UNIVERSIDAD DE OVIEDO

DEPARTAMENTO DE INFORMÁTICA

Métodos Heurísticos Avanzados para Problemas de Scheduling

Tesis Doctoral

Autor: Carlos Mencía Cascallana

Directores
José Ramiro Varela Arias
María R. Sierra Sánchez

Julio, 2013

A mis padres

Agradecimientos

Quiero agradecer a mis directores de tesis, Ramiro Varela Arias y María R. Sierra Sánchez su enorme dedicación a este proyecto. Sin duda, esta tesis no habría sido posible sin su trabajo. También, agradezco a mis compañeros del Grupo de Investigación en Optimización y Scheduling Inteligentes de la Universidad de Oviedo su ayuda prestada en los últimos años. Desde el primer día me he sentido completamente integrado en el grupo, he aprendido mucho trabajando con vosotros y, sinceramente, tengo que reconocer que me lo he pasado muy bien.

Estoy muy agradecido a mi familia y a mis amigos. Han sido muchas las ocasiones en las que os he *aburrido* hablando de temas muy alejados a vuestra actividad cotidiana. Siempre me he encontrado un apoyo impagable por vuestra parte.

Esta tesis doctoral ha sido financiada por el Principado de Asturias a través del programa de ayudas predoctorales *Severo Ochoa* (referencia BP-09105). También, se ha enmarcado en los proyectos del Plan Nacional de Investigación TIN2007-67466-C02-01 y TIN2010-20976-C02-02, financiados por el Gobierno de España.

Resumen

Los problemas de *scheduling* están presentes en numerosos ámbitos reales. En general, consisten en organizar la ejecución de un conjunto de operaciones en diferentes recursos, satisfaciendo diversas restricciones con el fin de optimizar alguna métrica o función objetivo. Muchos de ellos, como el paradigmático problema Job Shop Scheduling, pertenecen a la clase de complejidad NP-hard, lo que significa que pueden ser realmente difíciles de resolver. En las últimas décadas ha habido un estudio intensivo de los problemas de scheduling por parte de campos científicos como la Inteligencia Artificial o la Investigación Operativa, dando lugar a un notorio número de técnicas de resolución, tanto exactas como aproximadas.

Esta Tesis Doctoral tiene como objetivo el diseño de algoritmos exactos eficientes para problemas de scheduling, capaces de resolver de forma óptima instancias del problema de hasta cierto tamaño y de calcular tanto cotas inferiores no triviales como buenas soluciones para instancias más difíciles en un tiempo corto y empleado recursos de memoria escasos. Concretamente, la investigación se ha centrado en el problema Job Shop Scheduling y en el problema Job Shop Scheduling con Operarios, una generalización del primero que considera restricciones complejas derivadas de la presencia de trabajadores humanos en los entornos productivos. Por medio de técnicas de búsqueda heurística y de razonamiento basado en restricciones, hemos contribuido a este campo en varios modos.

En primer lugar, evaluamos dos variantes de algoritmos de búsqueda primero en profundidad en el contexto del problema Job Shop Scheduling, y proponemos un algoritmo de búsqueda en profundidad parcialmente informada que explota eficazmente el conocimiento del domino del problema para guiar la búsqueda hacia zonas prometedoras del espacio de búsqueda. Este algoritmo saca partido de un heurístico original que explota un método de propagación de restricciones.

En segundo lugar, proponemos una nueva estrategia de búsqueda, que extiende la técnica *Iterative Deepening A** para resolver problemas de scheduling y de optimización con restricciones. La nueva estrategia, denominada *Intensified Iterative Deepening A**, emplea la propagación de restricciones en un modo no conservativo, actualiza el umbral de coste de forma liberal entre iteraciones consecutivas y lanza búsquedas en profundidad limitadas desde algunos estados para muestrear distintas regiones del espacio de búsqueda. Esta técnica ha funcionado muy bien en el problema Job Shop Scheduling.

Por último, diseñamos varias componentes de algoritmos de búsqueda heurística para el problema Job Shop Scheduling con Operarios: Una definición de un espacio de búsqueda basado en un nuevo esquema de generación de planificaciones denominado *OG&T*, dos heurísticos y una relación de dominancia empleada para podar estados. Asimismo, proponemos una nueva estrategia de búsqueda heurística, denominada A*DFS, que combina de un modo original los algoritmos A* y búsqueda en profundidad, obteniendo muy buenos resultados.

Summary

Scheduling problems arise in a variety of real-life settings. In general, they consist on organizing the execution of a set of operations on different resources, satisfying several constraints with the aim of optimizing some metric or objective function. Many of them, as the paradigmatic Job Shop Scheduling Problem, belong to the NP-hard complexity class, which means they may be really difficult to solve. Over the last decades, scientific fields such as Artificial Intelligence or Operations Research have intensively studied scheduling problems, giving rise to a notorious number of solving techniques, both exact and approximate.

This Ph.D. Thesis is aimed at designing efