problemas recortados de las baterías LA, ORB, ABZ, YN y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 7 \times 7$	3554	2929	1722	0	1570	920	0
$LA17 - 7 \times 7$	3488	30404	17492	5	7277	4067	1
$LA18 - 7 \times 7$	3496	24592	14315	5	5878	3451	2
$LA19 - 7 \times 7$	3627	5537	3216	1	2168	1265	0
$LA20 - 7 \times 7$	3498	11646	7068	1	5011	3068	1
$ORB01 - 7 \times 7$	3589	11522	6737	2	5674	3324	1
$ORB02 - 7 \times 7$	3727	16362	10261	3	7219	4458	1
$ORB03 - 7 \times 7$	3754	49834	28470	9	11438	6411	3
$ORB04 - 7 \times 7$	3803	13602	8061	3	4971	2903	1
$ORB05 - 7 \times 7$	2939	7054	4314	1	4193	2538	1
$ORB06 - 7 \times 7$	3480	23094	13572	4	7106	4103	1
$ORB07 - 7 \times 7$	1747	79043	45934	14	22645	13065	5
$ORB08 - 7 \times 7$	3647	414127	230608	75	36418	19864	8
$ORB09 - 7 \times 7$	3428	6393	3927	1	3064	1795	1
$ORB10 - 7 \times 7$	3624	1681	1057	0	1264	797	0
$ABZ5 - 7 \times 7$	5144	4300	2535	1	1687	956	0
$ABZ6 - 7 \times 7$	3936	4193	2613	1	2687	1673	1
$ABZ7 - 7 \times 7$	1764	10213	5803	2	5022	2868	1
$ABZ8 - 7 \times 7$	1886	3450	1928	0	2324	1312	0
$ABZ9 - 7 \times 7$	1577	1238	766	1	1106	693	1
$FT10 - 7 \times 7$	3420	13924	8055	2	4135	2390	0
$LA21 - 7 \times 7$	3499	8642	5172	2	3608	2208	1
$LA24 - 7 \times 7$	3442	7133	4190	1	3190	1879	1
$LA25 - 7 \times 7$	3581	8818	5037	1	4448	2566	1
$LA27 - 7 \times 7$	3219	4779	3055	1	2449	1549	0
$LA29 - 7 \times 7$	3383	13513	8070	2	3778	2292	1
$LA38 - 7 \times 7$	3407	2346	1423	1	1565	955	0
$LA40 - 7 \times 7$	3128	10140	6419	1	2679	1644	1
$YN1 - 7 \times 7$	1947	12956	7786	3	4821	2852	1
$YN2 - 7 \times 7$	1988	18999	11293	3	5775	3391	1
$YN3 - 7 \times 7$	1883	9275	5603	2	4320	2626	1
$YN4 - 7 \times 7$	1883	5135	2969	1	2755	1584	0
Media		26152,31	14983,47	4,66	5695,16	3295,84	1,16
Desviación		71319,81	39708,86	12,93	6742,06	3711,09	1,56

Tabla 5.46: Resultados con h_4 para una serie de problemas de tamaño 8×8: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16-8\times8$	4600	87913	51643	20	23017	13552	6
$LA17-8\times8$	4366	375893	218285	77	38915	22659	10
$LA18-8\times8$	4690	97252	56944	21	15576	9230	4
$LA19-8\times8$	4612	84838	50138	17	19769	11766	4
$LA20-8\times8$	4616	130229	75048	26	28847	17044	6
$ORB01-8\times8$	4743	100765	55938	25	24990	13478	7
$ORB02-8\times8$	4678	74454	44359	16	29769	17717	8
$ORB03-8\times8$	4925	609311	351613	148	56856	31154	16
$ORB04-8\times8$	5081	77434	43922	20	21130	11863	6
$ORB05-8\times8$	4191	62687	37243	14	17241	10298	4
$ORB06-8\times8$	4673	390407	219030	97	63418	35236	18
$ORB07-8\times8$	2124	230111	132472	56	47649	27373	13
$ORB08-8\times8$	4749	**3725252**	**2020037**	**913**	226793	121065	66
$ORB09-8\times8$	4590	668994	419119	142	118268	73359	31
$ORB10-8\times8$	4959	5848	3595	2	3659	2239	1
$ABZ5-8\times8$	6818	84938	47264	20	18886	10383	5
$ABZ6-8\times8$	4900	99306	61342	21	22564	13802	6
$ABZ7-8\times8$	2257	127200	69466	29	30189	16920	8
$ABZ8-8\times8$	2496	114853	62987	26	31416	17453	8
$ABZ9-8\times8$	2213	40044	23488	9	17230	10112	4
$FT10-8\times8$	4559	152987	91739	36	25976	14923	7
$LA21-8\times8$	4459	140384	81464	29	29942	17697	7
$LA24-8\times8$	4641	242278	137185	50	44342	25469	11
$LA25-8\times8$	4750	231770	129753	53	52610	29956	13
$LA27-8\times8$	4381	45479	28121	9	14340	8739	3
$LA29-8\times8$	4540	95462	53596	20	18960	10898	5
$LA38-8\times8$	4439	71841	41176	16	31032	17992	7
$LA40-8\times8$	4090	87691	51673	18	29899	18029	7
$YN1-8\times8$	2480	64405	38933	14	17325	10393	5
$YN2-8\times8$	2510	33020	19500	7	11085	6562	3
$YN3-8\times8$	2459	67548	40268	15	23646	14085	6
$YN4-8\times8$	2618	194037	113316	46	30173	17645	8
	Media	157721,90	91955,48	35,45	30926,42	18000,84	7,97
	Desviación	154123,80	92151,94	35,20	20729,40	12488,26	5,60

Tabla 5.47: Resultados con h_4 para una serie de problemas de tamaño 9×9: problemas recortados de las baterías *LA*, *ORB*, *ABZ*, *YN* y *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$LA16 - 9 \times 9$	5724	1821768	1035580	513	188185	109132	61
$LA17 - 9 \times 9$	5390	**3118321**	**1762201**	**841**	506771	289413	159
$LA18 - 9 \times 9$	5770	1212541	683956	328	164115	94838	50
$LA19 - 9 \times 9$	5891	1440383	866739	359	138240	83476	41
$LA20 - 9 \times 9$	5915	**3206033**	**1850202**	**789**	520575	305397	156
$ORB01 - 9 \times 9$	6367	**3058442**	**1702208**	**951**	746367	421696	268
$ORB02 - 9 \times 9$	5867	**3028747**	**1672335**	**801**	441574	253181	139
$ORB03 - 9 \times 9$	6310	**2797369**	**1441025**	**959**	491559	259069	188
$ORB04 - 9 \times 9$	6661	**3318551**	**1962449**	**1030**	1130244	658936	426
$ORB05 - 9 \times 9$	5605	300402	176780	81	82202	47969	25
$ORB06 - 9 \times 9$	6106	**2832372**	**1476003**	**907**	860617	467176	327
$ORB07 - 9 \times 9$	2668	**3241887**	**1885779**	**933**	592514	345663	226
$ORB08 - 9 \times 9$	**5656**	**2547901**	**1191545**	**962**	1764412	908487	1021
$ORB09 - 9 \times 9$	6013	924995	565535	265	187106	110984	70
$ORB10 - 9 \times 9$	6328	**3065627**	**1709825**	**879**	486952	276823	170
$ABZ5 - 9 \times 9$	8586	1597328	934908	418	199753	114801	61
$ABZ6 - 9 \times 9$	6524	868456	509487	234	145586	85393	45
$ABZ7 - 9 \times 9$	2767	2242921	1268745	605	329395	190619	109
$ABZ8 - 9 \times 9$	3078	**3206130**	**1849932**	**883**	926602	534295	381
$ABZ9 - 9 \times 9$	2752	848377	476721	228	134290	75707	43
$FT10 - 9 \times 9$	5982	**3164321**	**1807970**	**966**	328658	183701	115
$LA21 - 9 \times 9$	5428	1734303	1036218	425	153623	92720	44
$LA24 - 9 \times 9$	5650	2213913	1317704	562	237752	142210	71
$LA25 - 9 \times 9$	5880	**3046613**	**1690640**	**845**	328962	186632	105
$LA27 - 9 \times 9$	5608	1982825	1150413	497	190427	109839	58
$LA29 - 9 \times 9$	5732	**3208154**	**1852269**	**816**	376439	220550	113
$LA38 - 9 \times 9$	5855	3121949	1778952	775	452713	261553	139
$LA40 - 9 \times 9$	5006	2620153	1445244	665	297176	172100	92
$YN1 - 9 \times 9$	3187	**3264428**	**1908526**	**797**	201505	118209	59
$YN2 - 9 \times 9$	3198	1788881	1024183	460	295385	171147	92
$YN3 - 9 \times 9$	3206	2115720	1238051	530	370941	217528	116
$YN4 - 9 \times 9$	3257	**3179432**	**1823431**	**869**	395188	228988	133
	Media	1677182,19	969326,00	434,06	222930,56	130001,00	69,81
	Desviación	706561,67	399086,68	174,67	96884,90	55984,58	30,59

Tabla 5.48: Resultados con h_4 para los únicos problema resueltos de tamaño 10×10: problemas recortados de las batería *Selectos*

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$ABZ7 - 10 \times 10$	3262	**2638953**	**1538023**	**913**	551935,00	326758,00	242,00
$ABZ9 - 10 \times 10$	3483	**2421749**	**1320576**	**881**	1049924,00	581325,00	554,00
$LA38 - 10 \times 10$	7113	**2461574**	**1360541**	**900**	934613,00	534787,00	406,00

Tabla 5.49: Resultados con h_4 para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*01	857	6435	2752	2	2906	1239	0
*Sadeh*02	850	8011	3224	2	4608	1935	2
*Sadeh*03	911	44294	19750	9	8632	3506	2
*Sadeh*04	832	9958	4566	2	4774	2137	1
*Sadeh*05	813	4567	2060	1	1326	557	0
*Sadeh*06	755	57854	24743	11	8973	3792	2
*Sadeh*07	883	23360	9512	5	3700	1432	1
*Sadeh*08	868	53131	25668	8	6843	3132	2
*Sadeh*09	766	104687	49178	16	5150	2232	1
*Sadeh*10	863	262745	121780	51	12020	5131	3
	Media	57504,20	26323,30	10,70	5893,20	2509,30	1,40
	Desviación	74719,62	34805,67	14,20	3057,38	1305,02	0,92

Tabla 5.50: Resultados con h_4 para los problemas de la batería *Sadeh* ($RG = 0{,}0$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*11	988	**3985922**	**1815546**	**728**	263769	111828	85
*Sadeh*12	997	673867	326101	117	13541	5719	3
*Sadeh*13	949	1515848	742407	277	185888	85928	52
*Sadeh*14	867	119543	51764	25	20593	8143	5
*Sadeh*15	853	752823	327285	148	54390	23139	13
*Sadeh*16	866	2679402	1367529	510	43487	18354	11
*Sadeh*17	945	294008	157032	48	23326	11182	6
*Sadeh*18	914	241118	115800	44	43375	19512	10
*Sadeh*19	980	145120	64679	25	12426	5304	3
*Sadeh*20	879	**3809283**	**1637280**	**699**	157757	65259	45
	Media	802716,13	394074,63	149,25	49628,25	22160,13	12,88
	Desviación	830883,04	423871,17	158,00	53487,57	24900,40	15,19

Tabla 5.51: Resultados con h_4 para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*21	869	6692	2846	2	3984	1697	1
*Sadeh*22	913	203625	83603	42	23032	9504	6
*Sadeh*23	911	4550	2012	1	2715	1158	1
*Sadeh*24	871	6602	2963	1	1851	811	0
*Sadeh*25	837	2376	991	1	1359	559	0
*Sadeh*26	774	9601	4453	2	3212	1413	1
*Sadeh*27	906	9489	4076	2	3037	1247	1
*Sadeh*28	960	435939	190071	95	35894	15642	10
*Sadeh*29	804	23463	10545	5	6531	2824	1
*Sadeh*30	891	26253	11046	6	11343	4699	3
	Media	72859,00	31260,60	15,70	9295,80	3955,40	2,40
	Desviación	134148,53	57996,34	28,98	10835,63	4662,02	3,04

Tabla 5.52: Resultados con h_4 para los problemas de la batería *Sadeh* ($RG = 0{,}1$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*31	1004	**4162517**	**1992767**	**731**	250052	111634	74
*Sadeh*32	1048	501492	236808	90	14980	5852	4
*Sadeh*33	961	446499	218340	82	92660	42972	23
*Sadeh*34	884	11954	5947	2	3467	1519	1
*Sadeh*35	972	2640852	1100668	523	136462	54348	39
*Sadeh*36	956	108591	53641	21	15063	7066	4
*Sadeh*37	976	103683	56721	15	5191	2574	1
*Sadeh*38	986	107730	48982	21	28444	12755	6
*Sadeh*39	1006	147579	59894	30	5770	2126	2
*Sadeh*40	943	**3816646**	**1645925**	**691**	311569	125634	100
	Media	508547,50	222625,13	98,00	37754,63	16151,50	10,00
	Desviación	822500,82	341108,74	163,38	46283,20	19279,68	12,86

Tabla 5.53: Resultados con h_4 para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 1$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*40	943	3816646	1645925	691	311569	125634	100
*Sadeh*41	912	310	166	0	226	117	0
*Sadeh*42	950	68795	29698	13	11571	5104	2
*Sadeh*43	921	252	115	0	252	115	0
*Sadeh*44	924	1376	572	0	1240	516	0
*Sadeh*45	878	949	429	0	794	353	0
*Sadeh*46	784	608	283	0	430	199	0
*Sadeh*47	932	20526	9708	4	1506	678	0
*Sadeh*48	1011	11349	5758	1	2012	925	0
*Sadeh*49	864	5033	2316	1	1478	640	0
*Sadeh*50	902	873	383	1	766	338	0
	Media	11007,10	4942,80	2,00	2027,50	898,50	0,20
	Desviación	20250,24	8781,80	3,85	3230,10	1423,88	0,60

Tabla 5.54: Resultados con h_4 para los problemas de la batería *Sadeh* ($RG = 0{,}2$, $BK = 2$)

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
*Sadeh*51	1038	4117350	1965266	684	105010	47115	26
*Sadeh*52	1075	56914	28290	10	2051	871	0
*Sadeh*53	979	424701	212536	71	58704	28450	13
*Sadeh*54	950	16456	8419	3	4659	2227	1
*Sadeh*55	1026	49880	22084	9	5603	2344	1
*Sadeh*56	1058	19157	9495	3	4390	2040	1
*Sadeh*57	1039	146089	76119	26	19064	9221	5
*Sadeh*58	1034	23606	11100	4	5037	2223	1
*Sadeh*59	1032	23797	10500	4	2932	1194	1
*Sadeh*60	985	**3790914**	**1619839**	**612**	100642	44213	23
	Media	541994,44	260423,22	90,44	23050,00	10631,67	5,44
	Desviación	1270080,96	605946,12	210,87	33608,07	15343,12	8,22

Tabla 5.55: Resultados con h_4 para los problemas: $FT06$ y $LA01 - 05$

		No poda			Poda		
Instancia	Sol.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)
$FT06$	265	184	111	0	165	100	0
$LA01$	4832	1105430	470037	227	246145	106711	58
$LA02$	4459	**3703408**	**1485722**	**725**	515840	215751	123
$LA03$	4151	389175	153266	79	79403	31886	19
$LA04$	4259	843025	339377	166	135074	56678	31
$LA05$	4072	**3570789**	**1400345**	**740**	408693	172827	102

Tabla 5.56: Cotas Inferiores (LB) y Máximas profundidades (MP) alcanzadas con h_4 para los problemas no resueltos

	No poda		Poda			No poda		Poda	
Instancia	LB	MP	LB	MP	Instancia	LB	MP	LB	MP
$LA06$	8072	23	8139	25	$ABZ7$	9711	42	9708	42
$LA07$	7728	24	7804	27	$ABZ8$	9833	39	9844	39
$LA08$	7085	22	7256	26	$ABZ9$	9527	44	9539	45
$LA09$	8117	23	8223	26	$FT10$	7102	47	7278	61
$LA10$	8406	27	8452	31	$FT20$	11924	26	12076	30
$LA11$	12702	17	12781	20	$LA21$	11121	45	11214	53
$LA12$	10662	20	10718	24	$LA24$	10445	45	10512	47
$LA13$	12016	18	12063	22	$LA25$	10185	36	10311	45
$LA14$	13527	18	13593	22	$LA27$	16249	27	16294	31
$LA15$	12786	19	12887	21	$LA29$	14872	33	14905	35
$LA16$	7098	59	7295	75	$LA38$	13564	69	13612	72
$LA17$	6356	62	6493	76	$LA40$	14287	67	14322	68
$LA18$	6795	59	6951	84	$YN1$	13215	51	13211	49
$LA19$	6993	62	7086	69	$YN2$	13961	43	13964	46
$LA20$	7082	58	7202	65	$YN3$	13498	62	13520	67
$ABZ5$	10151	60	10310	69	$YN4$	14575	37	14582	38
$ABZ6$	7589	61	7735	75	$ABZ8 - 10 \times 10$	3373	60	3448	72
$ORB01$	7857	59	7962	79	$LA21 - 10 \times 10$	6911	64	7052	84
$ORB02$	6975	59	7076	66	$LA24 - 10 \times 10$	6866	61	6989	69
$ORB03$	7561	46	7724	56	$LA25 - 10 \times 10$	6593	53	6767	69
$ORB04$	7732	66	7827	77	$LA27 - 10 \times 10$	6731	63	6855	73
$ORB05$	6760	58	6878	74	$LA29 - 10 \times 10$	6447	62	6652	80
$ORB06$	7679	50	7843	61	$LA40 - 10 \times 10$	6087	58	6208	68
$ORB07$	3190	53	3252	80	$YN1 - 10 \times 10$	3745	68	3839	86
$ORB08$	6540	35	6749	63	$YN2 - 10 \times 10$	3837	60	3926	74
$ORB09$	7226	61	7299	69	$YN3 - 10 \times 10$	3823	63	3871	70
$ORB10$	7597	55	7737	64	$YN4 - 10 \times 10$	4031	65	4092	71

5.8.4. Soluciones y Cotas Inferiores

En las dos tablas siguientes (Tablas 5.57 y 5.58) se muestra un resumen de los valores de las soluciones o de las mejores cotas inferiores obtenidas para los problemas utilizados en el estudio experimental. Los valores en negrita indican cotas inferiores. Para estos valores se indica si se han obtenido con o sin la técnica de poda (**P/NP**), así como el heurístico o heurísticos que han obtenido los valores (cuando se obtiene el mismo valor con todos ellos, no se pone nada).

La Tabla 5.57 se refiere a los problemas originales y la Tabla 5.58 a los problemas recortados.

Tabla 5.57: Problemas Baterías: Soluciones y Cotas inferiores (en negrita)

Tamaño	Inst.	Sol./LB	Tamaño	Inst.	Sol./LB
6×6	$FT06$	265	10×10	$ORB01$	**7962, P**
10×5	$LA01$	4832		$ORB02$	**7076, P**
	$LA02$	4459		$ORB03$	**7724, P**
	$LA03$	4151		$ORB04$	**7827, P**
	$LA04$	4259		$ORB05$	**6878, P**
	$LA05$	4072		$ORB06$	**7843, P**
15×5	$LA06$	**8139, P-h_4**		$ORB07$	**3252, P**
	$LA07$	**7804, P-h_4**		$ORB08$	**6749, P-h_4**
	$LA08$	**7256, P-h_4**		$ORB09$	**7299, P**
	$LA09$	**8223, P-h_4**		$ORB10$	**7737, P-h_3, h_4**
	$LA10$	**8452, P-h_4**	20×5	$FT10$	**7278, P**
20×5	$LA11$	**12781, P-h_4**	15×10	$FT20$	**12076, P-h_4**
	$LA12$	**10718, P-h_4**		$LA21$	**11214, P-h_4**
	$LA13$	**12063, P-h_4**		$LA24$	**10512, P-h_4**
	$LA14$	**13593, P-h_4**	20×10	$LA25$	**10311, P-h_4**
	$LA15$	**12887, P-h_4**		$LA27$	**16294, P-h_4**
10×10	$LA16$	**7295, P**	15×15	$LA29$	**14905, P-h_4**
	$LA17$	**6493, P-h_3, h_4**		$LA38$	**13612, P-h_3, h_4**
	$LA18$	**6951, P**	20×15	$LA40$	**14322, P**
	$LA19$	**7086, P**		$ABZ7$	**9711, NP-h_4**
	$LA20$	**7202, P**		$ABZ8$	**9844, P-h_4**
	$ABZ5$	**10312, P-h_2**	20×20	$ABZ9$	**9539, P-h_4**
	$ABZ6$	**7735, P**		$YN1$	**13215, NP**
				$YN2$	**13964, P-h_4**
				$YN3$	**13520, P-h_4**
				$YN4$	**14582, P-h_3, h_4**

Tabla 5.58: Problemas Recortados: Soluciones y Cotas inferiores (en negrita)

Inst.	Sol.	Ins.	Sol.	Inst.	Sol.	Inst.	Sol.
$LA16-6\times6$	2656	$LA20-7\times7$	3498	$ORB04-8\times8$	5081	$ORB08-9\times9$	5668
$LA17-6\times6$	2392	$ORB01-7\times7$	3589	$ORB05-8\times8$	4191	$ORB09-9\times9$	6013
$LA18-6\times6$	2393	$ORB02-7\times7$	3727	$ORB06-8\times8$	4673	$ORB10-9\times9$	6328
$LA19-6\times6$	2617	$ORB03-7\times7$	3754	$ORB07-8\times8$	2124	$ABZ5-9\times9$	8586
$LA20-6\times6$	2374	$ORB04-7\times7$	3803	$ORB08-8\times8$	4749	$ABZ6-9\times9$	6524
$ORB01-6\times6$	2574	$ORB05-7\times7$	2939	$ORB09-8\times8$	4590	$ABZ7-9\times9$	2767
$ORB02-6\times6$	2778	$ORB06-7\times7$	3480	$ORB10-8\times8$	4959	$ABZ8-9\times9$	3078
$ORB03-6\times6$	2759	$ORB07-7\times7$	1747	$ABZ5-8\times8$	6818	$ABZ9-9\times9$	2752
$ORB04-6\times6$	2894	$ORB08-7\times7$	3647	$ABZ6-8\times8$	4900	$FT10-9\times9$	5982
$ORB05-6\times6$	2236	$ORB09-7\times7$	3428	$ABZ7-8\times8$	2257	$LA21-9\times9$	5428
$ORB06-6\times6$	2730	$ORB10-7\times7$	3624	$ABZ8-8\times8$	2496	$LA24-9\times9$	5650
$ORB07-6\times6$	1215	$ABZ5-7\times7$	5144	$ABZ9-8\times8$	2213	$LA25-9\times9$	5880
$ORB08-6\times6$	2355	$ABZ6-7\times7$	3936	$FT10-8\times8$	4559	$LA27-9\times9$	5608
$ORB09-6\times6$	2312	$ABZ7-7\times7$	1764	$LA21-8\times8$	4459	$LA29-9\times9$	5732
$ORB10-6\times6$	2889	$ABZ8-7\times7$	1886	$LA24-8\times8$	4641	$LA38-9\times9$	5855
$ABZ5-6\times6$	3767	$ABZ9-7\times7$	1577	$LA25-8\times8$	4750	$LA40-9\times9$	5006
$ABZ6-6\times6$	2971	$FT10-7\times7$	3420	$LA27-8\times8$	4381	$YN1-9\times9$	3187
$ABZ7-6\times6$	1109	$LA21-7\times7$	3499	$LA29-8\times8$	4540	$YN2-9\times9$	3198
$ABZ8-6\times6$	1365	$LA24-7\times7$	3442	$LA38-8\times8$	4439	$YN3-9\times9$	3206
$ABZ9-6\times6$	1164	$LA25-7\times7$	3581	$LA40-8\times8$	4090	$YN4-9\times9$	3257
$FT10-6\times6$	2544	$LA27-7\times7$	3219	$YN1-8\times8$	2480	$ABZ7-10\times10$	3262
$LA21-6\times6$	2613	$LA29-7\times7$	3383	$YN2-8\times8$	2510	$ABZ8-10\times10$	**3448, P**
$LA24-6\times6$	2514	$LA38-7\times7$	3407	$YN3-8\times8$	2459	$ABZ9-10\times10$	3483
$LA25-6\times6$	2598	$LA40-7\times7$	3128	$YN4-8\times8$	2618	$LA21-10\times10$	**7053, P-h_3**
$LA27-6\times6$	2416	$YN1-7\times7$	1947	$LA16-9\times9$	5724	$LA24-10\times10$	**6989, P-h_4**
$LA29-6\times6$	2353	$YN2-7\times7$	1988	$LA17-9\times9$	5390	$LA25-10\times10$	**6767, P**
$LA38-6\times6$	2602	$YN3-7\times7$	1883	$LA18-9\times9$	5770	$LA27-10\times10$	**6855, P**
$LA40-6\times6$	2211	$YN4-7\times7$	1883	$LA19-9\times9$	5891	$LA29-10\times10$	**6652, P**
$YN1-6\times6$	1418	$LA16-8\times8$	4600	$LA20-9\times9$	5915	$LA38-10\times10$	7113
$YN2-6\times6$	1545	$LA17-8\times8$	4366	$ORB01-9\times9$	6367	$LA40-10\times10$	**6208, P-h_4**
$YN3-6\times6$	1436	$LA18-8\times8$	4690	$ORB02-9\times9$	5867	$YN1-10\times10$	**3839, P**
$YN4-6\times6$	1382	$LA19-8\times8$	4612	$ORB03-9\times9$	6310	$YN2-10\times10$	**3926, P**
$LA16-7\times7$	3554	$LA20-8\times8$	4616	$ORB04-9\times9$	6661	$YN3-10\times10$	**3871, P**
$LA17-7\times7$	3488	$ORB01-8\times8$	4743	$ORB05-9\times9$	5605	$YN4-10\times10$	**4092, P-h_4**
$LA18-7\times7$	3496	$ORB02-8\times8$	4678	$ORB06-9\times9$	6106		
$LA19-7\times7$	3627	$ORB03-8\times8$	4925	$ORB07-9\times9$	2668		

Capítulo 6

RELAJACIÓN DE LAS CONDICIONES DE OPTIMALIDAD

6.1. Introducción

En este capítulo consideramos la combinación del método de poda por dominancia con versiones del algoritmo A^* que permiten obtener soluciones subóptimas cuando los recursos computacionales son insuficientes para alcanzar soluciones óptimas. Concretamente consideraremos estrategias como el cálculo de cotas superiores con un algoritmo voraz, la búsqueda en anchura limitada (con el uso del parámetro delta en la generación de sucesores), la ponderación de la función heurística, la propagación de restricciones (en el caso del problema $J||C_{max}$) y la búsqueda con discrepancia heurística limitada (para el problema $J||\sum C_i$). De esta forma obtendremos versiones no admisibles del algoritmo de búsqueda (con la excepción del método de cálculo de cotas superiores y el de propagación de restricciones) que ofrecen soluciones subóptimas explotando al máximo los recursos computacionales disponibles.

A medida que el tamaño y la dificultad de los problemas $JSSP$ crece, el algoritmo A^* requiere consumir más recursos (tiempo y memoria), por lo que en muchas ocasiones, con los recursos disponibles, no es capaz de llegar a ninguna solución para el problema. Como hemos visto en los capítulos 4 y 5, en condiciones de optimalidad, para ambas versiones del problema $J||C_{max}$ y $J||\sum C_i$, el límite de problemas que puede resolver el algoritmo A^* de forma óptima está cercano al umbral de tamaño 10×10, aunque se queda un poco más lejos para la versión del problema $J||\sum C_i$. Para esta versión resolvíamos sin recortar las instancias $FT06$, $LA01-05$ y de las de tamaño 10×10 recortadas sólo se resuelven tres; mientras, que en la versión $J||C_{max}$ resolvíamos sin recortar la $FT06$, 14 instancias de las $LA01-15$, las instancias de 10×10 sin recortar $LA18$, $LA20$, $ABZ6$, $FT20$, $ORB07$ y $ORB10$, así como las recortadas hasta un tamaño de 10×10, exceptuando una.

El principal problema de la versión del algoritmo A^* empleada en los capítulos 4 y 5 es que para las instancias que no es capaz de resolver con los recursos disponibles, no ofrece ninguna solución. A lo largo del desarrollo de esta tesis hemos visto diversas estrategias que nos permiten calcular

soluciones, para el problema $JSSP$, relajando la optimalidad. En este capítulo vamos a tratar de ofrecer alguna solución, aunque no sea la óptima, para las instancias no resueltas.

Este capítulo se divide en dos secciones, una en la que experimentamos con el problema $J||C_{max}$ y otra en la que lo hacemos con el problema $J||\sum C_i$. Los resultados experimentales ponen de manifiesto que la eficacia de las técnicas empleadas depende de la dificultad del problema a resolver. Así pues, para el problema $J||C_{max}$ el cálculo de cotas superiores, la búsqueda en anchura limitada y la ponderación de la función heurística permiten resolver problemas no resueltos de otro modo. Sin embargo, en el caso del problema $J||\sum C_i$, ninguna de las estrategias ofrece mejoras significativas salvo el método de ponderación adaptativa al dominio.

La máquina donde se realizan los experimentos tiene un procesador Intel Core 2 Duo a $2,13$ GHz y con $7,6$ Gb. de RAM y los experimentos se han ejecutado sobre Ubuntu $V8{,}04$, una distribución de GNU/Linux.

6.2. Problema $J||C_{max}$

Para esta batería de experimentos hemos seleccionado de las instancias contempladas en el estudio experimental del capítulo 4, las instancias no recortadas de tamaño 10×10. Además, para todos ellos se ha empleado el heurístico h_3, sección 4.5.3, que es el mejor heurístico de los propuestos.

Como ya hemos indicado en la sección 4.7.3, el algoritmo de ramificación y poda propuesto por P. Brucker, B. Jurisch y B. Sievers en [12] es la mejor aproximación conocida hasta el momento para resolver de forma exacta el problema $J||C_{max}$. Las técnicas que proponemos en esta sección producirá peores resultados que los que proporciona este algoritmo. Sin embargo, que el algoritmo A^* resuelva la instancia $FT20$ (no resuelta por el algoritmo de Brucker en tres días de ejecución) junto con la ventaja de la adaptabilidad de la técnica de poda por dominancia a otras versiones del problema, hace que sea interesante ver el comportamiento de la poda por dominancia con estas técnicas.

6.2.1. Cálculo de Cotas Superiores

Para el cálculo de cotas superiores (UB) se emplea el algoritmo $G\&T$ descrito en la sección 2.4.1, ligeramente modificado para su aplicación a un estado intermedio n de la búsqueda. Este algoritmo determina cuáles, de todas las tareas posibles, son las siguientes que pueden ser planificadas para mantener la construcción de una planificación activa. Como se ha visto, este algoritmo no es determinista, por lo que se ha de forzar la elección de una de estas tareas susceptibles de ser planificadas. La elección de esta tarea puede hacerse de manera puramente estocástica ó guiada por un heurístico. Nuestra propuesta es utilizar en este punto el heurístico h_3. Esta elección con cada una de las tareas candidatas a ser planificadas, determinadas por el algoritmo, hace lo siguiente: considera que se planifica la tarea y se calcula la planificación JPS (consultar la sección 2.3.1) de todas las tareas que quedan sin planificar y que emplean su mismo recurso, es decir, se simula que esta tarea va a estar planificada antes que el resto de tareas que emplean su mismo recurso y que aún están sin planificar. Una vez hecho esto para todas las tareas candidatas a ser planificadas, será

el valor del coste de estas planificaciones el que determine cuales de ellas son mejores (las que tengan menor coste), es decir, cuales deberían ser planificadas primero. Este método no es determinista por lo que cuando haya más de una tarea en estas condiciones se elegirá entre ellas de forma aleatoria. El método de elección, por tanto, también tiene una componente aleatoria.

El algoritmo A^* con el cálculo de cotas superiores, emplea las cotas calculadas para podar el espacio de búsqueda, terminando bien cuando se certifica que la mejor solución encontrada hasta el momento es la óptima ($ABIERTA$ se queda vacía, ó la cota inferior (LB) del nodo expandido es igual a la cota superior (UB)), o cuando se agota el tiempo límite establecido o la memoria disponible. En las tablas de resultados de esta experimentación se marcan en verde las soluciones para las que se certificó la optimalidad, y en azul las soluciones óptimas no certificadas.

Calcular una cota superior (UB), de este modo, para todos los nodos expandidos supone un coste en tiempo demasiado elevado, por lo que en estos experimentos, cada vez que el algoritmo A^* expande un nodo se calcula, con cierta probabilidad, una cota superior de la solución. A la hora de elegir la probabilidad se estudiaron dos alternativas, una probabilidad fija de 0,01 ó una probabilidad proporcional a la profundidad del nodo (a mayor profundidad, mayor probabilidad de cálculo de UB). Para ello se hicieron unos sencillos experimentos en los que se realizó una ejecución del algoritmo A^* en las mejores condiciones, utilizando la técnica de poda y sin límite de tiempo, para ambas probabilidades. Los resultados de estos experimentos se pueden ver en la Tabla 6.1. En esta tabla se muestra, para cada problema, la solución óptima, las mejores cotas superiores alcanzadas y las podas realizadas por dichas cotas con ambas probabilidades, antes de agotar la memoria, y el tiempo en que ésta se agotó o se certificó que la mejor solución encontrada es la óptima. Como se observa en la tabla, la probabilidad proporcional a la profundidad tiene un mejor comportamiento. Por un lado certifica la solución óptima para 8 problemas, de los cuales el $ORB08$ y $ABZ6$ se certifican por cumplirse la igualdad $LB = UB$ y el resto por el hecho de que $ABIERTA$ se queda vacía. Además encuentra una cota superior igual al óptimo para los problemas $ORB01$ y $ORB09$. Con probabilidad fija se podan menos nodos por UB y se certifica la solución óptima para 5 problemas, el $LA18$ por quedarse $ABIERTA$ vacía y los demás por darse la igualdad $LB = UB$. En el resto de problemas alcanza soluciones peores que la probabilidad proporcional a la profundidad.

Con los resultados de estos experimentos vamos a analizar si la calidad de los UB calculados es buena, esto nos dará una idea de lo eficiente que puede resultar esta técnica. En la Figura 6.1 se ve para todos los problemas,la evolución de las cotas superiores, calculadas con probabilidad proporcional a la profundidad. Como se observa la evolución de dichas cotas es muy buena, ya que la cota superior inicial no está muy alejada de la solución óptima, además la cota inferior (LB) también es muy buena con lo que al no ser el intervalo $[LBini, UBini]$ demasiado grande, el cálculo de UB puede producir una buena reducción del espacio de búsqueda. En las Figuras 6.2 y 6.3, se puede ver ampliado lo que ocurre para dos de los problemas, uno que se resuelve, el $LA19$ y otro que no, el $ORB02$.

Una vez determinado que la mejor probabilidad es la proporcional a la profundidad, resolveremos ahora estos problemas ejecutando 20 veces el algoritmo A^* para cada instancia, empleando la técnica de poda y sin ella, con un límite de tiempo de 3600 segundos para cada ejecución. Los resultados de estos experimentos se pueden ver en la Tabla 6.2, en la que para cada instancia se muestra

la solución óptima, la mejor solución alcanzada y el número de veces que se alcanzó, así como las medias del número de nodos generados y expandidos, del tiempo de ejecución y de las cotas superiores calculadas. Los resultados de estos experimentos reflejan que en todos los casos la técnica de poda por dominancia produce mejores soluciones y en menos tiempo, ya que sin ella se agota antes la memoria. Además, con la técnica de poda se resuelven óptimamente 8 problemas, mientras que sin ella sólo 1, siendo el tiempo de ejecución con poda para ese problema mucho más pequeño. Con la técnica de poda hay 9 problemas que agotan el tiempo sin certificar la solución óptima, de los que para el $LA19$, $ORB01$ y $ORB09$ la mejor cota superior alcanzada coincide con la solución óptima. Para el resto de problemas se certifica la solución óptima, porque se cumple la igualdad $LB = UB$ para el $ABZ6$ y porque se vacía $ABIERTA$ para el $LA18$, $LA20$, $ORB07$ y $ORB10$. Sin poda sólo se certifica la solución óptima para el problema $ORB10$, terminando en 4 casos por agotar el tiempo límite establecido y en el resto por agotar la memoria.

El empleo del cálculo de cotas superiores junto con la técnica de poda por dominancia consigue resolver tres instancias que sin el cálculo de cotas no se resolvían: la $LA19$, $ORB01$ y $ORB09$, aunque el algoritmo no es capaz de certificar que son óptimas. Sin límite de tiempo, de estos 3 problemas, el algoritmo sólo certifica la solución óptima para el $LA19$. Otro detalle que se observa en los resultados de la Tabla 6.2 es que las 20 ejecuciones son muy similares entre sí, esto se debe principalmente a que la componente aleatoria del cálculo de cotas es muy pequeña, es decir, la secuencia de cotas superiores que se obtienen en distintas ejecuciones del algoritmo es muy similar.

Además de estos problemas, se ha resuelto con el cálculo de cotas el problema $LA11$, único problema de los $LA01-20$ que no se resolvía óptimamente. Este problema se resuelve expandiendo solamente un nodo, ya que la cota superior calculada para el nodo inicial es igual a la cota inferior de ese nodo, con lo que se certifica que la solución de coste 1222 es la óptima para este problema.

Tabla 6.1: Problema $J||C_{max}$. Cotas superiores calculadas con probabilidad fija 0,01 y probabilidad proporcional a la profundidad, para una ejecución del algoritmo A^* empleando la técnica de poda

		$P = 0{,}01$			P = proporcional prof.		
Inst.	Opt.	Mejor UB	Podas UB	T.(s)	Mejor UB	Podas UB	T.(s)
$LA16$	945	960	164453	2955	960	167759	4278
$LA17$	784	785	463921	219	784	467168	338
$LA18$	848	848	236803	3434	848	255248	4243
$LA19$	842	842	345533	3414	842	416388	4916
$LA20$	902	902	218791	2872	902	287040	3556
$ORB01$	1059	1060	450223	2414	1059	557856	2636
$ORB02$	888	896	207473	3840	889	318983	5461
$ORB03$	1005	1063	4031	2890	1055	6767	3268
$ORB04$	1005	1015	205627	1782	1015	124285	1623
$ORB05$	887	894	236769	5829	889	350782	6766
$ORB06$	1010	1032	226378	3795	1031	233148	4365
$ORB07$	397	397	103238	102	397	145766	166
$ORB08$	899	911	347382	6796	899	368436	7856
$ORB09$	934	937	320952	4840	934	365012	6101
$ORB10$	944	944	46933	1640	944	52079	2070
$ABZ05$	1234	1255	189148	4330	1241	299859	4276
$ABZ06$	943	945	68719	2271	943	69853	2760
$FT10$	930	959	71615	2399	939	268086	3649

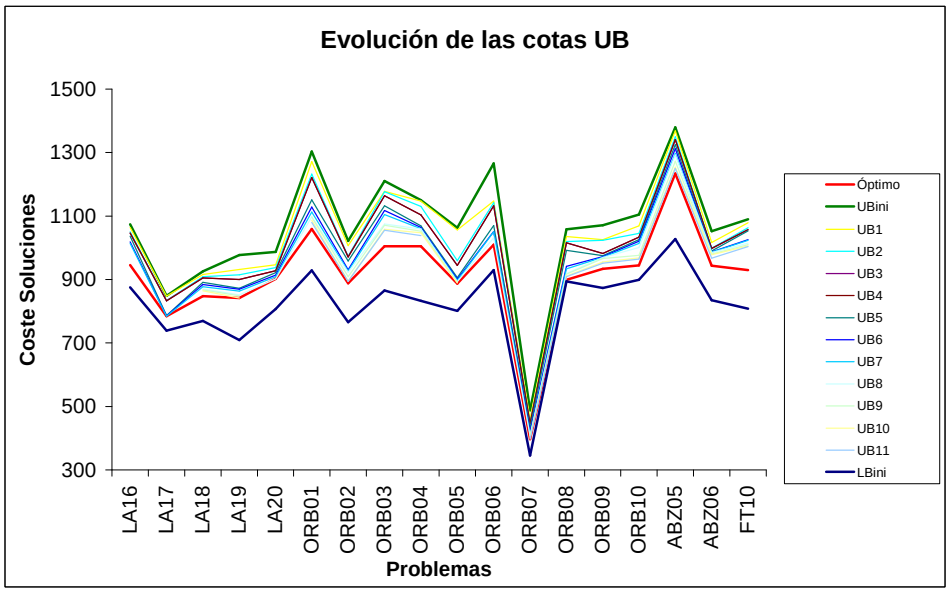


Figura 6.1: Problema $J||C_{max}$. Gráfica que muestra la evolución de las cotas superiores calculadas con probabilidad proporcional a la profundidad

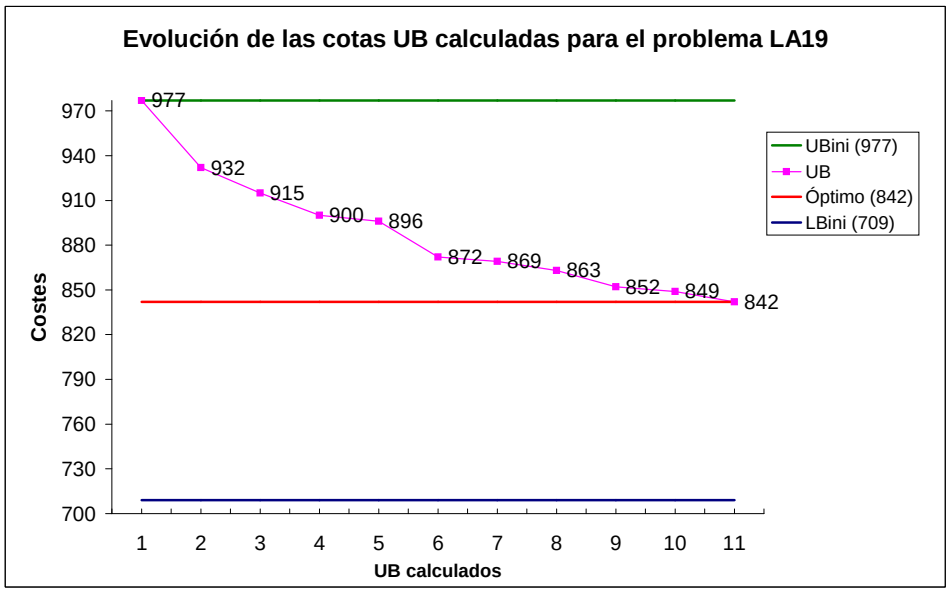


Figura 6.2: Problema $J||C_{max}$. Gráfica que muestra la evolución de las cotas superiores calculadas, con probabilidad proporcional a la profundidad, para el problema $LA19$

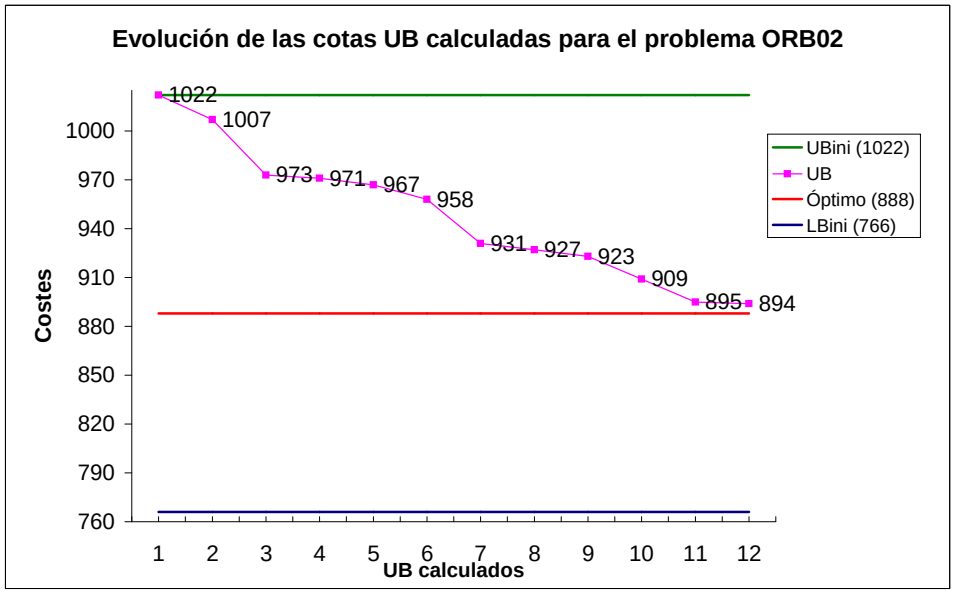


Figura 6.3: Problema $J||C_{max}$. Gráfica que muestra la evolución de las cotas superiores calculadas, con probabilidad proporcional a la profundidad, para el problema $ORB02$

Tabla 6.2: Problema $J||C_{max}$. Resultados de las 20 ejecuciones con el algoritmo A^* calculando cotas superiores con una probabilidad proporcional a la profundidad. Tiempo límite 3600 segundos

No Poda							
Inst.	Opt.	Gen.	Exp.	T.(s)	Mejor UB	UB	Veces
$LA16$	945	2142773,85	1049210,8	2439,1	996	996	20
$LA17$	784	2335217,1	1242646,65	3600	786	786	20
$LA18$	848	5070921,2	2709158,1	3169,45	855	855	20
$LA19$	842	4138626,9	2237272,25	2916,05	852	852	20
$LA20$	902	3624974,65	2093107,85	3600	909	909	20
$ORB01$	1059	3408684,1	1754399	1647,55	1072	1072	20
$ORB02$	888	3131950,25	1740896,6	2683,1	909	909	20
$ORB03$	1005	1910070,75	814167,1	1349,45	1058	1058	20
$ORB04$	1005	1951715,7	837087,1	1057,2	1025	1028,55	3
$ORB05$	887	2994218,25	1584210,3	2372,3	897	897	20
$ORB06$	1010	2844045,4	1460790,4	1391,75	1031	1031	20
$ORB07$	397	3566634,4	1979754,75	2008,4	405	405	20
$ORB08$	899	4604201,6	2577462,7	3600	917	917	20
$ORB09$	934	3365971,25	1748999,3	2056,45	951	951	20
$ORB10$	944	4308824	2372644	2051,45	944	944	20
$ABZ05$	1234	2619656,3	1401884,85	2405,6	1250	1250	20
$ABZ06$	943	4450072,1	2489173,65	3600	948	948	20
$FT10$	930	2091966,9	912903,15	1158,55	952	952	20
Poda							
Inst.	Opt.	Gen.	Exp.	T.(s)	Mejor UB	UB	Veces
$LA16$	945	765276,7	411032	3600	960	960	20
$LA17$	784	1142034	644807,95	3600	785	785	20
$LA18$	848	878465	492229	2078,95	848	848	20
$LA19$	842	1142561,8	634854,45	3600	842	842	20
$LA20$	902	987639	582758	2779,8	902	902	20
$ORB01$	1059	1582743,95	831865,4	3600	1059	1059	20
$ORB02$	888	1337523,4	751482	3600	889	889	20
$ORB03$	1005	1242045,7	569878,2	3578,3	1055	1055,3	17
$ORB04$	1005	1391149,75	694110,75	2795,8	1015	1015	20
$ORB05$	887	1267921,4	682224	3600	889	889	20
$ORB06$	1010	1476680,7	778536,55	3290,15	1026	1030,75	1
$ORB07$	397	649096	366469	1633,8	397	397	20
$ORB08$	899	1227271,3	698669,35	3600	906	906	20
$ORB09$	934	1369983,35	697644,3	3600	934	934	20
$ORB10$	944	163700	90734	166	944	944	20
$ABZ05$	1234	1372280,15	775450,4	3600	1241	1241	20
$ABZ06$	943	239658,55	139603,45	340,7	943	943	20
$FT10$	930	1479895,95	715880,2	3571,75	939	939	20

6.2.2. Reducción del Espacio de Búsqueda con el Parámetro δ en la Generación de Sucesores

Como hemos visto en la sección 4.3, el espacio de búsqueda que manejamos es el formado por las planificaciones activas. Este espacio de búsqueda se genera mediante el Algoritmo 4.1 que es una versión reducida del Algoritmo $G\&T$ visto en la sección 2.4.1. El algoritmo $G\&T$ en su versión híbrida (Algoritmo 2.3) restringe, mediante el parámetro de reducción $\delta \in [0,1]$, los intervalos de inactividad de las máquinas cuando hay tareas disponibles. Modificando con esta idea el algoritmo de generación de sucesores reducimos el espacio de búsqueda perdiendo la completud. Así, con $\delta = 1$ no habrá reducción y tendremos el mismo espacio de planificaciones activas manejado hasta el momento. Con $\delta < 1$, el espacio de búsqueda se restringe a un subconjunto de las planificaciones activas; lo que hace imposible garantizar que se pueda alcanzar una solución óptima. Con $\delta = 0$, el algoritmo produce planificaciones densas [9].

En esta sección, realizamos una serie de experimentos con el algoritmo A^*, reduciendo el espacio de búsqueda con $\delta = 0{,}5$. Utilizamos este valor por tratarse de una reducción media del espacio de búsqueda. Como el objetivo perseguido con estos experimentos es encontrar alguna solución para los problemas contemplados, aunque no sea la óptima, se realizan estos experimentos sin límite de tiempo. Además para ver el efecto de la técnica de poda con la reducción del espacio de búsqueda realizaremos los mismos empleando la técnica de poda y sin ella.

Los resultados de esta experimentación se recogen en la Tabla 6.3. En ella, para cada problema se muestra la solución óptima, los nodos generados y expandidos, el tiempo de ejecución y la mejor solución o bien la mejor cota inferior alcanzada en ese espacio de búsqueda. Para los problemas que se resuelven se marca en verde la solución cuando es óptima y en azul si no lo es. Para los problemas no resueltos se muestra, el número de nodos expandidos y el tiempo empleado en la búsqueda de la solución. Como se observa en la Tabla 6.3, utilizando la técnica de poda junto con la reducción del espacio de búsqueda, se alcanza una solución para problemas que sin dicha reducción y con la técnica de poda no eran resueltos: $LA16$, $LA17$, $LA19$, $ORB02$ y el $ABZ5$, alcanzando la solución óptima para los problemas $LA16$, $LA17$, $LA18$, $ORB02$ y $ABZ5$. Sin embargo, esta reducción, hace que dejemos fuera del espacio de búsqueda las soluciones óptimas de los problemas $LA20$, $ORB07$, $ORB10$ y $ABZ06$, que sí se resuelven óptimamente mediante la técnica de poda. Para los problemas $ORB10$ y $ABZ6$, se llega a la misma solución sin la técnica de poda y con ella, pero el número de nodos expandidos y el tiempo de ejecución es mucho menor empleando la técnica de poda.

Tabla 6.3: Problema $J||C_{max}$. A^* con reducción del espacio de búsqueda $\delta = 0{,}5$

		No Poda				Poda			
Ins.	Opt.	Gen.	Exp.	T.(s)	Sol./LB	Gen.	Exp.	T.(s)	Sol./LB
$LA16$	945	2761185	1668640	428		983037	628779	1544	945
$LA17$	784	2918621	1826290	429		267386	176216	177	784
$LA18$	848	3138974	2046580	473		179851	120873	94	848
$LA19$	842	3168076	2075815	472		240514	161394	160	846
$LA20$	902	3406601	2314376	486		326214	231897	273	907
$ORB01$	1059	2830221	1737897	472		1189894	747580	2040	
$ORB02$	888	3012255	1919876	468		750825	501454	908	888
$ORB03$	1005	2209134	1116813	407		1222685	696119	3058	
$ORB04$	1005	2282378	1189770	387		1164198	681470	2298	
$ORB05$	887	3093831	2001599	487		1174816	773412	2642	
$ORB06$	1010	2977633	1885362	491		1189891	759413	1856	
$ORB07$	397	3241936	2149710	515		529076	353857	1170	401
$ORB08$	899	3099832	2007519	460		1174273	800645	4359	
$ORB09$	934	2944907	1852395	466		1145563	708285	9348	
$ORB10$	944	1015004	717248	152	946	78776	54538	31	946
$ABZ05$	1234	3448648	2356405	534		830297	566109	1148	1234
$ABZ06$	943	783099	558208	104	947	39606	28312	12	947
$FT10$	930	2270837	1178303	398		1196471	716340	3194	

6.2.3. Ponderación de la Función Heurística

Posiblemente la técnica más utilizada para agilizar la búsqueda a costa de relajar las condiciones de optimalidad es la ponderación de la función heurística (sección 3.4.2). Esta técnica hace que se alcance una solución de forma más rápida, pues hace más atractivos los nodos cercanos a la solución. En esta sección emplearemos la ponderación estática (sección 3.4.2), por ser la aproximación más utilizada en la literatura y que mejores resultados aporta según nuestra experiencia. Como vimos esta técnica consiste en sustituir la función heurística original por $wh(n)$ siendo la función de evaluación en este caso $f'(n) = g(n) + wh(n)$, con $w > 1$. Los resultados experimentales de esta sección se recogen en una serie de tablas en las que para cada problema se muestra la solución óptima, el número de nodos expandidos, el tiempo de ejecución y la solución ponderada alcanzada.

En el primer grupo de experimentos resolvemos los problemas con el algoritmo A^*, con w tomando los valores [1,25; 1,5; 1,75; 2], con y sin la técnica de poda, y sin límite de tiempo. Como se observa en la Tabla 6.4, a medida que la ponderación es menor la calidad de las soluciones es mayor, siendo el tiempo requerido para alcanzar la primera solución menor. El efecto de la poda cuando nos quedamos con la primera solución ponderada es pequeño, pues se expanden pocos nodos y no se realizan muchas podas. De hecho, la primera solución ponderada alcanzada sin utilizar la técnica de poda, es la misma que la obtenida con ella para la mayoría de los problemas, con la excepción de los problemas $ORB09$ y $ABZ06$ con ponderación $w = 1{,}75$ para los que la calidad de la solución sin utilizar la técnica de poda es mejor, y los problemas $ORB04$ y $FT10$ con ponderación $w = 1{,}5$ para los que la calidad de la solución es mejor utilizando la técnica de poda. A medida que la ponderación requiere expandir mayor número de nodos para alcanzar una solución el efecto de la técnica de poda se va haciendo cada vez más patente. Así, con la ponderación más pequeña, ponderación $w = 1{,}25$, la poda reduce considerablemente el número de nodos expandidos y el tiempo para llegar a la misma solución que sin poda.

Tras estos experimentos, convertimos el algoritmo A^* en un algoritmo *"Anytime"* que permite encontrar todas las soluciones y podar el espacio de búsqueda utilizando como cotas superiores (UB) las soluciones que va encontrando (para ello cada nodo guarda tanto la h ponderada como sin ponderar). Utilizando esta versión *"Anytime"* del algoritmo A^* ($A^* - Anytime$) junto con la técnica de poda resolvemos de nuevo los problemas, obteniendo soluciones hasta agotar la memoria disponible. Los resultados de estos experimentos se pueden ver en la Tabla 6.5, en la que se muestran en verde las soluciones ponderadas encontradas que son óptimas. Como se observa, en general, a medida que la ponderación es más pequeña la calidad de las soluciones es más alta, siendo el tiempo en que se agota la memoria menor, ya que se requiere expandir mayor número de nodos. A la vista de estos resultados, la mejor ponderación para este grupo de problemas es la que emplea $w = 1{,}25$, ya que con ella las soluciones alcanzadas son mejores y se resuelven óptimamente 3 problemas, 2 de los cuales no eran resueltos con la técnica de poda y sin ponderación ($LA19$ y $ORB01$). En la Tabla 6.6 podemos ver los resultados para la ponderación $w = 1{,}25$ con y sin el empleo de la técnica de poda. Como se observa, sin la técnica de poda, todas las soluciones alcanzadas son peores que las que se alcanzan con ella, no encontrando para ningún problema la solución óptima y expandiendo muchos más nodos. Por tanto, podemos concluir que la técnica de poda también

muestra un comportamiento eficiente con la ponderación estática.

Tabla 6.4: Problema $J||C_{max}$. A^* con ponderación estática. El algoritmo se detiene cuando encuentra la primera solución

No Poda													
		$w=2$			$w=1{,}75$			$w=1{,}5$			$w=1{,}25$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
*LA*16	945	121	0	1098	139	0	1051	185	0	988	1514	0	1018
*LA*17	784	122	0	869	116	0	853	129	0	848	157	0	821
*LA*18	848	151	0	923	154	0	923	334	0	932	77290	25	883
*LA*19	842	120	0	935	131	0	942	256	0	907	11418	3	878
*LA*20	902	125	0	1032	119	0	1009	367	0	969	3289	1	961
*ORB*01	1059	139	0	1242	165	0	1252	182	0	1198	43267	10	1112
*ORB*02	888	118	0	968	204	0	981	1269	1	979	13742	4	910
*ORB*03	1005	191	0	1214	159	1	1141	565	0	1141	45999	16	1057
*ORB*04	1005	452	0	1158	3499	1	1167	40646	19	1108	271753	115	1072
*ORB*05	887	129	1	1010	141	0	1004	296	0	1022	1573	0	956
*ORB*06	1010	181	0	1214	447	0	1227	912	1	1136	8588	3	1078
*ORB*07	397	109	0	470	127	1	445	190	0	422	1446	0	411
*ORB*08	899	165	0	1039	118	0	1054	127	0	1007	1684	1	1007
*ORB*09	934	154	0	1115	240	0	1069	1054	0	1052	6892	3	993
*ORB*10	944	118	0	1113	118	0	1113	146	0	1107	549	0	1027
*ABZ*05	1234	121	0	1354	218	0	1335	1650	1	1317	166280	46	1268
*ABZ*06	943	148	0	1079	126	0	1022	400	0	1022	2488	1	997
*FT*10	930	148	0	1105	170	0	1128	1851	1	1039	61205	29	968
Poda													
		$w=2$			$w=1{,}75$			$w=1{,}5$			$w=1{,}25$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
*LA*16	945	121	0	1098	139	0	1051	185	0	988	1106	0	1018
*LA*17	784	122	0	869	116	0	853	129	0	848	156	0	821
*LA*18	848	143	0	923	149	0	923	253	0	932	15049	6	883
*LA*19	842	120	0	935	131	0	942	233	1	907	4730	2	878
*LA*20	902	125	0	1032	119	0	1009	330	0	969	1690	0	961
*ORB*01	1059	138	0	1242	161	0	1252	169	0	1198	4920	2	1112
*ORB*02	888	118	0	968	189	1	981	950	0	979	6845	2	910
*ORB*03	1005	158	0	1214	144	0	1141	400	0	1141	10433	5	1057
*ORB*04	1005	329	0	1158	1693	1	1167	12456	7	1051	48889	25	1072
*ORB*05	887	129	0	1010	141	0	1004	296	0	1022	963	0	956
*ORB*06	1010	181	0	1214	312	0	1227	516	0	1136	3763	2	1078
*ORB*07	397	109	0	470	127	0	445	170	0	422	1106	0	411
*ORB*08	899	165	0	1039	118	0	1054	127	0	1007	1110	1	1007
*ORB*09	934	154	0	1115	229	0	1096	862	1	1052	5261	2	993
*ORB*10	944	118	0	1113	118	0	1113	146	0	1107	438	1	1027
*ABZ*05	1234	121	0	1354	218	0	1335	941	0	1317	12799	4	1268
*ABZ*06	943	148	1	1079	175	0	1081	370	0	1022	1824	1	997
*FT*10	930	148	0	1105	170	0	1128	989	1	993	10919	6	968

Tabla 6.5: Problema $J||C_{max}$. $A^* - Anytime$ con ponderación estática y la técnica de poda. Encuentra todas las soluciones posibles hasta agotar la memoria

		Poda											
		$w = 2$			$w = 1{,}75$			$w = 1{,}5$			$w = 1{,}25$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$LA16$	945	1042245	291	1000	967428	280	1009	902408	272	988	755529	266	960
$LA17$	784	1046495	284	843	1038365	270	822	978275	296	805	878011	271	792
$LA18$	848	1045185	273	905	1035688	264	905	1003551	263	905	862625	261	855
$LA19$	842	1056071	279	895	1040160	275	889	996205	275	875	819024	300	842
$LA20$	902	1050718	285	978	1027987	269	958	969676	272	935	842247	296	902
$ORB01$	1059	1065206	320	1198	1043650	323	1175	996167	293	1122	837989	294	1059
$ORB02$	888	1056590	279	939	1031347	277	934	977284	275	933	815161	291	889
$ORB03$	1005	1063769	328	1148	1049424	324	1092	977510	312	1060	727864	326	1022
$ORB04$	1005	1040441	295	1103	1033377	290	1105	962124	269	1039	760677	270	1027
$ORB05$	887	1066990	267	958	1034249	262	956	988843	260	925	799727	244	891
$ORB06$	1010	1052348	340	1100	1020274	318	1124	964923	331	1054	820055	299	1031
$ORB07$	397	1060815	277	433	1052191	275	422	1020322	274	417	872990	345	401
$ORB08$	899	1051473	338	962	1011945	319	962	950549	330	961	766340	355	942
$ORB09$	934	1046953	275	1033	1037967	279	1049	986790	282	1024	841681	250	944
$ORB10$	944	1063302	279	1068	1059759	276	1068	1029278	270	1068	905229	264	1005
$ABZ05$	1234	1060434	279	1295	1056705	272	1260	943310	259	1282	782447	276	1239
$ABZ06$	943	1059655	290	1037	1025351	283	1020	1013440	273	1002	843529	246	962
$FT10$	930	1064107	307	1048	1031528	296	1042	938956	311	970	777009	314	938

Tabla 6.6: Problema $J||C_{max}$. $A^* - Anytime$ con ponderación estática $w = 1{,}25$, con y sin la técnica de poda. Encuentra todas las soluciones posibles hasta agotar la memoria

		Poda			*No Poda*		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$LA16$	945	755529	266	960	6393398	889	979
$LA17$	784	878011	271	792	15045060	1844	805
$LA18$	848	862625	261	855	10494475	1370	871
$LA19$	842	819024	300	842	8092324	1075	848
$LA20$	902	842247	296	902	7063955	961	908
$ORB01$	1059	837989	294	1059	6264353	875	1107
$ORB02$	888	815161	291	889	8056962	1074	896
$ORB03$	1005	727864	326	1022	3713895	600	1042
$ORB04$	1005	760677	270	1027	4459669	700	1038
$ORB05$	887	799727	244	891	5852538	844	907
$ORB06$	1010	820055	299	1031	14451295	1808	1042
$ORB07$	397	872990	345	401	15177169	1846	406
$ORB08$	899	766340	355	942	5008449	727	961
$ORB09$	934	841681	250	944	7775185	1049	968
$ORB10$	944	905229	264	1005	13403778	1667	1012
$ABZ05$	1234	782447	276	1239	4101417	663	1253
$ABZ06$	943	843529	246	962	8638697	1135	972
$FT10$	930	777009	314	938	12852784	1620	958

6.2.4. Propagación de Restricciones

Como hemos visto en la sección 2.5.1, Brucker emplea en su algoritmo de ramificación y poda una técnica de propagación de restricciones que le permite fijar arcos adicionales, concretamente la *Selección Inmediata* (Anexo 4.10). En esta tesis hemos estudiado el efecto que tiene la *Selección Inmediata* en el espacio de búsqueda de las planificaciones activas, recorrido por el algoritmo A^*. Para ello, se ha estudiado su aplicación en el algoritmo A^* en dos puntos: cada vez que se planifica una tarea (sea en la generación de sucesores o en el cálculo de cotas superiores) y a la hora de generar el conjunto de sucesores de un estado. Además, la *Selección Inmediata*, en el algoritmo A^*, se aplica al estado inicial.

La *Selección Inmediata* requiere para su aplicación una cota superior de la solución del problema a resolver. Cuanto mejor sea esa cota mayor efecto producirá la *Selección Inmediata*.

Para ver el efecto real de esta propagación, se resolvieron los problemas $LA01-15$ y los cinco no reducidos resueltos ($ORB07$, $ORB10$, $LA18$, $LA20$, $ABZ6$) con el algoritmo A^*, combinando el empleo de la *Selección Inmediata* con la técnica de poda y sin límite de tiempo. Estos experimentos se realizaron en las mejores condiciones para la *Selección Inmediata*, es decir con una cota superior de partida igual a la solución óptima, y con un límite de tiempo de una hora. En la Tabla 6.7 se pueden ver los resultados de estos experimentos, aplicando la *Selección Inmediata* tras planificar una tarea (SIP), en la generación del conjunto de sucesores (SIS) o en ambas situaciones (SI). Para cada problema se muestra la solución óptima, el número de nodos expandidos, el tiempo de ejecución y la solución óptima para los problemas que se resuelven sin agotar los recursos (los valores de las soluciones óptimas se ponen en verde) o la mejor cota inferior para los que no se resuelven. Para los problemas que no aparecen en la tabla ($LA01$, $LA02$, $LA05-15$) se certificó que el UB de partida era la solución óptima en el estado inicial, pues su LB es igual al UB de partida, que es el óptimo.

Estos experimentos ponen de manifiesto que la *Selección Inmediata* con el espacio de búsqueda que manejamos no es una técnica demasiado efectiva, ya que aún en las mejores condiciones, cuando se produce una reducción del espacio de búsqueda, ésta es insignificante. Sin embargo, cuando no se emplea la técnica de poda la *Selección Inmediata*, aplicada tras planificar una tarea, se vuelve algo más efectiva, siendo capaz de certificar la solución para el problema $ABZ6$ que sin ella no se puede certificar. Con respecto a esto, parece curioso que la propagación de restricciones requiera expandir 6863792 nodos para certificar la solución del problema $ABZ6$ y que sin embargo, el A^* sólo no pueda certificarla y agote la memoria expandiendo únicamente 1563099 nodos, la explicación está en que la *Selección Inmediata* permite podar por cota superior casi tantos nodos como expande (5271910), lo que se traduce en un gran ahorro de memoria, mientras que el A^* sólo poda 117591 nodos, con lo cual mantiene más nodos en $ABIERTA$. Cuando la cota UB de partida no es la óptima el efecto de la *Selección Inmediata* sólo es apreciable cuando las cotas UB calculadas son o se aproximan mucho a la solución óptima.

En estos experimentos sigue quedando patente la eficiencia de la técnica de poda, ya que certifica la solución óptima para más problemas, expandiendo menos nodos.

Tabla 6.7: Problema $J||C_{max}$. A^* con propagación de restricciones *Selección Inmediata*, con y sin la técnica de poda. La cota superior de partida es el óptimo

No Poda													
		A^*			A^* y SIP			A^* y SIS			A^* y SI		
Inst.	Opt.	Exp	T.(s)	S/LB	Exp	T.(s)	S/LB	Exp	T.(s)	S/LB	Exp	T.(s)	S/LB
$LA03$	597	5	0	597	5	0	597	5	0	597	5	0	597
$LA04$	590	63892	12	590	54606	17	590	54606	38	590	54606	38	590
$LA18$	848	13285021	3600	843	1077582	568	823	190128	1463	806	190128	1468	806
$LA20$	902	5931776	1541	886	1723318	768	885	385643	1191	879	385643	1195	879
$ORB07$	397	6484500	1891	397	6566267	3107	397	251980	1487	383	251980	1481	383
$ORB10$	944	2372644	673	944	2381726	1121	944	721458	1384	937	721457	1376	937
$ABZ6$	943	1563099	422	934	6863792	3002	943	833303	1467	934	833303	1467	934
Poda													
		A^*			A^* y SIP			A^* y SIS			A^* y SI		
Inst.	Opt.	Exp	T.(s)	S/LB	Exp	T.(s)	S/LB	Exp	T.(s)	S/LB	Exp	T.(s)	S/LB
$LA03$	597	5	0	597	5	0	597	5	0	597	5	0	597
$LA04$	590	7714	3	590	6183	3	590	6183	8	590	6183	9	590
$LA18$	848	492229	1627	848	393255	1339	843	210760	1095	843	210760	1089	843
$LA20$	902	582758	2165	902	404092	1463	897	254323	1139	897	254323	1137	897
$ORB07$	397	366469	1390	397	369034	1786	397	226486	1459	393	224547	1425	393
$ORB10$	944	90734	89	944	90845	136	944	90845	348	944	90845	347	944
$ABZ6$	943	139661	219	943	135977	288	943	135977	602	943	135977	588	943

6.2.5. Conclusiones

En esta sección hemos combinado, para el problema $J||C_{max}$, la técnica de poda por dominancia con distintas estrategias, concretamente con el cálculo de cotas superiores (UB), con la reducción del espacio de búsqueda con el parámetro δ, con la ponderación de la función heurística y con la propagación de restricciones. Como hemos visto la propagación de restricciones *Selección Inmediata*, aún en las mejores condiciones, no es efectiva en el espacio de planificaciones activas. Sin embargo, la combinación de la técnica de poda con las otras tres estrategias permite ofrecer una solución, aunque no sea óptima, para los problemas de tamaño 10×10 no resueltos empleando únicamente la técnica de poda. Además, gracias a estas estrategias hemos alcanzado la solución óptima para 7 problemas más de tamaño 10×10 ($LA16$, $LA17$, $LA19$, $ORB01$, $ORB02$, $ORB09$ y $ABZ5$) y para el problema $LA11$, único de la batería $LA01 - 15$ no resuelto.

De estos resultados se puede concluir que cuando los problemas son difíciles, el empleo de estas estrategias está justificado, ya que permiten ofrecer una solución, aunque no sea óptima, en poco tiempo. Además hemos visto que emplear la técnica de poda con ellas mejora su eficiencia, ya que les permite alcanzar mejores soluciones, generalmente, en menos tiempo.

6.3. Problema $J||\sum C_i$

En este sección consideramos el problema $J||\sum C_i$ y hacemos un estudio similar al realizado en la sección anterior para el problema $J||C_{max}$. Además de la búsqueda en anchura limitada (a través del parámetro delta del algoritmo $G\&T$) y de la ponderación estática de la función heurística, en este caso consideramos otras posibilidades como la mejora de los heurísticos mediante el diseño de heurísticos aditivos, así como otras estrategias de búsqueda, como por ejemplo la búsqueda con discrepancia heurística limitada. Como veremos, al combinar la búsqueda con poda por dominancia con alguno de estos métodos, se consiguen algoritmos comparables con otros actuales.

Para esta batería de experimentos hemos seleccionado de las instancias contempladas en el estudio experimental del capítulo 5, las instancias de la batería LA ($LA01-20$), las instancias recortadas de tamaño 9×9 resueltas expandiendo más de 500000 estados ($ORB04-9\times9$, $ORB08-9\times9$, $LA38-9\times9$) y los tres únicos problemas de tamaño 10×10 resueltos ($ABZ7\_10\times10$, $ABZ9\_10\times10$, $LA38\_10\times10$). Además, para todos ellos se ha empleado el heurístico h_2, sección 5.3.2, que es, para la batería de problemas considerada, el mejor heurístico de los propuestos.

Como vimos en la sección 5.5.3, para los problemas resueltos, el algoritmo A^* empleando la técnica de poda, tiene un comportamiento mejor en comparación tanto con un algoritmo genético como con la búsqueda local propuesta en [47]. Aquí, trataremos de resolver los problemas no resueltos con el algoritmo A^*, combinando la técnica de poda con diversas estrategias, algunas en condiciones de no optimalidad. Además, compararemos las soluciones obtenidas con las que nos proporcionan otros métodos, concretamente con el algoritmo genético y la búsqueda local comentados anteriormente.

6.3.1. Mejora de los Heurísticos

En la sección 5.3.5 planteamos un método que permite mejorar las estimaciones de los heurísticos. Como vimos todos los heurísticos diseñados en el capítulo se basan en el cálculo de cotas inferiores independientes para cada una de las partes o subproblemas en que se divide el problema asociado a cada estado n, es decir $P(n)|_{J_m}$ y $P(n)|_{J\overline{m}}$ para cada máquina m con tareas sin planificar en n. La mejora consiste en hacer más precisa la estimación para el problema $P(n)|_{J\overline{m}}$. En esta sección veremos como mejorar el heurístico h_2 para obtener la versión que denominaremos $h_{2'}$ que se define del siguiente modo:

$$\begin{gathered} h_{2'}(n) = F'_2(P(n)) - g(n); \\ F_{2'}(P(n)) = \text{máx}_{m\in R}\{H_2(P(n)|_{J_m}) + F_{2'}(P(n)|_{J_{\overline{m}}})\}; \\ H_2(P(n)|_{J_m}) = \textstyle\sum_{J_i\in J_m}(C_{\theta_i^m} + q_{\theta_i^m}). \end{gathered} \tag{6.1}$$

donde ${\theta_i}^m$ es la tarea del trabajo J_i que requiere la máquina m y $C_{{\theta_i}^m}$ su tiempo de completud en la planificación de Jackson (JPS) modificada, empleada en el heurístico h_2 (sección 5.3.2), en la que las prioridades se asignan en relación inversa a los tiempos de procesamiento remanente. El heurístico $h_{2'}$ es consistente ya que se obtiene como el coste de la solución óptima de una versión relajada del problema original [62].

Los resultados de los experimentos con el algoritmo A^* y el heurístico $h_{2'}$ se recogen en la Tabla 6.8. En ella se compara con el heurístico h_2, y para cada problema se muestran el número de nodos generados y expandidos, el tiempo de ejecución y la solución óptima (en verde) o la mejor cota inferior si el problema no se resuelve antes de agotar la memoria. Como se observa, para los problemas resueltos con $h_{2'}$ se logra una ligera reducción del número de nodos expandidos, sin embargo el tiempo crece mucho con respecto a h_2. Para los problemas no resueltos las cotas LB son un poco mejores. A la vista de estos resultados, para estos problemas no merece la pena utilizar el heurístico mejorado, ya que el ahorro de memoria no compensa el tiempo de ejecución.

Tabla 6.8: Problema $J||\sum C_i$. A^* con la técnica de poda y los heurísticos h_2 y $h_{2'}$

	h_2				h'_2			
Inst.	Gen.	Exp.	T.(s)	Sol./LB.	Gen	Exp.	T.(s)	Sol./LB.
$ORB04-9\times9$	1132648	660501	273	6661	1099672	641797	499	6661
$ORB08-9\times9$	1765913	909440	772	5668	1761805	907246	1118	5668
$ABZ8-9\times9$	927825	535069	286	3078	765777	443172	428	3078
$ABZ7-10\times10$	552246	326967	160	3262	457322	270395	259	3262
$ABZ9-10\times10$	1050968	581996	382	3483	975241	541227	660	3483
$LA38-10\times10$	937039	536287	265	7113	920239	526220	465	7113
$LA01$	248935	107955	35	4832	245494	106593	42	4832
$LA02$	520356	217751	80	4459	508357	212752	104	4459
$LA03$	79883	32118	10	4151	74022	29689	13	4151
$LA04$	136928	57558	19	4259	132726	55790	22	4259
$LA05$	412277	174320	68	4072	372898	157088	75	4072
$LA06$	1334365	402170	437	8136	1337749	403157	489	8136
$LA07$	1344617	390813	420	7802	1345572	391038	454	7802
$LA08$	1356179	401921	398	7252	1356274	401924	408	7252
$LA09$	1348941	406063	428	8218	1348625	405733	450	8219
$LA10$	1282891	368730	352	8448	1282070	368894	370	8452
$LA11$	975384	208753	289	12777	973523	208728	308	12783
$LA12$	992296	236694	361	10710	992365	236617	380	10710
$LA13$	987090	218394	323	12052	987072	218393	329	12052
$LA14$	986491	219861	323	13589	986491	219861	335	13589
$LA15$	1000344	222394	312	12880	1000168	222140	331	12881
$LA16$	1202828	697667	340	7295	1203199	698013	629	7297
$LA17$	1217144	700369	377	6492	1220007	702864	1164	6502
$LA18$	1204168	695743	345	6951	1203711	696228	745	6966
$LA19$	1177065	677836	354	7086	1178363	680959	772	7096
$LA20$	1190795	696516	363	7202	1188553	695720	749	7218

6.3.2. Cálculo de Cotas Superiores

El método de cálculo de cotas superiores (UB) empleado en esta sección es básicamente el mismo que se ha utilizado para el problema $J||C_{max}$, descrito en la sección 6.2.1 de este capítulo. Como vimos en dicha sección, el cálculo de cotas superiores, guiado por el heurístico h_3, produce cotas no muy lejanas a las soluciones óptimas del problema $J||C_{max}$. Sin embargo, esto no ocurre para la

versión del problema $J||\sum C_i$, empleando para la construcción de las cotas el heurístico h_2. Para esta versión del problema se resuelven los problemas con el algoritmo A^* utilizando la técnica de poda, sin límite de tiempo, y calculando cotas superiores, con una probabilidad proporcional a la profundidad, para todos los nodos expandidos.

En primer lugar hemos estudiado la evolución de las cotas para los problemas que sí se resuelven con el algoritmo A^* ($ORB04-9\times9$, $ORB08-9\times9$, $LA38-9\times9$, $ABZ7\_10\times10$, $ABZ9\_10\times10$ y $LA38\_10\times10$). Las Figura 6.4 muestra la evolución de las cotas superiores para estos problemas y la Figura 6.5 muestra la evolución más detallada para uno de ellos, el $ORB08-9\times9$. Como se observa la cota superior de partida está muy lejos de la solución óptima, mucho más lejos de lo que estaba en la versión del problema $J||C_{max}$, y su evolución también. Además, el intervalo $[LB, UB]$ en el que se va a estar moviendo el algoritmo A^*, es también mucho más grande. Por otro lado, a pesar de que las cotas superiores (UB) producen podas del espacio de búsqueda, la reducción en el número de nodos generados y expandidos es pequeña (ver Tabla 6.9), lo cual es un indicio de que las cotas superiores calculadas son malas.

Las Figura 6.6 recoge la evolución de las cotas superiores para los problemas para los que desconocemos la solución óptima ($LA06-20$) y la Figura 6.7 muestra la evolución más detallada para uno de ellos, el problema $LA17$. Como se observa, en estos problemas la evolución de las cotas superiores aún es peor, la cota de partida es mala y el intervalo en que se mueve el algoritmo A^* muy grande. Además si nos fijamos en la Tabla 6.10, para los problemas $LA06-15$ no se realiza ninguna poda por cota superior, lo que es claro signo de que las cotas son malísimas. Para los problemas $LA16-20$, se realizan algunas podas por cota superior, por lo que las cotas para estos problemas son un poco mejores. Además, la máxima profundidad alcanzada es la misma, con y sin el cálculo de cotas superiores, para todos los problemas excepto uno. Esto es un indicio de que los problemas son difíciles, que las cotas superiores calculadas son malas y que están lejos de la solución óptima, pues si estuviesen cerca el número de podas por cota superior sería mayor.

Tabla 6.9: Problema $J||\sum C_i$. A^* con la técnica de poda, con y sin el cálculo de cotas superiores. Para cada problema se muestra su solución, los nodos generados y expandidos y las podas realizadas por UB

		Sin UB			Con UB			
Inst.	Opt.	Gen.	Exp.	T.(s)	Gen.	Exp.	T.(s)	Podas UB
$SORB04-9\times9$	6661	1132648	660501	273	1131824	660014	824	108197
$SORB08-9\times9$	5668	1765913	909440	772	1756203	904188	1079	480279
$ABZ8-9\times9$	3078	927825	535069	286	924376	533112	646	249295
$ABZ7-10\times10$	3262	552246	326967	160	552238	326959	517	66347
$ABZ9-10\times10$	3483	1050968	581996	382	1050943	581972	1065	96714
$LA38-10\times10$	7113	937039	536287	265	933967	534489	745	220079
$LA01$	4832	248935	107955	35	248928	107948	86	68141
$LA02$	4459	520356	217751	80	520131	217653	154	188138
$LA03$	4151	79883	32118	10	79880	32115	27	14064
$LA04$	4259	136928	57558	19	136919	57549	50	23058
$LA05$	4072	412277	174320	68	412087	174236	173	75361

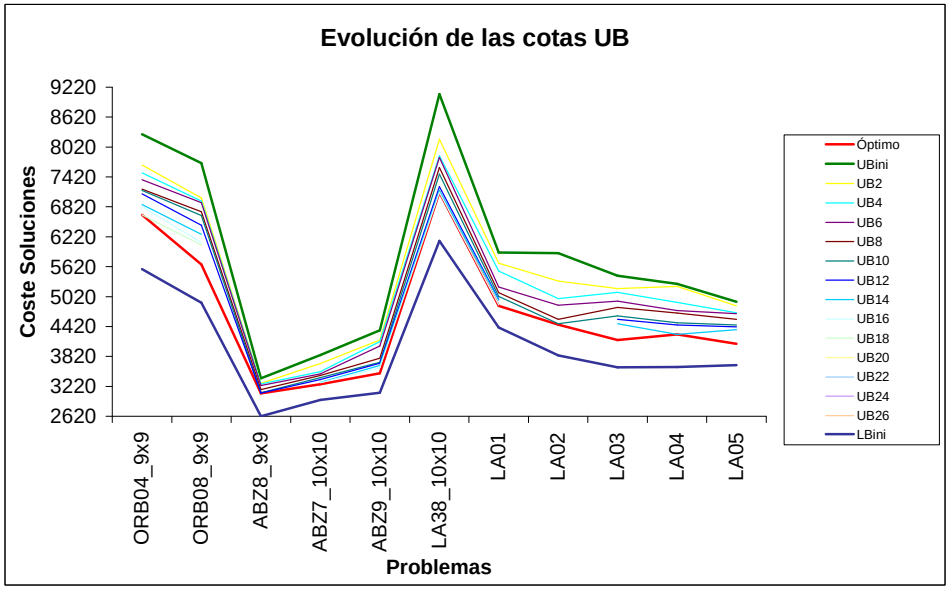


Figura 6.4: Problema $J||\sum C_i$. Gráfica que muestra la evolución de las cotas superiores calculadas con probabilidad proporcional a la profundidad

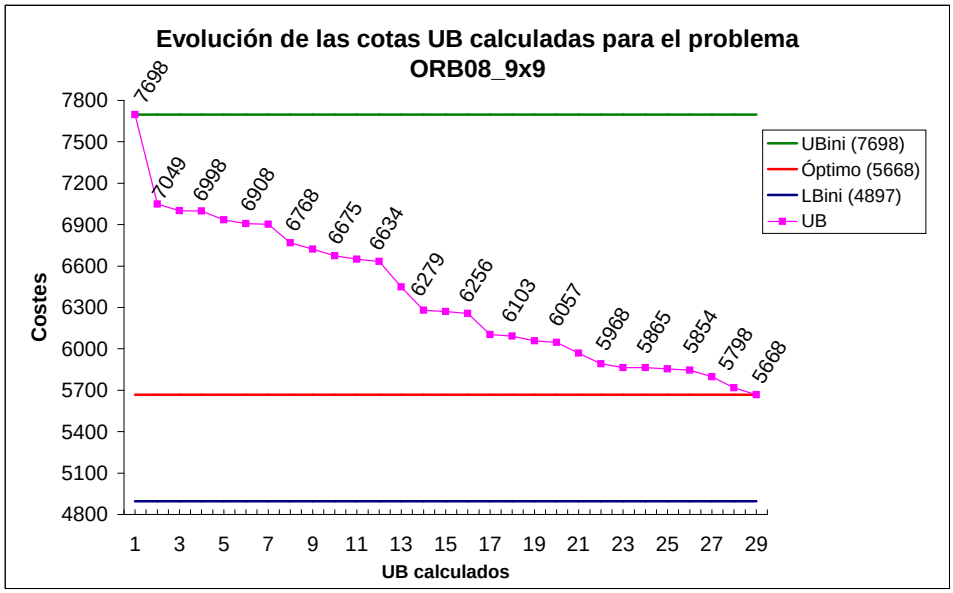


Figura 6.5: Problema $J||\sum C_i$. Gráfica que muestra la evolución de las cotas superiores calculadas, con probabilidad proporcional a la profundidad, para el problema $ORB08-9\times 9$

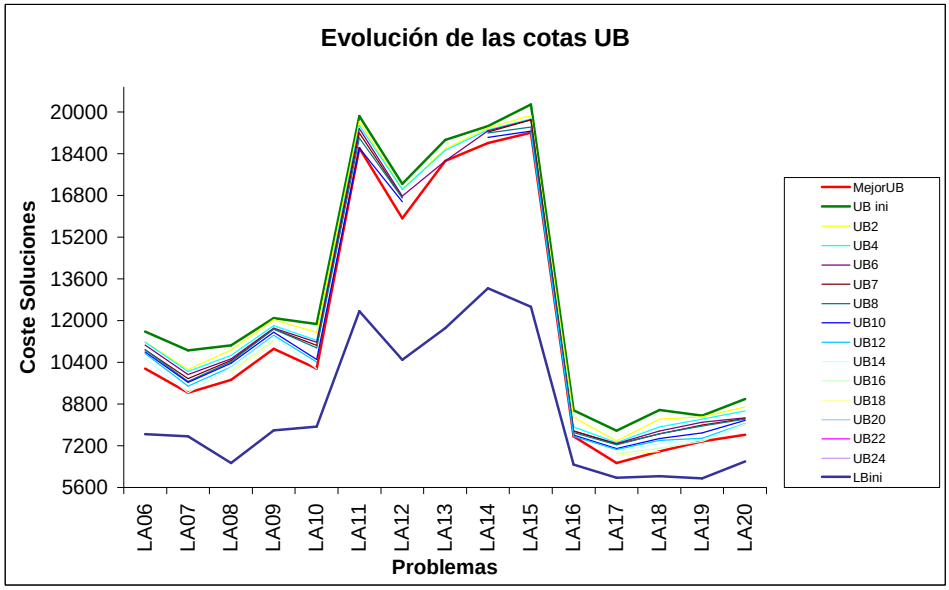


Figura 6.6: Problema $J||\sum C_i$. Gráfica que muestra la evolución de las cotas superiores calculadas con probabilidad proporcional a la profundidad

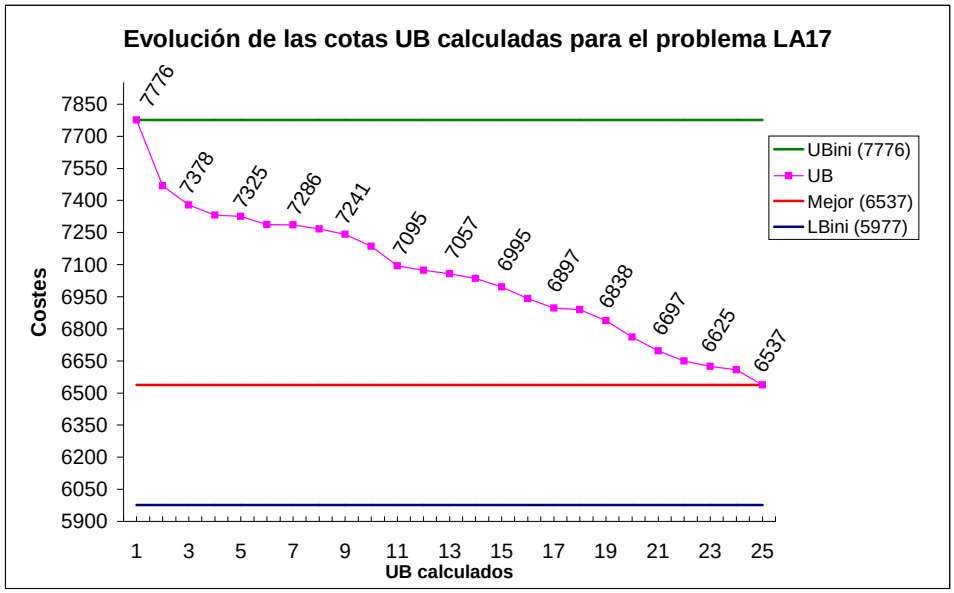


Figura 6.7: Problema $J||\sum C_i$. Gráfica que muestra la evolución de las cotas superiores calculadas, con probabilidad proporcional a la profundidad, para el problema $LA17$

Tabla 6.10: Problema $J||\sum C_i$. A^* con la técnica de poda, con y sin el cálculo de UB. Para cada problema se muestra los nodos generados y expandidos, la máxima profundidad alcanzada, el mejor UB, y las podas realizadas por cota superior

	Sin UB				Con UB					
Inst.	Gen.	Exp.	T.(s)	MaxP.	Gen.	Exp.	T.(s)	MaxP.	UB	Podas UB
$LA06$	1334365	402170	437	25	1338827	403539	1765	25	10162	0
$LA07$	1344617	390813	420	27	1345543	391073	1613	27	9237	0
$LA08$	1356179	401921	398	26	1356179	401921	1857	26	9725	0
$LA09$	1348941	406063	428	26	1349400	406211	1975	26	10913	0
$LA10$	1282891	368730	352	29	1301744	374249	1420	29	10165	0
$LA11$	975384	208753	289	19	975384	208753	1489	19	18614	0
$LA12$	992296	236694	361	24	992296	236694	1984	24	15918	0
$LA13$	987090	218394	323	21	987090	218394	1695	21	18123	0
$LA14$	986491	219861	323	22	986491	219861	1658	22	18812	0
$LA15$	1000344	222394	312	21	1000344	222394	1715	21	19201	0
$LA16$	1202828	697667	340	75	1221113	708391	1554	75	7556	16724
$LA17$	1217144	700369	377	76	1461152	840372	1528	78	6537	250491
$LA18$	1204168	695743	345	84	1374643	795489	1422	84	6984	206987
$LA19$	1177065	677836	354	69	1192678	686635	1608	69	7367	14530
$LA20$	1190795	696516	363	65	1192873	697728	1953	65	7623	1898

6.3.3. Reducción del Espacio de Búsqueda con el Parámetro δ en la Generación de Sucesores

Como vimos en el capítulo 5, el espacio de búsqueda contemplado para esta versión del problema también es el formado por las planificaciones activas. Los resultados experimentales de dicho capítulo pusieron de manifiesto la dificultad de la versión del problema $J||\sum C_i$ frente a la versión $J||C_{max}$, por tanto puede ser que la reducción del espacio de búsqueda con $\delta = 0{,}5$ no sea la adecuada para esta versión del problema. Por ello, se realizaron experimentos contemplando para δ los valores $[0{,}75, 0{,}5, 0{,}25, 0]$, es decir de poca reducción $\delta = 0{,}75$ a reducción total $\delta = 0$ (planificaciones densas). El objetivo de estos experimentos es encontrar alguna solución, aunque no sea la óptima, por tanto no se pone límite de tiempo. Además, para ver el efecto de la técnica de poda con la reducción del espacio de búsqueda realizaremos los experimentos empleando la técnica de poda y sin ella.

Los resultados de esta experimentación se recogen en las Tablas 6.11 (sin poda) y 6.12 (con poda). En ellas, para cada problema se muestra la solución óptima (si se conoce), los nodos generados y expandidos, el tiempo de ejecución y la mejor solución o cota LB alcanzada en ese espacio de búsqueda. En los problemas resueltos se marca en verde la solución cuando es óptima y en azul si no los es (recordemos que para los problemas $LA06-20$ desconocemos cual es la solución óptima). Para los problemas no resueltos se muestra, el número de nodos expandidos y el tiempo empleados en la búsqueda de una solución. Como se observa a medida que la reducción se hace mayor se comienzan a resolver problemas que no se resolvían, pero la calidad de las soluciones empeora. Con todas las reducciones, el número de problemas resueltos empleando la técnica de poda es mayor. Cuando los problemas se resuelven de ambas maneras (con y sin técnica de poda) el número de nodos expandidos y el tiempo de ejecución es mucho menor empleando la técnica de poda. Sin embargo, para unos pocos problemas, cuando la reducción es muy agresiva ($\delta = 0{,}25$ y $\delta = 0$), la solución alcanzada sin la técnica de poda es un poco mejor. Para que un nodo n pode a otro n' es necesario que $f^*(n) <= f^*(n')$, esto garantiza que la solución que podemos alcanzar por n es mejor o igual que la que podemos alcanzar por n'. Sin embargo, puede ocurrir que al reducir el espacio de búsqueda estemos dejando fuera la mejor solución a la que podemos llegar por n, por lo que puede que estemos podando un nodo n' que nos lleve a una mejor solución, de las que quedan en el espacio de búsqueda, que n. A pesar de esto, el ahorro de memoria y tiempo que logra la técnica de poda compensa a la poca calidad que se pierde en las soluciones.

A la vista de los resultados experimentales parece que la mejor combinación para esta versión del problema $J||\sum C_i$, es la misma que para la versión $J||C_{max}$: el empleo de la técnica de poda con una reducción del espacio de búsqueda con $\delta = 0{,}5$. Esta reducción del espacio de búsqueda hace que podamos ofrecer una solución para los problemas $LA15-20$, no resueltos sin reducción, pero deja fuera las soluciones óptimas de problemas que antes eran resueltos óptimamente.

Tabla 6.11: Problema $J||\sum C_i$. A^* sin la técnica de poda y con reducciones del espacio de búsqueda $\delta \in [0{,}75, 0{,}5, 0{,}25, 0]$

No Poda															
		$\delta = 0{,}75$			$\delta = 0{,}5$			$\delta = 0{,}25$			$\delta = 0$		
	Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$SORB04 - 9 \times 9$	6661	2407214	710		3114733	789		3069581	717	6759	1437818	326	6792	
$SORB08 - 9 \times 9$	5668	1478029	509		1938226	590		2516450	698		2766024	749		
$ABZ8 - 9 \times 9$	3078	2213432	588		2011027	481	3085	526673	114	3107	125244	26	3139	
$ABZ7 - 10 \times 10$	3262	1621456	505	3262	796810	231	3262	280531	74	3285	64327	16	3285	
$ABZ9 - 10 \times 10$	3483	1573011	537		1692465	533	3514	805520	220	3605	204459	50	3688	
$LA38 - 10 \times 10$	7113	1237346	409	7113	1141615	322	7359	395008	103	7385	73953	17	7599	
$LA01$	4832	292254	76	4833	210506	51	4833	95689	22	4837	82458	18	4890	
$LA02$	4459	1706389	439		1043581	245	4480	288278	64	4498	152151	31	4515	
$LA03$	4151	85941	23	4151	59387	16	4151	27303	7	4161	31631	8	4227	
$LA04$	4259	161878	42	4259	96017	23	4303	145348	33	4390	34258	7	4390	
$LA05$	4072	1624911	434		2002930	490		1634465	365	4119	1433118	289	4194	
$LA06$		547896	310		638020	331		782766	364		940656	411		
$LA07$		493898	300		545427	311		633080	330		728282	355		
$LA08$		511788	303		583787	319		655905	334		719172	349		
$LA09$		561737	313		627451	332		727465	353		820386	379		
$LA10$		516926	308		580116	323		656594	345		715759	354		
$LA11$		243105	279		261195	288		275030	292		291402	299		
$LA12$		323999	303		348836	314		349811	315		365704	317		
$LA13$		268531	287		281048	294		302978	297		332551	310		
$LA14$		268424	287		293329	296		324794	306		342216	318		
$LA15$		249720	279		262870	289		287284	294		317768	304		
$LA16$		1657333	575		2132058	665		2703206	743	7514	870601	230	7556	
$LA17$		1821246	578		2179285	631	6537	516266	139	6566	36080	9	6634	
$LA18$		1620620	545		1991254	627		1504780	418	7074	428290	113	7194	
$LA19$		1818240	571		2141132	636		309710	85	7268	193942	46	7525	
$LA20$		1857406	600		2439570	707		197830	55	7388	78868	19	7804	

Tabla 6.12: Problema $J||\sum C_i$. A^* con la técnica de poda y con reducciones del espacio de búsqueda $\delta \in [0{,}75, 0{,}5, 0{,}25, 0]$

Poda		$\delta = 0{,}75$			$\delta = 0{,}5$			$\delta = 0{,}25$			$\delta = 0$		
Ins.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$SORB04 - 9 \times 9$	6661	591500	222	6679	428098	142	6688	224214	68	6759	226164	64	6875
$SORB08 - 9 \times 9$	5668	478270	206	5668	246793	91	5668	168799	58	5750	162440	51	5945
$ABZ8 - 9 \times 9$	3078	399352	171	3078	184265	60	3085	49628	13	3107	13252	3	3139
$ABZ7 - 10 \times 10$	3262	199857	82	3262	113719	41	3262	48457	16	3285	27886	9	3326
$ABZ9 - 10 \times 10$	3483	312600	149	3483	215039	88	3514	129195	45	3611	33563	10	3694
$LA38 - 10 \times 10$	7113	118369	48	7113	101199	35	7359	40773	13	7385	12542	3	7602
$LA01$	4832	70800	21	4833	55484	16	4833	27200	7	4837	15435	4	4890
$LA02$	4459	159547	53	4459	61332	17	4480	21018	6	4498	13319	4	4515
$LA03$	4151	21097	7	4151	15106	4	4151	11999	4	4181	9701	2	4227
$LA04$	4259	29699	9	4259	15261	5	4303	18356	5	4390	9793	2	4390
$LA05$	4072	140182	49	4072	111993	34	4103	59264	16	4119	33163	8	4194
$LA06$		432118	454		488648	469		558167	484		606779	484	
$LA07$		418214	443		456387	459		502938	474		545514	466	
$LA08$		428167	416		457729	423		482971	406		502065	400	
$LA09$		430978	450		470759	479		515314	515		580187	557	
$LA10$		396959	396		434855	390		480693	421		528316	462	
$LA11$		217752	295		231620	302		249549	307		263414	310	
$LA12$		248861	371		265101	372		273791	363		286466	347	
$LA13$		228545	338		238057	332		255987	346		276844	367	
$LA14$		234847	351		250644	351		271447	359		283325	372	
$LA15$		226148	316		234595	328		246041	318		256439	319	
$LA16$		737697	335		552894	222	7470	220740	76	7519	98907	31	7556
$LA17$		550911	247	6537	168853	62	6537	76063	26	6566	17344	6	6634
$LA18$		452478	189	6977	276933	102	7055	131057	44	7111	46262	14	7194
$LA19$		735875	357		310135	117	7248	101056	34	7296	40294	12	7525
$LA20$		762375	368		492804	190	7366	39300	12	7388	39300	13	7388

6.3.4. Ponderación de la Función Heurística

Como vimos en el apartado 6.2.3 de este capítulo, otra de las técnicas más empleadas en la relajación de la optimalidad, es la ponderación de la función heurística. Por tanto, al igual que se hizo para la versión del problema $J||C_{max}$, en esta versión del problema $J||\sum C_i$ emplearemos ponderación estática. Los resultados de los experimentos realizados en esta sección se recogen en una serie de tablas en las que para cada problema se muestra la solución óptima (si se conoce), el número de nodos expandidos, el tiempo de ejecución y la solución alcanzada.

En el primer grupo de experimentos resolvemos los problemas con el A^*, utilizando ponderaciones [1,25; 1,5; 1,75; 2], con y sin la técnica de poda, y sin límite de tiempo. Las Tablas 6.13 (sin poda) y 6.14 (con poda) muestran los resultados de estos experimentos. En ellas se observa el mismo comportamiento que para la versión del problema $J||C_{max}$, a menor ponderación mayor calidad de las soluciones y menor tiempo para alcanzar la primera solución. La primera solución alcanzada es la misma, para todos los problemas resueltos con y sin la técnica de poda. Sin embargo, con poda se encuentra solución para más problemas, expandiendo menor número de nodos y por tanto en menor tiempo. Este efecto se ve claramente con $w = 1{,}5$ y $w = 1{,}25$.

Los resultados de estos experimentos ponen de manifiesto que merece la pena combinar la técnica de poda con la ponderación estática. Por tanto, resolveremos ahora los problemas con una versión *"Anytime"* del algoritmo, similar a la empleada para la versión del problema $J||C_{max}$ en la sección 6.2.3, utilizando la técnica de poda junto con la ponderación estática. Este algoritmo nos permitirá calcular todas las soluciones hasta que se agote la memoria disponible. Con estos experimentos trataremos de ver hasta donde es capaz de llegar la ponderación estática combinada con la técnica de poda, y cual es la mejor de las ponderaciones. Los resultados de estos experimentos se pueden ver en la Tabla 6.15, en la que se muestran en verde las soluciones ponderadas encontradas que son óptimas. A medida que la ponderación es más pequeña la calidad de las soluciones es más alta, siendo el tiempo en que se agota la memoria menor, pues requiere expandir más nodos.

A la vista de estos resultados, la mejor ponderación, en general, para este grupo de problemas parece que se encuentra entre $w = 1{,}5$ y $w = 1{,}25$. Con $w = 1{,}25$ las soluciones alcanzadas son mejores y se resuelven óptimamente 9 problemas, pero quedan sin resolver 5 problemas. Con $w = 1{,}5$ se alcanza solución para todos los problemas, resolviendo óptimamente sólo 2 y siendo las soluciones en el resto de los casos peores que las obtenidas con $w = 1{,}25$. Sin embargo, si miramos los problemas por grupos, para los problemas $LA01 - 05$ (10×5), los problemas recortados $(9 \times 9$ y $10 \times 10)$ y los problemas $LA16 - 20$ (10×10) la mejor ponderación estática es claramente $w = 1{,}25$, se resuelven todos y las soluciones son mejores. Para el grupo de problemas $LA06 - 10$, con ponderación 1,25 se resuelven todos menos 1, y para los $LA11 - 15$ sólo se resuelve 1 por tanto, para estos problemas estaríamos entre una ponderación estática de 1,5 y 1,25.

A la vista de los resultados anteriores, planteamos ahora un método que nos permitirá, de forma sistemática, determinar la mejor ponderación estática para cada problema. Con esto tratamos de, a partir de $w = 1{,}5$ (ponderación para la que se encuentra solución a todos los problemas), mejorar la calidad de las soluciones encontradas para los problemas $LA06 - 20$, problemas no resueltos con el A^* empleando la técnica de poda. Para ello, modificamos el algoritmo A^*, empleando la técnica

de poda, para que resuelva un problema con varias ponderaciones. Comenzará resolviéndolo con ponderación $w = 1{,}5$, e irá decrementando la ponderación en $0,05$ hasta que se encuentre una ponderación para la que el problema ya no se resuelva o hasta llegar a $w = 1{,}05$, ya que con $w = 1$ tendríamos el algoritmo A^* sin ponderación, con el que sabemos que estos problemas no son resueltos. Con esto obtendríamos una primera solución para estos problemas con una ponderación estática más ajustada. Los resultados de esta primera solución calculada se recogen en la parte izquierda(*Primera Solución*) de la Tabla 6.16. En ella, para cada problema se muestra la última ponderación para la que se encontró solución, el número de ejecuciones y el tiempo empleado hasta llegar a una ponderación para la que no se alcanza solución o a $w = 1{,}05$, y la solución alcanzada con la última de las ponderaciones con la que se resuelve el problema. Como se observa, la calidad de las soluciones es mejor que las obtenidas con $w = 1{,}5$ y $w = 1{,}25$ (Tabla 6.14), para todos los problemas excepto el $LA06$. Este método puede mejorarse, calculando con la última ponderación para la que se resuelve el problema, todas las soluciones posibles hasta agotar los recursos. Los resultados de estos experimentos se recogen en la parte derecha (*Varias Soluciones*) de la Tabla 6.16. Ahora, todas las soluciones son mejores o iguales que las calculadas con $w = 1{,}5$ y $w = 1{,}25$ (Tabla 6.15). De estos resultados se desprende que el método propuesto parece un buen método para adaptar la ponderación estática a cada problema.

Tabla 6.13: Problema $J||\sum C_i$. A^* sin la técnica de poda y con ponderación estática. El algoritmo se detiene cuando encuentra la primera solución

No Poda													
		$w = 2$			$w = 1{,}75$			$w = 1{,}5$			$w = 1{,}25$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04 - 9 \times 9$	6661	155	0	7309	165	0	6944	219	0	6968	761	0	6808
$ORB08 - 9 \times 9$	5668	122	0	6336	131	0	6464	187	0	6216	9088	4	5829
$ABZ8 - 9 \times 9$	3078	84	0	3318	86	0	3332	109	0	3324	562	0	3160
$ABZ7 - 10 \times 10$	3262	101	1	3492	101	0	3515	105	0	3285	107	0	3285
$ABZ9 - 10 \times 10$	3483	107	0	3762	106	0	3809	111	0	3809	3540	2	3678
$LA38 - 10 \times 10$	7113	108	0	7790	108	0	7635	167	0	7589	3360	1	7506
$LA01$	4832	56	0	5391	54	0	5569	57	0	5376	91	0	5126
$LA02$	4459	51	0	4706	51	0	4678	69	0	4774	1662	1	4546
$LA03$	4151	53	0	4465	55	0	4465	60	0	4399	117	0	4162
$LA04$	4259	62	0	5115	75	0	4988	102	0	4744	1768	0	4293
$LA05$	4072	52	0	4607	52	0	4410	52	0	4385	1479	1	4253
$LA06$		110	0	9848	120	0	9866	200	0	9802	484208	298	
$LA07$		91	0	9454	86	0	9322	105	1	9444	25808	15	8915
$LA08$		117	0	9714	287	0	9465	5263	3	9047	468817	290	
$LA09$		83	0	10623	104	0	10529	266	0	10642	480003	290	
$LA10$		110	0	10834	89	0	10190	127	0	10284	664	1	9420
$LA11$		187	1	17857	420	1	17363	230850	266		224200	275	
$LA12$		127	0	14762	592	0	14501	2695	2	13964	292121	273	
$LA13$		136	0	16606	166	0	16395	1801	2	15517	247103	278	
$LA14$		140	0	17997	158	1	17455	463	0	16862	232956	272	
$LA15$		155	0	17691	232	0	17808	2389	3	16906	241221	275	
$LA16$		115	0	7918	123	0	7796	215	0	7940	2075	0	7416
$LA17$		102	0	6957	102	0	6892	108	0	7018	364	1	6865
$LA18$		106	0	7485	103	0	7339	103	0	7188	398	0	7201
$LA19$		113	0	7949	119	0	7654	152	0	7731	828	0	7398
$LA20$		102	0	8157	112	0	7871	101	0	7701	111	0	7586

Tabla 6.14: Problema $J||\sum C_i$. A^* con la técnica de poda y con ponderación estática. El algoritmo se detiene cuando encuentra la primera solución

Poda		$w = 2$			$w = 1{,}75$			$w = 1{,}5$			$w = 1{,}25$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04 - 9 \times 9$	6661	153	1	7309	162	0	6944	194	0	6968	577	0	6808
$ORB08 - 9 \times 9$	5668	120	0	6336	129	0	6464	175	0	6216	1647	1	5829
$ABZ8 - 9 \times 9$	3078	84	0	3318	86	0	3332	109	0	3324	416	0	3160
$ABZ7 - 10 \times 10$	3262	101	0	3492	101	0	3515	105	0	3285	107	0	3285
$ABZ9 - 10 \times 10$	3483	107	0	3762	106	0	3809	111	0	3809	1750	1	3678
$LA38 - 10 \times 10$	7113	108	0	7790	108	0	7635	151	0	7589	2276	1	7506
$LA01$	4832	56	0	5391	54	0	5569	57	0	5376	91	0	5126
$LA02$	4459	51	0	4706	51	0	4678	69	0	4774	835	0	4546
$LA03$	4151	53	0	4465	55	0	4465	60	0	4399	86	0	4162
$LA04$	4259	62	0	5115	75	0	4988	102	0	4744	873	0	4293
$LA05$	4072	52	0	4607	52	0	4410	52	0	4385	659	1	4253
$LA06$		110	0	9848	120	0	9866	200	0	9802	109727	72	8911
$LA07$		91	0	9454	86	0	9322	105	0	9444	5199	4	8915
$LA08$		116	0	9714	280	0	9465	1356	1	9047	393886	277	
$LA09$		83	0	10623	104	0	10529	258	0	10642	184771	123	9604
$LA10$		110	0	10834	89	0	10190	127	0	10284	592	0	9420
$LA11$		187	1	17857	417	1	17363	22477	28	16627	200225	258	
$LA12$		127	0	14762	425	0	14502	1862	2	13964	169710	167	12875
$LA13$		136	0	16606	166	0	16395	1418	1	15517	211193	260	
$LA14$		140	0	17997	158	0	17455	362	1	16862	199808	253	
$LA15$		155	0	17691	231	0	17808	1456	1	16906	209769	259	
$LA16$		115	0	7918	123	0	7796	213	0	7940	1434	1	7416
$LA17$		102	0	6957	102	1	6892	108	0	7018	336	0	6865
$LA18$		106	0	7485	103	0	7339	103	1	7188	331	0	7201
$LA19$		113	0	7949	119	0	7654	152	0	7731	603	0	7398
$LA20$		102	0	8157	112	0	7871	101	0	7701	111	0	7586

Tabla 6.15: Problema $J||\sum C_i$. A^* − *Anytime* con ponderación estática y la técnica de poda. Encuentra todas las soluciones posibles hasta agotar la memoria

Poda													
		$w = 2$			$w = 1{,}75$			$w = 1{,}5$			$w = 1{,}25$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04-9\times9$	6661	1339298	393	7053	1330995	392	6864	1306909	389	6800	1121981	332	6661
$ORB08-9\times9$	5668	1337387	402	6204	1329356	401	6204	1286671	380	5986	1096113	334	5717
$ABZ8-9\times9$	3078	1341114	338	3241	1338321	340	3176	1317214	334	3167	1148686	323	3078
$ABZ7\_10\times10$	3262	1091123	323	3448	1089334	316	3448	1085015	314	3275	1018709	289	3275
$ABZ9\_10\times10$	3483	1091710	322	3648	1091210	325	3648	1087734	318	3628	1018192	301	3483
$LA38\_10\times10$	7113	1091522	322	7544	1091387	327	7516	1084602	322	7516	985897	291	7113
$LA01$	4832	2120291	522	5102	2104754	527	5102	2025786	508	5093	1786079	461	4832
$LA02$	4459	2107992	591	4501	2082034	619	4501	1981526	546	4459	1599362	446	4459
$LA03$	4151	2124205	539	4161	2108236	527	4161	2051986	537	4151	1768662	454	4151
$LA04$	4259	2122162	568	4437	2095485	581	4270	2002919	532	4270	1640572	420	4259
$LA05$	4072	2122458	563	4253	2103307	550	4253	1984897	528	4198	1567164	423	4072
$LA06$		1188608	370	9489	1184332	368	9585	1177913	364	9319	889639	315	8748
$LA07$		1187865	371	9163	1185014	372	9085	1167175	371	9010	1072494	349	8419
$LA08$		1180076	419	9048	1182013	384	8898	1094154	378	8474	393886	277	
$LA09$		1184707	375	10334	1172963	369	10088	1133178	363	9840	444190	252	9274
$LA10$		1188377	396	10106	1187322	368	9773	1165754	383	9793	1031051	343	9162
$LA11$		896953	405	17297	894109	364	16962	798939	346	16070	200225	258	
$LA12$		893996	367	14314	888440	354	14109	858486	347	13633	251692	237	12517
$LA13$		896584	368	16009	893770	368	15932	873121	355	14974	211193	260	
$LA14$		897466	334	17170	896873	356	16813	890962	343	16438	199808	253	
$LA15$		895863	392	17100	893102	368	17387	878904	361	16512	209769	259	
$LA16$		1091343	332	7787	1090733	336	7673	1084870	340	7668	978035	308	7376
$LA17$		1091258	338	6770	1089821	338	6770	1082889	318	6840	1005160	296	6591
$LA18$		1090557	329	7394	1090240	319	7133	1080060	322	7133	979340	295	6970
$LA19$		1088066	335	7720	1082170	329	7556	1061600	326	7499	966109	292	7227
$LA20$		1091886	344	7669	1087795	336	7669	1074577	315	7623	918143	288	7465

Tabla 6.16: Problema $J||\sum C_i$. A^* con ponderación estática y la técnica de poda. Soluciones para los problemas $LA06-20$, encontradas a partir de $w = 1{,}5$ con reducciones de $0,05$ hasta no encontrar solución o llegar a $w = 1{,}05$. Deteniendo el algoritmo en la primera solución y calculando varias soluciones

		Primera Solución			*Varias Soluciones*		
Inst.	w	Nº Eje.	T.(s)	Sol.	Nº Eje.	T.(s)	Sol.
$LA06$	1,2	8	484	9123	9	761	8683
$LA07$	1,15	9	317	8498	10	618	8088
$LA08$	1,3	6	442	8663	7	699	8216
$LA09$	1,25	7	394	9604	8	757	9274
$LA10$	1,1	10	462	8985	11	710	8862
$LA11$	1,4	4	438	16013	5	745	15442
$LA12$	1,25	7	494	12875	8	841	12517
$LA13$	1,3	6	547	14500	7	820	14223
$LA14$	1,3	6	325	16197	7	631	15778
$LA15$	1,35	5	338	16096	6	661	15599
$LA16$	1,05	10	190	7376	11	480	7376
$LA17$	1,05	10	99	6537	11	385	6537
$LA18$	1,05	10	117	6977	11	398	6970
$LA19$	1,1	10	332	7227	11	599	7217
$LA20$	1,1	10	388	7366	11	669	7345

6.3.5. Discrepancia Heurística Limitada

A partir de los resultados obtenidos con ponderación estática, viendo la dificultad de los problemas $LA06-15$, la intuición nos dice que, para esta versión del problema debe haber muchos nodos con igual valor de la función de evaluación f, por lo que para llegar a una solución es necesario expandir muchos nodos del mismo nivel. Para reducir el número de nodos a expandir y poder alcanzar una solución, aunque no sea óptima, vamos a emplear el método de búsqueda con discrepancia heurística limitada [52]. Este método consiste en expandir primero aquellos nodos con una discrepancia heurística de 0, luego los de 1 y así sucesivamente. Para ello, el algoritmo considera en cada expansión como máximo un número de sucesores igual a la discrepancia heurística (DH). Obviamente los mejores. De este modo se llega más rápido a una solución dándole a la búsqueda un aspecto de búsqueda en profundidad. Para estos experimentos emplearemos una versión modificada del algoritmo A^* con la técnica de poda, que permite utilizar discrepancias heurísticas limitadas. Los resultados de los experimentos realizados en esta sección se recogen en una serie de tablas en las que para cada problema se muestra la solución óptima (si se conoce), el número de nodos expandidos, el tiempo de ejecución y la solución alcanzada (en verde si es la óptima).

En primer lugar se resuelven los problemas con el algoritmo A^* y discrepancias heurísticas limitadas ($DH \in [1,2,3,4]$), sin límite de tiempo. La Tabla 6.17 muestra los resultados de estos experimentos. Como se ve a medida que la discrepancia heurística limitada se incrementa la calidad de las soluciones y el tiempo empleado para alcanzarlas es mayor. Sin embargo, el número de problemas resueltos es menor. No obstante no resulta un método muy efectivo ya que solamente es capaz de dar una solución a todos los problemas cuando la discrepancia es igual a 1. Como lo que

pretendemos es alcanzar solución para los problemas no resueltos, veamos ahora que ocurre cuando ponderamos la función heurística, con las ponderaciones estáticas que parecían mejores $w = 1{,}5$ y $w = 1{,}2$, con las discrepancias heurísticas limitadas $DH = 1$ y $DH = 2$, discrepancias con las que se resuelve un mayor número de problemas. Utilizaremos una versión *"Anytime"* del algoritmo que nos permite calcular todas las soluciones hasta agotar la memoria disponible. Los resultados de estos experimentos se recogen en la Tabla 6.18. Con $DH = 1$ ambas ponderaciones alcanzan las mismas soluciones, pero son soluciones peores que las que se obtienen con $DH = 2$. Con $DH = 2$ las soluciones resultantes de ponderar con $w = 1{,}25$ son mejores que las que se obtienen ponderando con $w = 1{,}5$, pero no se resuelve la instancia $LA12$. Para los problemas resueltos tanto con ponderación como sin ella y con $DH = 2$, la calidad de las soluciones es mejor sin ponderación, pero quedan más problemas sin resolver. Podemos decir, a la vista de estos resultados, que el empleo de discrepancias heurísticas limitadas no parece una buena opción. Este hecho constituye una evidencia más de la falta de precisión del heurístico.

Tabla 6.17: Problema $J||\sum C_i$. A^* con la técnica de poda y discrepancias heurísticas limitadas. El algoritmo se detiene cuando encuentra la primera solución

		Poda											
		$DH = 1$			$DH = 2$			$DH = 3$			$DH = 4$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04 - 9 \times 9$	6661	82	0	8323	38537	12	7001	238996	85	6922	188793	67	6710
$ORB08 - 9 \times 9$	5668	82	0	7707	201067	70	5962	804220	461		801754	466	
$ABZ8 - 9 \times 9$	3078	82	0	3841	167663	55	3110	487435	243	3078	533293	285	3078
$ABZ7\_10 \times 10$	3262	101	0	4531	488500	240	3385	184837	81	3262	321216	156	3262
$ABZ9\_10 \times 10$	3483	101	0	4157	218511	89	3511	552735	340	3483	581503	381	3483
$LA38\_10 \times 10$	7113	101	1	9079	709221	310	7302	705674	362		536320	263	7113
$LA01$	4832	51	0	5898	22774	6	5015	67965	21	4832	107804	35	4832
$LA02$	4459	51	0	5328	26816	7	4509	129037	42	4480	196826	70	4459
$LA03$	4151	51	0	5850	1482	0	4611	3895	1	4182	10313	3	4151
$LA04$	4259	51	0	5442	4175	1	4425	23112	7	4369	66000	20	4318
$LA05$	4072	51	0	5172	15479	4	4203	120400	41	4072	173785	67	4072
$LA06$		76	0	10656	717176	465		507931	449		428404	438	
$LA07$		76	0	9678	628276	299	8619	535421	421		429394	412	
$LA08$		76	0	10382	666988	419		488597	401		425014	383	
$LA09$		76	0	10986	665820	400		477874	435		419340	437	
$LA10$		76	0	11612	677553	418		478815	410		401399	367	
$LA11$		101	0	17921	511709	348		356331	322		285008	313	
$LA12$		101	0	15883	518988	352		348573	335		304650	398	
$LA13$		101	0	16371	505117	348		356410	352		283399	350	
$LA14$		101	0	17368	504682	341		367088	384		273186	344	
$LA15$		101	0	17704	522578	335		367079	322		296663	328	
$LA16$		101	0	9526	208179	79	7543	716215	343		701618	340	
$LA17$		101	0	7974	324842	129	6548	710551	380		701236	377	
$LA18$		101	0	8889	181513	68	7247	708822	344		696333	344	
$LA19$		101	0	8534	745663	341		686108	353		678225	349	
$LA20$		101	0	8785	768740	361		702822	365		696658	363	

Tabla 6.18: Problema $J||\sum C_i$. A^* − *Anytime* con la técnica de poda, discrepancias heurísticas limitadas y ponderación estática. Encuentra todas las soluciones posibles hasta agotar la memoria

		Poda											
		$DH = 1 \vert w = 1{,}5$			$DH = 1 \vert w = 1{,}25$			$DH = 2 \vert w = 1{,}5$			$DH = 2 \vert w = 1{,}25$		
	Inst. Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04 - 9 \times 9$	6661	82	1	8323	82	0	8323	1333664	370	7079	1228351	382	7001
$ORB08 - 9 \times 9$	5668	82	0	7707	82	0	7707	1321221	357	6094	1158570	322	5962
$ABZ8 - 9 \times 9$	3078	82	0	3841	82	0	3841	1328250	324	3181	1207619	309	3112
$ABZ7\_10 \times 10$	3262	101	0	4531	101	0	4531	1086294	304	3486	998021	280	3385
$ABZ9\_10 \times 10$	3483	101	0	4157	101	0	4157	1083309	313	3695	1032136	307	3519
$LA38\_10 \times 10$	7113	101	0	9079	101	0	9079	1083584	311	7374	992607	292	7319
$LA01$	4832	51	0	5898	51	0	5898	2077029	490	5091	1929377	463	5015
$LA02$	4459	51	0	5328	51	0	5328	2075304	540	4509	1950318	486	4509
$LA03$	4151	51	0	5850	51	0	5850	2124013	500	4611	2005913	475	4611
$LA04$	4259	51	0	5442	51	0	5442	2120803	493	4425	1938922	457	4425
$LA05$	4072	51	0	5172	51	0	5172	2111864	524	4203	2047605	490	4203
$LA06$		76	0	10656	76	0	10656	1183031	345	9611	1027886	312	9132
$LA07$		76	0	9678	76	0	9678	1187642	360	8869	1096840	336	8619
$LA08$		76	0	10382	76	0	10382	1170937	352	8703	883510	301	8533
$LA09$		76	0	10986	76	0	10986	1154749	358	10057	956825	319	9349
$LA10$		76	0	11612	76	0	11612	1180135	403	10175	1014936	326	9450
$LA11$		101	0	17921	101	0	17921	892745	331	15926	703438	307	15152
$LA12$		101	0	15883	101	0	15883	879410	337	13936	504703	303	
$LA13$		101	0	16371	101	0	16371	892708	342	15098	784158	326	14257
$LA14$		101	0	17368	101	0	17368	895306	320	17091	791084	307	16208
$LA15$		101	0	17704	101	0	17704	896420	340	16120	867711	330	15279
$LA16$		101	0	9526	101	0	9526	1077726	329	7877	1001531	304	7543
$LA17$		101	0	7974	101	0	7974	1088959	307	6608	1038612	298	6591
$LA18$		101	0	8889	101	0	8889	1086213	330	7455	1001378	296	7247
$LA19$		101	0	8534	101	0	8534	1072187	310	7629	981075	298	7264
$LA20$		101	0	8785	101	0	8785	1084655	341	7757	928225	308	7460

6.3.6. Ponderación Adaptativa al Dominio

En la sección 6.3.4, hemos visto cómo calcular soluciones, en condiciones de no optimalidad, empleando la ponderación estática de la función heurística. Para los problemas no resueltos óptimamente, el valor del parámetro de la ponderación estática que mejor se comporta depende de las características del problema, siendo los problemas más difíciles los $LA06-20$ y de ellos los $LA06-15$. Estos resultados nos han hecho pensar en una ponderación, similar a estas, pero que tenga en cuenta información sobre el dominio del problema. Para ello hemos construido un heurístico no admisible h_w, basado en los heurísticos h_1 y h_2. Recordemos el cálculo de los heurísticos h_1 y h_2:

$$\begin{gathered} h_1(n) = F_1(P(n)) + \textstyle\sum_{J_i \in J_{SC}(n)}(r_{\theta_{iM}} + p_{\theta_{iM}}) - g(n); \\ F_1(P(n)) = \textstyle\sum_{J_i \in J_m \cup J_{\overline{m}}}(r_{\theta_{iM}} + p_{\theta_{iM}}) \\ h_2(n) = F_2(P(n)) + \textstyle\sum_{J_i \in J_{SC}(n)}(r_{\theta_{iM}} + p_{\theta_{iM}}) - g(n); \end{gathered} \tag{6.2}$$

$$F_2(P(n)) = \text{máx}_{m \in R}\{\text{máx}(\textstyle\sum_{J_i \in J_m}(C_{\theta_i{}^m} + q_{\theta_i{}^m}), \sum_{J_i \in J_m}(r_{\theta_{iM}} + p_{\theta_{iM}})) \ + \sum_{J_i \in J_{\overline{m}}}(r_{\theta_{iM}} + p_{\theta_{iM}})\}. \tag{6.3}$$

donde ${\theta_i}^m$ es la tarea del trabajo J_i que requiere la máquina m y $C_{\theta_i{}^m}$ su tiempo de completud en la planificación de Jackson (JPS) modificada, empleada en el heurístico h_2 (sección 5.3.2), en la que las prioridades se asignan en relación inversa a los tiempos de procesamiento remanente.

El heurístico h_2 está mejor informado que el heurístico h_1, en realidad el cálculo de h_2 se puede expresar como:

$$\begin{gathered} h_2(n) = h_1(n) + \text{máx}_{m \in R}\{\Delta_m\}, con \\ \Delta_m = \text{máx}(0, \textstyle\sum_{J_i \in J_m}(C_{\theta_i{}^m} + q_{\theta_i{}^m}) - \sum_{J_i \in J_m}(r_{\theta_{iM}} + p_{\theta_{iM}})). \end{gathered} \tag{6.4}$$

Dado que la estimación que hace el heurístico h_2 está lejos del valor de h^*, tal y como se ha puesto de manifiesto en muchos de los experimentos anteriores, lo que proponemos aquí es utilizar una función alternativa al máximo para estimar la contribución de los factores Δ_m al valor del heurístico. Evidentemente si esta función toma valores mayores que el máximo, la estimación del heurístico puede sobrepasar el valor de h^*. En principio hemos utilizado la suma y los resultados experimentales han demostrado que en este caso se sobrepasa ampliamente el valor de h^*. Por lo que en una segunda aproximación hemos considerado la posibilidad de ponderar la contribución de los distintos factores. La propuesta concreta consiste en utilizar funciones de la forma:

$$F_w(n) = \Delta_1 + \textstyle\sum_{2 \leq i \leq M} \Delta_i / 2^{w_i + \delta}, w_i > 0 \tag{6.5}$$

donde los Δ_i están ordenados de mayor a menor valor y los w_i han de ser valores crecientes. En principio proponemos $w_i = (i-1)$, con $2 \leq i \leq M$.

De este modo, al contemplar el mayor Δ completo (Δ_1) y dividir el resto, estamos ajustando la estimación de h_w a un valor más próximo a h_2. Además, cuando $\delta = 0$ es similar a ponderar la función heurística h_2 con una ponderación estática $w \leq 2$ y cuando $\delta = 1$ similar a ponderarla con ponderación estática $w \leq 1{,}5$.

La Tabla 6.19 muestra los resultados de aplicar este método de ponderación con los valores $\delta = 0$ y $\delta = 1$ y $w_i = (i-1)$, con $2 \leq i \leq M$. En dicha tabla para cada problema se muestra el número de nodos expandidos, el tiempo de ejecución y la solución alcanzada (en verde si es la óptima). Como se observa, para los problemas recortados que se resuelven óptimamente con el algoritmo A^*, cuando $\delta = 0$ hay 7 problemas para los que no se llega a la solución óptima, con $\delta = 1$ solamente 2. Para los problemas $LA06 - 20$ con $\delta = 0$ se resuelven todos, mientras que con $\delta = 1$ hay 4 que no se resuelven siendo la calidad de las soluciones para los resueltos mejor o igual que con $\delta = 0$. De estos resultados se desprende que el heurístico aproxima mejor cuando $\delta = 1$, lo cual era de esperar ya que de este modo la ponderación del heurístico es más pequeña. Veamos que ocurre si empleamos una versión *"Anytime"* del algoritmo que nos dé todas las soluciones hasta agotar la memoria. Los resultados de estos experimentos se recogen en la Tabla 6.20, en la que para cada problema se muestra la misma información que en la tabla anterior, siendo la solución la mejor encontrada. Como se observa, con ambos δ se resuelven óptimamente los mismos problemas, pero la calidad de las soluciones es mejor con $\delta = 1$.

De los resultados anteriores se desprende que las características de los problemas influyen a la hora de la ponderación más adecuada para los factores Δ_m. Por ello vamos a utilizar el mismo método sistemático empleado con ponderación estática, para los problemas no resueltos óptimamente, $LA06 - 20$. En este caso, este método nos permitirá ajustar la estimación heurística h_w, concretamente el valor de δ, a cada problema. Para ello, modificamos el algoritmo A^* para que resuelva un problema con varios δ. Comenzará resolviéndolos con $\delta = 0$, e irá incrementando los δ en $0,2$ hasta que el problema no se resuelva o hasta llegar a $\delta = 2$. Se para en $\delta = 2$ para tener el mismo número de valores de prueba que con este mismo método en ponderación estática y así poder comparar los resultados. Los resultados de la primera solución se recogen en la parte izquierda (*Primera Solución*) de la Tabla 6.21. En ella, para cada problema se muestra el último valor de δ para el que se encontró solución, el número de ejecuciones y el tiempo empleado hasta llegar a un valor de δ para la que no se alcanza solución o bien a $\delta = 2$, y la solución alcanzada con la última de las ponderaciones con la que se resuelve el problema. Como se observa, la calidad de las soluciones es mejor que las obtenidas con $\delta = 0$ y $\delta = 1$ (Tabla 6.19), para todos los problemas. Este método puede mejorarse, calculando con el último δ para el que se resuelve el problema, todas las soluciones posibles hasta agotar los recursos. Los resultados de estos experimentos se recogen en la parte derecha (*Varias Soluciones*) de la Tabla 6.21. Todas las soluciones son mejores o iguales que las calculadas con $\delta = 0$ y $\delta = 1$ (Tabla 6.19), con la excepción del problema $LA12$ para el que con $\delta = 1$ la solución alcanzada es mejor. De estos resultados se desprende que el método propuesto parece un buen método para adaptar el heurístico no admisible a cada problema. Hay que destacar en relación al tiempo empleado por este método sistemático que tanto aquí como en el caso de la ponderación estática, si los incrementos de δ o los decrementos de w se hiciesen empleando una búsqueda binaria los tiempo se reducirían considerablemente. Si comparamos los resultados de este método con el mismo en ponderación estática, vemos que la ponderación adaptativa al dominio presenta un comportamiento mejor, ya que resuelve los mismos problemas siendo la calidad de las soluciones alcanzadas mejor en todos los casos, tanto si nos quedamos con la primera solución como si calculamos varias hasta agotar la memoria. La Tabla 6.22 resume la comparación de ambos méto-

dos sistemáticos; para cada método se muestra, w o δ, la solución alcanzada y el tiempo de ejecución. El tiempo medio de ejecución es mayor con la ponderación adaptativa al dominio, lo cual es lógico si tenemos en cuenta que cuanto más se aproxime h_w a h^*, es necesario expandir un mayor número de nodos para alcanzar una solución, lo cual implica un tiempo mayor. Sin embargo, en general los tiempos son bastante similares a excepción de las instancias $LA16-20$. Esto se debe a que estos problemas son más fáciles por lo que al incrementar δ el heurístico está menos ponderado y ajusta mejor a h^*, necesitando expandir más nodos para alcanzar una solución, además, los resultados experimentales ponen de manifiesto que para esos problemas no es necesario llegar tan lejos con el δ, pues para todos ellos a partir de $\delta = 0$ con todos los δ se llega a la misma solución. De todos modos y como se ha indicado anteriormente el tiempo puede reducirse en un orden logarítmico con el número de ejecuciones haciendo una búsqueda binaria del siguiente valor (w o δ) a probar.

Los resultados de esta sección nos han ofrecido las mejores soluciones alcanzadas para los problemas no resueltos óptimamente con el algoritmo A^* y la técnica de poda. En la siguiente sección compararemos estos resultados con los obtenidos con un algoritmo genético y un algoritmo de búsqueda local.

Tabla 6.19: Problema $J||\sum C_i$. A^* empleando el heurístico h_w y la técnica de poda. El algoritmo se detiene cuando encuentra la primera solución

		$\delta = 0$			$\delta = 1$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04 - 9 \times 9$	6661	46971	16	6661	197503	71	6661
$ORB08 - 9 \times 9$	5668	4731	2	5758	152884	63	5668
$ABZ8 - 9 \times 9$	3078	29969	10	3078	170378	61	3078
$ABZ7\_10 \times 10$	3262	6781	3	3262	66005	26	3262
$ABZ9\_10 \times 10$	3483	1768	1	3488	58882	25	3483
$LA38\_10 \times 10$	7113	23229	10	7113	151643	66	7113
$LA01$	4832	229	0	4946	2788	1	4833
$LA02$	4459	875	0	4567	14105	5	4459
$LA03$	4151	80	0	4157	283	0	4157
$LA04$	4259	318	1	4270	9811	3	4259
$LA05$	4072	1720	0	4262	8792	3	4072
$LA06$		166	0	9182	3293	2	8665
$LA07$		106	0	8616	119	0	8327
$LA08$		8122	4	8568	413904	331	
$LA09$		248	0	9842	423267	296	
$LA10$		81	0	9245	10718	5	9168
$LA11$		123	0	15014	3429	4	14411
$LA12$		139	0	13117	8909	7	12091
$LA13$		532	0	14941	39908	20	13711
$LA14$		153	0	16249	264	0	15149
$LA15$		402	0	15858	1212	1	14711
$LA16$		73490	30	7376	425040	190	7376
$LA17$		4462	2	6557	242063	103	6537
$LA18$		99296	42	6970	327823	143	6970
$LA19$		252760	105	7217	675526	342	
$LA20$		187588	77	7345	698131	352	

Tabla 6.20: Problema $J||\sum C_i$. A^* empleando el heurístico h_w y la técnica de poda. Encuentra todas las soluciones posibles hasta agotar la memoria

		$\delta = 0$			$\delta = 1$		
Inst.	Opt.	Exp.	T.(s)	Sol.	Exp.	T.(s)	Sol.
$ORB04\_9x9$	6661	870443	362	6661	870443	363	6661
$ORB08\_9x9$	5668	846592	397	5668	846593	400	5668
$ABZ8\_9x9$	3078	856194	576	3078	856191	579	3078
$ABZ7\_10 \times 10$	3262	707306	443	3262	707306	443	3262
$ABZ9\_10 \times 10$	3483	678544	396	3483	678544	394	3483
$LA38\_10 \times 10$	7113	709399	347	7113	709399	347	7113
$LA01$	4832	1263138	805	4832	1263138	808	4832
$LA02$	4459	1241006	903	4459	1241007	902	4459
$LA03$	4151	1357237	925	4157	1357237	921	4157
$LA04$	4259	1227459	920	4259	1227459	918	4259
$LA05$	4072	1186642	991	4072	1186642	990	4072
$LA06$		799022	333	9035	586459	285	8649
$LA07$		934473	361	8423	715606	322	8276
$LA08$		523629	280	8497	406852	333	
$LA09$		690785	294	9704	423267	298	
$LA10$		865059	318	9172	498175	237	9086
$LA11$		704804	315	14971	566972	255	14374
$LA12$		574587	283	13069	352534	227	12074
$LA13$		541703	248	14686	433484	234	13671
$LA14$		691546	290	16167	609578	272	15019
$LA15$		708681	285	15532	493693	242	14615
$LA16$		712279	315	7376	700334	337	7376
$LA17$		742957	326	6537	703542	350	6537
$LA18$		712348	326	6970	701288	339	6970
$LA19$		679076	332	7217	675528	345	
$LA20$		705829	337	7345	698131	355	

Tabla 6.21: Problema $J||\sum C_i$. A^* con h_w y la técnica de poda. Soluciones para los problemas $LA06-20$, encontradas a partir de $\delta = 0$ con incrementos de $0,2$ hasta no encontrar solución o llegar a $\delta = 2$. Deteniendo el algoritmo al encontrar la primera solución y calculando varias soluciones

		Primera Solución			*Varias Soluciones*		
Inst.	δ	Nº Eje.	T.(s)	Sol.	Nº Eje.	T.(s)	Sol.
$LA06$	1,4	9	531	8634	10	541	8631
$LA07$	2	11	710	8069	12	722	8069
$LA08$	0,4	4	444	8198	5	449	8190
$LA09$	0,8	6	301	9279	7	308	9153
$LA10$	2	11	480	8798	12	492	8798
$LA11$	1,4	9	421	14048	10	431	14014
$LA12$	0,6	5	272	12597	6	278	12594
$LA13$	1,4	9	307	13495	10	317	13495
$LA14$	1,8	11	360	14650	12	372	14556
$LA15$	1,4	9	277	14290	10	287	14279
$LA16$	1,6	10	1804	7376	11	1815	7376
$LA17$	2	11	2073	6537	12	2085	6537
$LA18$	2	11	3483	6970	12	3495	6970
$LA19$	0,6	5	1161	7217	6	1167	7217
$LA20$	0,6	5	999	7345	6	1005	7345

Tabla 6.22: Problema $J||\sum C_i$. Comparación del A^* y la técnica de poda, empleando ajuste sistemático de w en ponderación estática (A^*-SW) y de δ en h_w (A^*-DW). Soluciones para los problemas $LA06-20$, deteniendo el algoritmo al encontrar la primera solución y calculando varias soluciones

Primera Solución								
		A^*-SW				A^*-DW		
Inst.	w	NºEje.	T.(s)	Sol.	δ	NºEje.	T.(s)	Sol.
$LA06$	1,2	8	484	9123	1,4	9	531	8634
$LA07$	1,15	9	317	8498	2	11	710	8069
$LA08$	1,3	6	442	8663	0,4	4	444	8198
$LA09$	1,25	7	394	9604	0,8	6	301	9279
$LA10$	1,1	10	462	8985	2	11	480	8798
$LA11$	1,4	4	438	16013	1,4	9	421	14048
$LA12$	1,25	7	494	12875	0,6	5	272	12597
$LA13$	1,3	6	547	14500	1,4	9	307	13495
$LA14$	1,3	6	325	16197	1,8	11	360	14650
$LA15$	1,35	5	338	16096	1,4	9	277	14290
$LA16$	1,05	10	190	7376	1,6	10	1804	7376
$LA17$	1,05	10	99	6537	2	11	2073	6537
$LA18$	1,05	10	117	6977	2	11	3483	6970
$LA19$	1,1	10	332	7227	0,6	5	1161	7217
$LA20$	1,1	10	388	7366	0,6	5	999	7345
		Media	358			Media	908	
Varias Soluciones								
		A^*-SW				A^*-DW		
Inst.	w	NºEje.	T.(s)	Sol.	δ	NºEje.	T.(s)	Sol.
$LA06$	1,2	9	761	8683	1,4	10	541	8631
$LA07$	1,15	10	618	8088	2	12	722	8069
$LA08$	1,3	7	699	8216	0,4	5	449	8190
$LA09$	1,25	8	757	9274	0,8	7	308	9153
$LA10$	1,1	11	710	8862	2	12	492	8798
$LA11$	1,4	5	745	15442	1,4	10	431	14014
$LA12$	1,25	8	841	12517	0,6	6	278	12594
$LA13$	1,3	7	820	14223	1,4	10	317	13495
$LA14$	1,3	7	631	15778	1,8	12	372	14556
$LA15$	1,35	6	661	15599	1,4	10	287	14279
$LA16$	1,05	11	480	7376	1,6	11	1815	7376
$LA17$	1,05	11	385	6537	2	12	2085	6537
$LA18$	1,05	11	398	6970	2	12	3495	6970
$LA19$	1,1	11	599	7217	0,6	6	1167	7217
$LA20$	1,1	11	669	7345	0,6	6	1005	7345
		Media	652			Media	918	

6.3.7. Comparación con Otros Métodos

En esta sección comparamos los resultados del método de ponderación adaptativa con los resultados de los algoritmos genético y de búsqueda local utilizados también en la sección 5.5.4. Recordemos que el algoritmo genético es similar al propuesto para la minimización del makespan en [32] y descrito en la sección 2.4.2, con la única diferencia de la función de evaluación. La búsqueda local fue propuesta por Kreipl en [47] para resolver el problema JSS con minimización del tardiness ponderado, sin embargo, si consideramos que todos los trabajos tienen el mismo peso y que todos los due dates tienen valor 0, el problema se reduce al $J||\sum C_i$. Ambos métodos son muy utilizados en la resolución de problemas de scheduling. El algoritmo genético que vamos a emplear no es muy sofisticado, pues no utiliza ningún método de búsqueda local, sin embargo, el método de búsqueda local es un método que se puede considerar del estado del arte actual. Todos los experimentos con el algoritmo genético y la búsqueda local se ejecutarán 20 veces para cada instancia dado que se trata de algoritmos no deterministas. El algoritmo genético se configura con las probabilidades de cruce y mutación, tamaño de población y número de generaciones que se indican en las tablas. Las tablas de los experimentos realizados con el algoritmo genético y la búsqueda local recogen para cada instancia su solución óptima, la mejor solución alcanzada, la media de las mejores soluciones alcanzadas en las 20 pruebas y el número de veces que se alcanzó la mejor solución.

En la Tabla 6.23 se muestran los resultados obtenidos para los problemas $LA01-05$ y los recortados de tamaño 9×9 con ambos métodos. Como hemos visto, para estos problemas se comporta mejor el algoritmo A^*, pero entre ambos métodos se comporta mejor el de búsqueda local. Veamos ahora que ocurre con los tres únicos problemas resueltos de tamaño 10×10 y los problemas no resueltos $LA06-20$. La Tabla 6.24 recoge los tiempos empleados por el algoritmo A^* para estos problemas. Como podemos ver, las tres instancias de tamaño 10×10 se resuelven, pero para las instancias $LA06-20$ la memoria se agota. El algoritmo A^* emplea para resolver los tres problemas de tamaño 10×10 un tiempo medio de 269 segundos, siendo el mayor de los tiempos empleados 382 segundos. Para estos tres problemas hemos configurado el algoritmo genético de forma que el tiempo medio de cada prueba sea aproximadamente 840 segundos ($382s$ tiempo medio en el S.O. Ubuntu multiplicado por el factor de conversión al S.O. Windows, es decir $382s * 2{,}2 = 840s.$), tiempo que también se le dará a cada ejecución de la búsqueda local. Los resultados de estos experimentos se recogen en la Tabla 6.25. En ella se ve que, para estos problemas, la búsqueda local es mejor que el algoritmo genético; sin embargo ambos métodos son peores que el algoritmo A^* (Tabla 6.24). Para los problemas $LA06-20$, el algoritmo A^* agota la memoria en un tiempo medio de 361 segundos, siendo el mayor de los tiempos empleados 437 segundos. Para estos problemas se empleará la misma configuración anterior, de manera que se le dé al algoritmo genético un tiempo por prueba de aproximadamente 961 segundos ($437s. * 2{,}2 = 961s.$), tiempo que se le dará también a cada ejecución de la búsqueda local. La Tabla 6.26 muestra los resultados de estos experimentos. Como se observa, en media la búsqueda local es mejor que el algoritmo genético, ya que además de superar a la media del algoritmo genético en 10 instancias, alcanza mejor solución para 8 problemas y la misma para 1. El algoritmo genético alcanza mejor solución que la búsqueda local en 6 problemas y mejor media en 5 problemas.

Tabla 6.23: Resultados con el AG y la BL para las instancias resueltas de tamaño 9×9 y los problemas $LA01 - 05$

Algoritmo Genético									
10×5, $AG(/0,8/0,2/2000/6000/20/)$, 93s.					9×9, $AG(/0,8/0,2/1000/6000/20/)$, 294s.				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
$LA01$	4832	4843	4850,4	16	$ORB04 - 9 \times 9$	6661	6679	6770	1
$LA02$	4459	4459	4481,6	10	$ORB08 - 9 \times 9$	5656	5668	5671	19
$LA03$	4151	4151	4167,9	5	$ABZ8 - 9 \times 9$	3078	3091	3092,5	18
$LA04$	4259	4259	4326,5	3					
$LA05$	4072	4082	4090,6	3					
Búsqueda Local									
10×5, 20 eje., 93s.					9×9, 20 eje., 294s.				
Inst.	Opt.	Mejor	Media	Veces	Inst.	Opt.	Mejor	Media	Veces
$LA01$	4832	4832	4832,9	1	$ORB04 - 9 \times 9$	6661	6661	6661	20
$LA02$	4459	4479	4483,2	4	$ORB08 - 9 \times 9$	5656	5668	5693,3	2
$LA03$	4151	4151	4151,0	20	$ABZ8 - 9 \times 9$	3078	3078	3080,3	12
$LA04$	4259	4259	4268,8	2					
$LA05$	4072	4072	4095,0	2					

Tabla 6.24: Tiempos empleados en el S.O. Ubuntu, por el algoritmo A^* que empleando la técnica de poda, para los tres problemas resueltos de tamaño 10×10 y los problemas no resueltos $LA06 - 20$, y su traslación al S.O. Windows

Ins.	T(s) Ubuntu	Inst.	Sol.	T(s) Ubuntu
$LA06$	437	$ABZ7\_10 \times 10$	3262	160
$LA07$	420	$ABZ9\_10 \times 10$	3483	382
$LA08$	398	$LA38\_10 \times 10$	7113	265
$LA09$	428			
$LA10$	352			
$LA11$	289			
$LA12$	361			
$LA13$	323			
$LA14$	323			
$LA15$	312			
$LA16$	340			
$LA17$	377			
$LA18$	345			
$LA19$	354			
$LA20$	363	T(s) Windows		T(s) Windows
Media	361,47	795,23	Media	269	591,80
Max	437	961,40	Max	382	840,40

Tabla 6.25: Resultados de las 20 ejecuciones con el AG y la BL para las tres instancias resueltas de tamaño 10×10. Configuración AG, $(/0, 8/0, 2/300/2000/20/)$. Tiempo de aproximado de cada ejecución 840 segundos

		AG			BL		
Inst.	Opt.	Mejor	Media	Veces	Mejor	Media	Veces
$ABZ7 - 10 \times 10$	3262	3285	3290,3	1	3262	3262	20
$ABZ9 - 10 \times 10$	3483	3483	3520,9	8	3483	3483	20
$LA38 - 10 \times 10$	7113	7113	7149,1	12	7113	7118,15	14

Tabla 6.26: Resultados de las 20 ejecuciones con el AG y la BL para las instancias no resueltas $LA06 - 20$. Configuración AG, $(/0, 8/0, 2/300/2000/20/)$. Tiempo de aproximado de cada ejecución 961 segundos

	AG			BL		
Inst.	Mejor	Media	Veces	Mejor	Media	Veces
$LA06$	8632	8702,7	5	8644	8670,9	1
$LA07$	8069	8150,05	4	8116	8165,9	1
$LA08$	7946	7991,55	1	7949	7960,7	8
$LA09$	9157	9240,35	1	9113	9186,55	1
$LA10$	8931	9010,15	1	8821	8881,65	1
$LA11$	14193	14250,65	1	14148	14196,4	2
$LA12$	11844	11936,5	1	11733	11819	1
$LA13$	13415	13504,8	1	13477	13558,1	1
$LA14$	14614	14676,55	1	14671	14738,7	1
$LA15$	14221	14337,2	1	14285	14379,95	1
$LA16$	7393	7436,5	1	7376	7376,5	19
$LA17$	6537	6554,65	3	6537	6566,8	1
$LA18$	7052	7101,15	1	6970	7004,95	1
$LA19$	7218	7279,6	10	7217	7217,7	15
$LA20$	7465	7482,65	3	7394	7397,4	15

Por último vamos a comparar las soluciones alcanzadas por el algoritmo A^* con el método de ponderación adaptativa al dominio, visto en la sección anterior 6.3.6. Recordemos que este método resuelve los problemas con el algoritmo A^*, con la técnica de poda, empleando el heurístico no admisible h_{w_i}, en el que se va incrementando en $0,2$ el valor de δ hasta no alcanzar solución o llegar a $\delta = 2$. Con el último delta para el que se alcanzó solución, si se desean varias soluciones, se buscan soluciones hasta agotar la memoria. En la sección 6.3.6 hemos visto que este método se comporta mejor que el mismo método sistemático aplicado a la ponderación estática. Veamos ahora cómo es de bueno en comparación con el algoritmo genético y con la búsqueda local. Aunque los tiempos empleados por este método, trasladados al S.O Windows, son mayores que los que se da a cada prueba del algoritmo genético y de la búsqueda local podemos decir que son comparables. Por un lado, como ya hemos comentado estos tiempos pueden reducirse en un orden logarítmico haciendo una búsqueda binaria del siguiente δ a probar, además, en global el tiempo que se le da al algoritmo genético y a la búsqueda local es mucho mayor ya que se realizan 20 ejecuciones por problema. Por

otro lado, el algoritmo A^* tiene la ventaja de que una vez determinado el mejor δ se puede alcanzar las mismas soluciones en una única ejecución, lo cual requiere muy poco tiempo.

La Tabla 6.27 recoge las soluciones alcanzadas por el método de ponderación adaptativa al dominio, cuando encuentra la primera solución o cuando calcula varias. Además, muestra la mejor solución y la media de las soluciones encontradas por el algoritmo genético y la búsqueda local. Por último recoge para el algoritmo genético y la búsqueda local el número de veces que la ponderación adaptativa al dominio alcanza mejor solución, es mejor en media o igual que estos métodos. Si nos fijamos sólo en la primera solución, ya vemos que el método que proponemos es comparable con ambos métodos, ya que como se ve alcanza mejores soluciones que la búsqueda local para 5 problemas, la misma para 4 y es mejor en media para 12 problemas. Además, alcanza mejor solución que el algoritmo genético en 6 problemas, es igual en 2, y es mejor en media en 12 problemas. Cuando con el mejor δ se calculan todas las soluciones posibles hasta agotar la memoria, el método aún se comporta mejor, ya que es capaz de mejorar las mejores soluciones de la búsqueda local en 2 problemas más y la media en 1 problema más. Con respecto al algoritmo genético, mejora la mejor solución en 4 problemas más y la media en 1 problema más.

Tabla 6.27: Comparación del algoritmo A^* utilizando la poda por dominancia y el método sistemático de ponderación adaptativa al dominio (A^*-DW), con un algoritmo genético (AG) y la búsqueda local (BL), para los problemas $LA06 - 20$

	A^*-DW		AG		BL	
Inst.	Primera Sol.	Varias Sol.	Mejor	Media	Mejor	Media
$LA06$	8634	8631	8632	8702,7	8644	8670,9
$LA07$	8069	8069	8069	8150,05	8116	8165,9
$LA08$	8198	8190	7946	7991,55	7949	7960,7
$LA09$	9279	9153	9157	9240,35	9113	9186,55
$LA10$	8798	8798	8931	9010,15	8821	8881,65
$LA11$	14048	14014	14193	14250,65	14148	14196,4
$LA12$	12597	12594	11844	11936,5	11733	11819
$LA13$	13495	13495	13415	13504,8	13477	13558,1
$LA14$	14650	14556	14614	14676,55	14671	14738,7
$LA15$	14290	14279	14221	14337,2	14285	14379,95
$LA16$	7376	7376	7393	7436,5	7376	7376,5
$LA17$	6537	6537	6537	6554,65	6537	6566,8
$LA18$	6970	6970	7052	7101,15	6970	7004,95
$LA19$	7217	7217	7218	7279,6	7217	7217,7
$LA20$	7345	7345	7465	7482,65	7394	7397,4

Resumen de Resultados de A^*-DW *con respecto al AG y la BL*

	AG		BL	
	Primera Sol.	Varias Sol.	Primera Sol.	Varias Sol.
Mejor Sol.	6	10	5	7
Igual Sol.	2	2	4	4
Mejor Media	12	13	12	13

A la vista de estos resultados podemos decir que el método sistemático de ponderación adaptativa al dominio es un método competitivo tanto con el algoritmo genético como con la búsqueda local. Siendo, además un método apropiado para ofrecer soluciones no óptimas para aquellos problemas que no somos capaces de resolver en condiciones de optimalidad.

6.3.8. Conclusiones

En la sección anterior hemos visto para la versión del problema $J||C_{max}$, cómo la combinación de la técnica de poda con otros métodos es un método efectivo. En esta sección, para la versión del problema $J||\sum C_i$, versión que parece más difícil, nos hemos centrado en tratar de resolver los problemas no resueltos con diversas estrategias, concretamente con la mejora de heurísticos, el cálculo de cotas superiores, la reducción del espacio de búsqueda con el parámetro δ en la generación de sucesores, la ponderación de la función heurística, las discrepancias heurísticas limitadas y la ponderación adaptativa al dominio. Además, hemos comparado las soluciones encontradas para los problemas $LA06-20$ en esta sección con las soluciones encontradas por un algoritmo genético y un algoritmo de búsqueda local.

De acuerdo con el estudio experimental, la mejora de heurísticos, el cálculo de cotas superiores y las discrepancias heurísticas limitadas no ofrecen buenos resultados. La ponderación estática y el heurístico no admisible propuesto, por si solos tampoco ofrecen soluciones con buena calidad. Sin embargo, el método que permite ajustar la ponderación al problema presenta un mejor comportamiento, siendo mejor la ponderación adaptativa al dominio, ya que incluye en la ponderación información sobre el problema. Este método combinado con la técnica de poda es comparable tanto con el algoritmo genético como con la búsqueda local, ofreciendo en muchos casos mejores soluciones que las que ofrecen estos métodos.

Como hemos visto los problemas $J||\sum C_i$ son problemas más difíciles, por lo que algunas de las técnicas que sí eran efectivas en la versión del problema $J||C_{max}$ aquí no lo son. Sin embargo, hemos propuesto un método sistemático que nos permite llegar a soluciones con una calidad similar, y muchas veces mejor, a las que proporcionan un algoritmo genético y un algoritmo de búsqueda local, métodos más empleados cuando relajamos la condición de optimalidad.

6.4. Conclusiones

En este capítulo hemos tratado de resolver algunos de los problemas más representativos de los no resueltos con el algoritmo A^* empleando la técnica de poda. Concretamente para la versión del problema $J||C_{max}$ nos hemos centrado en los problemas de las baterías LA y *Selectos* de tamaño 10×10, y en la versión $J||\sum C_i$ en los problemas $LA06-20$ y en los tres únicos problemas de tamaño 10×10 resueltos óptimamente. En ambos casos, en primer lugar hemos tratado de resolverlos, manteniendo la restricción de optimalidad, mediante el cálculo de cotas superiores que nos permiten podar el espacio de búsqueda y ofrecer alguna solución en caso de no encontrar la óptima. Como vimos para la versión $J||C_{max}$ el cálculo de cotas sí es efectivo, mientras que en el caso de $J||\sum C_i$ no lo es, ya que las cotas calculadas no son buenas. En la versión del problema

$J||C_{max}$ hemos tratado de reducir también el espacio de búsqueda utilizando la propagación de restricciones *Selección Inmediata*, como vimos aún en las mejores condiciones esta técnica no es efectiva en el espacio de búsqueda de planificaciones activas empleado por el algoritmo A^*.

Una vez visto adonde podemos llegar en condiciones de optimalidad, hemos relajado la optimalidad en ambas versiones del problema empleando diferentes estrategias. Entre ellas podemos destacar la reducción del espacio de búsqueda mediante el parámetro δ en la generación de sucesores y la ponderación estática de la función heurística. En la versión $J||C_{max}$ gracias a estas técnicas hemos logrado resolver 8 problemas que sin ellas no se resuelven. Para el problema $J||\sum C_i$ los mejores resultados los hemos obtenido con el método de ponderación dependiente del dominio. Los resultados de este método son comparables a los que ofrecen otros métodos del estado del arte como son un algoritmo genético y un algoritmo de búsqueda local. Las soluciones que presentamos para los problemas $LA16-19$ son las mismas por los tres métodos, por lo que pensamos que se puede tratar de las soluciones óptimas, aunque no lo podremos asegurar hasta que seamos capaces de resolver estos problemas sin relajar la optimalidad.

En este capítulo hemos visto cómo la técnica de poda por dominancia combinada con estrategias que relajan la optimalidad, es capaz de ofrecer buenas soluciones para aquellos problemas que no se resuelven óptimamente, siendo capaz en algunos casos de llegar a la solución óptima para instancias cuya solución óptima es conocida.

Algunos de los resultados de este estudio experimental se recogen en [76, 81, 80].

Capítulo 7

CONCLUSIONES Y TRABAJO FUTURO

7.1. Conclusiones y Principales Aportaciones

La motivación inicial de esta tesis fue el desarrollo de métodos exactos eficientes, aplicables a distintas versiones del problema JSS, basados en el algoritmo A^*. Para ello hemos elegido el espacio de búsqueda de las planificaciones activas por el hecho de que es dominante para todas las funciones objetivo convencionales. Nos hemos centrado en dos versiones del problema que consideramos representativas de las distintas variantes que se encuentran en la literatura: $J||C_{max}$ y $J||\sum C_i$. En la primera de ellas la función objetivo es el makespan y en la segunda el tiempo de flujo total. Estas dos funciones objetivo son representativas de las clases *bottleneck* y *suma* respectivamente, de acuerdo con la clasificación propuesta por P. Brucker en [13].

En los dos casos hemos comenzado por tratar de definir heurísticos admisibles con el mayor grado de información posible. Para ello hemos utilizado distintas relajaciones a problemas de secuenciamiento una máquina (OMS) y nos hemos apoyado en los resultados obtenidos por otros investigadores para distintas versiones del problema OMS como por ejemplo en los trabajos de Carlier y Pinson [15], [16] y [17] sobre el problema OMS con cabezas, colas y minimización del makespan, o el trabajo de Baptiste, Carlier y Jouglet [5] sobre el problema OMS con due dates y minimización del tardiness.

Como aportación principal, en esta tesis hemos formulado y desarrollado una técnica de poda basada en relaciones de dominancia entre estados y la hemos aplicado a las versiones del problema JSS, $J||C_{max}$ y $J||\sum C_i$. Esta técnica permite reducir considerablemente tanto el espacio de búsqueda efectivo como el tiempo de ejecución. Para ello, se establece una condición suficiente de dominancia que depende de la naturaleza del problema (función objetivo y función heurística), así como unas reglas que hacen eficiente la evaluación de dicha condición.

Para ambas versiones del problema, hemos realizado un estudio experimental en el que se comparan los heurísticos propuestos, se analiza la eficiencia de la técnica de poda por dominancia y se

muestra el umbral del tamaño de las instancias que se pueden resolver con la combinación de estos métodos. Para la versión del problema $J||C_{max}$ el límite del tamaño de problemas que podemos resolver óptimamente, gracias al empleo de la técnica de poda por dominancia, está próximo al umbral de tamaño que deja de considerarse pequeño, es decir 10×10. La eficiencia de nuestra aproximación está lejos de la del algoritmo propuesto por P. Brucker et al. en [12], que es sin duda la mejor aproximación conocida hasta el momento. Sin embargo, es más eficiente que otras aproximaciones que al igual que la nuestra no se basan en el camino crítico y que por lo tanto son más fácilmente generalizables a otras funciones objetivo distintas del makespan. Este es el caso del algoritmo de backtracking, guiado por un heurístico de ordenación de variables y valores, propuesto por N. Sadeh y M. Fox en [74].

Los resultados experimentales ponen de manifiesto que la versión del problema $J||\sum C_i$ es más difícil que la versión $J||C_{max}$. En este caso el límite del tamaño de problemas que podemos resolver óptimamente es un poco menor que para la versión $J||C_{max}$. Para las instancias con igual número de problemas y máquinas (problemas cuadrados), el límite es también 10×10, aunque se resuelven algunas instancias más que con la versión $J||C_{max}$. Sin embargo, no es posible resolver ninguna de las instancias de tamaños 15×5 y 20×5, a diferencia del problema $J||C_{max}$ para el que prácticamente se resuelven todas estas instancias. Es decir que la dificultad del problema crece al aumentar el número de trabajos. Los resultados experimentales demuestran también que nuestra aproximación para el problema $J||\sum C_i$ es más eficiente que dos métodos tomados de la literatura. Estos métodos son un algoritmo genético y un algoritmo de búsqueda local. La razón por la que hemos elegido estos métodos no es otra que el hecho de que disponemos de las implementaciones. En el caso del algoritmo genético, hemos hecho una adaptación del algoritmo propuesto en [32] para el problema $J||C_{max}$. Y en el caso del algoritmo de búsqueda local, hemos utilizado la implementación del método propuesto en [47] para el problema de minimización del tardiness ponderado. El tiempo de flujo total es un caso particular del tardiness ponderado en el que todos los trabajos tienen la misma prioridad y todos los due dates son cero. Esta implementación forma parte de la herramienta LEKIN que se puede descargar de forma libre de la dirección http://www.stern.nyu.edu/om/software/lekin.

También hemos estudiado de forma experimental el comportamiento del método de poda por dominancia en combinación con otras estrategias como el cálculo de cotas superiores y la propagación de restricciones, y en particular con estrategias en las que se relaja la condición de optimalidad, como la búsqueda en anchura limitada y la ponderación de la función heurística. En el caso de la ponderación heurística, hemos optado por mantener las mismas condiciones de dominancia formuladas para el algoritmo original, de este modo es posible que se descarten algunas podas que de otro modo sí se podrían detectar, pero con un coste computacional mayor.

En el caso de la versión $J||C_{max}$, hemos comprobado que la combinación del algoritmo exacto con el cálculo de cotas superiores, aplicando un algoritmo voraz a partir de una fracción pequeña de los nodos expandidos, produce una mejora muy importante. En particular, se resuelven todas las instancias de tamaños 15×5 y 20×5, y en muchos casos se reduce de forma importante el tiempo de ejecución. Sin embargo, este método tiene un efecto nulo cuando se aplica al problema $J||\sum C_i$. En nuestra opinión, este resultado constituye una evidencia más de que el problema $J||\sum C_i$ es más difícil de resolver que el problema $J||C_{max}$, y en particular que los heurísticos que se utilizan para

guiar al algoritmo voraz, que no son otros que los heurísticos admisibles que se utilizan para guiar al algoritmo A^*, son menos precisos en el segundo caso que en el primero.

Para el problema $J||C_{max}$ hemos experimentado con el método de propagación de restricciones propuesto en [16] denominado *selección inmediata*. Este método se utiliza también en el algoritmo de ramificación y poda propuesto en [12], y en este caso constituye uno de los elementos esenciales para la eficiencia de este algoritmo. Sin embargo, cuando, como en nuestro caso, se utiliza en combinación con el espacio de búsqueda de las planificaciones activas, la reducción del espacio de búsqueda efectivo que produce es prácticamente irrelevante, sobre todo cuando se utiliza en combinación con el método de poda por dominancia. En la literatura existen otras reglas de propagación de restricciones, como por ejemplo las que se proponen en [25] y [26]. Aunque no hemos experimentado de forma exhaustiva con todas ellas, suponemos que su efecto será similar al de la selección inmediata. Este hecho pone de manifiesto una de las desventajas del espacio de búsqueda que hemos utilizado frente a otros espacios de soluciones como el que explora el algoritmo de P. Brucker.

Con las dos versiones del problema hemos experimentado con métodos de búsqueda en anchura limitada, en particular con un método simple de búsqueda con discrepancia heurística limitada y con un método de reducción del número de sucesores de cada estado que limita el tiempo de máximo de inactividad de las máquinas (mediante el uso del parámetro δ tal y como se describe en la sección 2.4.1). También hemos considerado versiones *"Anytime"* de estos algoritmos. En general se consiguen soluciones sub-óptimas en un tiempo razonable, pero estos métodos no resultan muy competitivos.

Por último hemos considerado la ponderación de la función heurística, en particular el método de ponderación estática que es el más simple y uno de los que mejores resultados producen tal y como se indica, por ejemplo, en [87]. En el caso del problema $J||\sum C_i$, con este método se consiguen resultados similares o mejores a los que se obtienen con los métodos de búsqueda con anchura limitada, en particular cuando se utilizan versiones *"Anytime"* y se aplican de forma iterativa para distintos valores del parámetro de ponderación. Sin embargo, el comportamiento de estos métodos junto con la evidencia de que los heurísticos para este problema son poco precisos, nos ha llevado al diseño de un nuevo método de ponderación dependiente del problema que de nuevo ofrece resultados competitivos con el algoritmo genético y con el algoritmo de búsqueda local en el cálculo de soluciones sub-óptimas. En nuestra opinión, este resultado constituye una de las aportaciones relevantes de esta tesis, ya que se trata de un método que puede resultar apropiado para cualquier problema en el que se utilicen heurísticos disyuntivos y de forma especial si además las funciones objetivo son aditivas.

7.2. Trabajo Futuro

El trabajo desarrollado durante esta tesis deja abiertas unas cuantas líneas de investigación. La primera de ellas y la más natural es generalizar la aplicación de la poda por dominancia a otras funciones objetivo distintas del makespan y del tiempo de flujo total. También sería interesante estudiar la posibilidad de aplicar la técnica de poda por dominancia a otras versiones del problema JSS más próximas al mundo real, como por ejemplo la versión del problema con tiempos de setup

dependientes de la secuencia, denotada como $J|S_{ij}^k|C_{max}$, donde S_{ij}^k representa el tiempo de setup de la máquina k cuando sale una tarea del trabajo i y entra a continuación otra tarea del trabajo j. Por otro lado también resultaría interesante tratar de aplicar la técnica de poda por dominancia a otros problemas de optimización combinatoria, como por ejemplo el TSP (Travelling Salesman Problem), o el problema de corte de piezas (Cutting Stock).

Como hemos visto en el desarrollo de la tesis, los heurísticos admisibles diseñados para el problema $J||\sum C_i$ no son muy precisos, es decir subestiman mucho el coste real para alcanzar una solución desde un estado, por lo que el algoritmo A^* tiene que visitar muchos estados para alcanzar una solución. Por ello creemos que deberíamos esforzarnos en tratar de diseñar heurísticos más informados y también heurísticos con propiedades que nos permitan mejorar el proceso de poda.

Además, cuando relajamos la condición de optimalidad, el método sistemático de ponderación adaptativa al dominio ha dejado abierta su aplicación a la versión del problema $J||C_{max}$, así como a otras versiones del problema JSS e incluso a otras familias de problemas.

Por último, sería interesante también abordar con búsqueda heurística otras variantes más sofisticadas del problema JSS, como por ejemplo la versión del problema en la que se considera que las tareas tienen duraciones imprecisas. Este problema ha sido tratado con distintas metaheurísticas, como por ejemplo algoritmos evolutivos [29], o con combinaciones de estos algoritmos con estrategias de búsqueda local como las propuestas en [33] y [34]. En estas aproximaciones las duraciones se representan mediante números borrosos. Nuestro objetivo es utilizar la formulación del problema propuesta en [34] y a partir de ella definir los elementos del algoritmos de búsqueda como los costes de los operadores y las funciones heurísticas. Hasta donde nosotros sabemos, no existe en la literatura ninguna aproximación a problemas de scheduling borrosos mediante búsqueda heurística en espacios de estados, por lo que ésta puede ser una línea interesante de trabajo futuro.

Bibliografía

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Managament Science*, 34:391–401, 1988.

[2] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 3:149–156, 1991.

[3] C. Artigues and D. Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, 159(1):135–159, 2008.

[4] K. R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, New York, 1974.

[5] P. Baptiste, J. Carlier, and A. Jouglet. A brachand-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 158:595–608, 2004.

[6] J. Beasley. Or-library. `http://people.brunel.ac.uk/ mastjjb/jeb/info.html`, 1990.

[7] J. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[8] C. Bierwirth. A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spectrum*, 17:87–92, 1995.

[9] C. Bierwirth and D. C. Mattfeld. Production scheduling and rescheduling with genetic algoritms. *Evolutionary Computation*, 7:1–17, 1999.

[10] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Werglarz. *Scheduling Computer and Manufacturing Processes.* Springer, Berlin, 1996.

[11] P. Brucker. *Scheduling Algorithms.* Springer, 4th edition, 2004.

[12] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[13] P. Brucker and S. Knust. *Complex Scheduling.* Springer, 2006.

[14] P. Brucker and O. Thiele. A branch and bound method for the general-job shop problem with sequence-dependent setup times. *Operations Research Spektrum*, 18:145–161, 1996.

[15] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[16] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[17] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[18] P. Chakrabarti, S. Ghosh, and S. DeSarkar. Admissibility of ao* when heuristics overestimate. *Artificial Intelligence*, 34 (1):97–113, 1988.

[19] E. G. J. Coffman. *Scheduling in Computer and Job Shop Systems.* J. Wiley, New York, 1976.

[20] R. W. Conway, W. Maxwell, and L. Miller. *Theory of Scheduling.* Addison Wesley, Reading, Mass. USA, 1967.

[21] H. Davis, A. Bramanti-Gregor, and J. Wang. The advantages of using depth and breadth components in heuristic search. *Methodologies for Intelligent Systems*, 3:19–28, 1988.

[22] P. de Diversas Universidades Españolas. Coordinadores José Tomás Palma Méndez y Roque Marín Morales. *Inteligencia Artificial: Métodos, técnicas y aplicaciones.* MC Graw Hill, Madrid, España, 2008.

[23] R. Dechther and J. Pearl. Generalized best first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, 32 (3):505–536, 1985.

[24] M. Dell' Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research*, 41:231–252, 1993.

[25] U. Dorndorf, E. Pesch, and T. Phan-Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122:189–240, 2000.

[26] U. Dorndorf, E. Pesch, and T. Phan-Huy. Constraint propagation and problem descomposition: A preprocessing procedure for the job shop problem. *Annals of Operations Research*, 115:125–142, 2002.

[27] H. Emmons. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17:701–715, 1969.

[28] I. Essafi, Y. Mati, and S. Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35:2599–2616, 2008.

[29] P. Fortemps and M. Roubens. Ranking and defuzzification methods based on area compensation. *Fuzzy Sets and Systems*, 82:319–330, 1996.

[30] S. French. *Secuencing and scheduling: an introduction to the mathematics of the job-shop*. Ellis Horwood Limited, New York, 1982.

[31] B. Giffler and G. L. Thomson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[32] M. A. González, C. R. Vela, and R. Varela. Scheduling with memetic algorithms over the spaces of semi-active and active schedules. *Lecture Notes in Artificial Intelligence*, 4029:370–379, 2006.

[33] I. González Rodríguez, J. Puente, C. R. Vela, and R. Varela. Semantics of schedules for the fuzzy job shop problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 38(3):655–666, 2008.

[34] I. González Rodríguez, C. R. Vela, J. Puente, and R. Varela. A new local search for the job shop problem with uncertain durations. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008)*, pages XX–XX. AAAI Press, 2008.

[35] E. Hansen and R. Zhou. Anytime heuristic search. *Artificial Intelligence*, 28:267–297, 2007.

[36] E. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129 (1-2):35–62, 2001.

[37] L. Hatzikonstantis and C. B. Besant. Job-shop scheduling using certain heuristic search algorithms. *Int. J. Adv. Manuf. Tecnol.*, 7:251–261, 1992.

[38] O. Holthaus and C. Rajendram. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48 (1):87–105, 1997.

[39] L. P. Ingolotti Hetter. *PhD: Modelos y Métodos para la Optimización y Eficiencia de la Programación de Horarios Ferroviarios*. Directores: Federico Barber Sanchis y Pilar Tormos Juan, 2007.

[40] R. Korf. Depth-first iterative deepening: An optimal admissible tree search algorithm. *Artificial Intelligence*, 27:97–109, 1985.

[41] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62 (1):47–78, 1993.

[42] R. Korf. Divide-and-conquer bidirectional search: First results. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI99)*, pages 1184–1189, Stockholm, Swedem, 1999.

[43] R. Korf. An improved algorithm for optimal bin-packing. In *Proceedings of the 13th International Conference on Artificial Intelligence (IJCAI03)*, pages 1252–1258, 2003.

[44] R. Korf. Optimal rectangle packing: New results. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS04)*, pages 132–141, 2004.

[45] R. Korf and W. Zhang. Divide-and-conquer frontier search applied to optimal secuence aligment. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI00)*, pages 910–916, Austin, TX, 2000.

[46] R. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier search. *J. ACM*, 52 (2):715–748, 2005.

[47] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3:125–138, 2000.

[48] V. Kumar. Algorithms for constraint satisfaction problems. *AI Magazine*, 13:32–44, 1992.

[49] E. Lawer, J. Lenstra, A. Rinooy Kan, and D. Shmoys. *Secuencing and Scheduling: Algorithm and Complexity*, volume 4. North-Holand, Amsterdan, 1993.

[50] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1984.

[51] D. C. Mattfeld. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling.* Springer-Verlag, 1995.

[52] P. Meseguer and T. Walsh. Interleaved and discrepancy based search. In *Proceedings of the 13th European Conference on Artificial Intelligence, ECAI-98*, pages 239–243, 1998.

[53] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics.* Springer, 2000.

[54] T. Nazaret, S. Verma, S. Bhattacharya, and A. Bagchi. The multiple resource constrained project scheduling problem: A breadth-first approach. *European Journal of Operational Research*, 112:347–366, 1999.

[55] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[56] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, CA, 1980.

[57] N. J. Nilsson. *Principios de Inteligencia Artificial.* Díaz de Santos, Madrid, 1987.

[58] N. J. Nilsson. *Artificial Intelligence. A New Synthesis.* Morgan Kaufmann, San Francisco, California, 1998.

[59] N. J. Nilsson. *Inteligencia Artificial. Una Nueva Síntesis.* McGraw Hill, Madrid, 2001.

[60] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop scheduling problem. *Management Science*, 42:797–813, 1996.

[61] J. Pearl. *Heuristics: Intelligent Search strategies for Computer Problem Solving.* Addison-Wesley, Massachusetts, 1984.

[62] J. Pearl. Parallel machine scheduling models with fuzzy processing times. *Information Sciences*, 166:49–66, 1984.

[63] T. Phan-Huy. *Constraint Propagation in Flexible Manufacturing.* Springer, Heidelberg, 2000.

[64] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, New York, 1995.

[65] M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services.* Springer, New York, 2005.

[66] M. L. Pinedo. *Scheduling. Theory, Algorithms, and Systems.* Springer, third edition, 2008.

[67] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[68] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1:193–204, 1970.

[69] I. Pohl. The avoidance of relative catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI-73*, pages 12–17. SCHEDULIN, Ponderación, 1973.

[70] A. H. G. Rinnooy Kan. *Machine Scheduling Problems: Classification, complexity and computations.* Martinus Nijhoff, The Hague, 1976.

[71] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec constraintes disjonctives. Note d.s. no. 9 bis, d6c, SEMA, Matrouge, Paris, 1964.

[72] S. Russell. Efficient memory-bounded search methods. In *10th European Conference on Artificial Intelligence (ECAI92)*, pages 1–5, 1992.

[73] S. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach.* Prentice Hall, US, 2 edition, 2003.

[74] N. Sadeh and M. S. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86:1–41, 1996.

[75] M. Shimbo and T. Ishida. Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, 146 (1):1–41, 2003.

[76] M. Sierra and R. Varela. Búsqueda heurística para problemas de scheduling. In *Actas de CIO 2005, IX Congreso de Ingeniería de Organización*, pages 199–200, 2005.

[77] M. Sierra and R. Varela. Optimal scheduling with heuristic best first search. *Lecture Notes in Computer Science*, 3673:173–176, 2005.

[78] M. Sierra and R. Varela. Pruning by dominance in best-first search. In *Proceedings of CAEPIA'2007*, volume 2, pages 289–298, 2007.

[79] M. Sierra and R. Varela. Pruning by dominance in best-first search for the job shop scheduling problem with total flow time. *Journal of Intelligent Manufacturing (JIM)*, To appear, 2009.

[80] M. R. Sierra, M. C., and R. Varela. Weighting disjunctive heuristics for scheduling problems with summation cost functions. In *Proceedings of Workshop on Planning, Scheduling and Constraint Satisfaction, CAEPIA'2009*, Submited, 2009.

[81] M. R. Sierra and R. Varela. Pruning search spaces by dominance rule: Case study in the job shop scheduling. *International Journal of Reasoning-based Intelligent Systems (IJRIS)*, Accepted. In edition.

[82] M. R. Sierra and R. Varela. A new admissible heuristic for the job shop scheduling problem with total flow time. In *Proceedings of Workshop on Constraint Satisfaction Techniques for Planning and Scheduling. International conference on Automated Planning and Scheduling. ICAPS'2008*, Online, 2008.

[83] R. Storer and F. Talbot. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38:1494–1509, 1992.

[84] E. Taillard. Benchmarks for basic scheduling problems. *European Journal od operational Research*, 64:278–285, 1993.

[85] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6 (2):108–117, 1994.

[86] V. Tanaev, V. Gordon, and Y. Shafransky. *Scheduling theory. Single-stage systems.* Kluwer Academic Publishers, Dordrecht, 1994.

[87] J. Thayer and W. Ruml. Faster than weighted A*: An optimistic aproach to bounded suboptimal search. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 355–362, 2008.

[88] E. Tsang. Consistency and satisfiability in constraint satisfaction problems. In *Proceedings of the Artificial Intelligence and Simulated Behaviour. 89 Conference*, pages 41–48, Brighton, 1989.

[89] P. Van Laarhoven, E. Aarts, and K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[90] R. Varela, D. Serrano, and M. Sierra. New codification schemas for scheduling with genetic algorithms. *Proceedings of IWINAC 2005. Lecture Notes in Computer Science*, 3562:11–20, 2005.

[91] R. Varela and E. Soto. Sheduling as heuristic search with state space reduction. *Lecture Notes in Computer Science*, 2527:815–824, 2002.

[92] R. Varela, C. R. Vela, J. Puente, and A. Gómez. A knowledge-based evolutionary strategy for scheduling problems with bottlenecks. *European Journal of Operational Research*, 145:57–71, 2003.

[93] A. Vepsalanen and T. Morton. Priority rules for job shops with weighted tardiness cost. *Management Science*, 33 (8):1035–1047, 1987.

[94] T. Yamada and R. Nakano. Scheduling by genetic local search with multi-step crossover. In *Proceedings of Fourth International Conference On Parallel Problem Solving from Nature (PPSN IV 1996)*, pages 960–969, 1996.

[95] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research*, 35:282–294, 2008.

[96] R. Zhou and E. Hansen. Sparse-memory graph search. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI03)*, pages 1259–1266, Acapulco, Mexico, 2003.

[97] R. Zhou and E. Hansen. Sweep A*: Space-efficient heuristic search in partially ordered graphs. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI03)*, pages 427–434, Sacramento, CA, 2003.

[98] Y. Zhou, B. Li, and J. Yang. Study on job shop scheduling with sequence-dependent setup times using biological immune algorithm. *The International Journal of Advanced Manufacturing Technology*, 30(1-2):105–111, 2006.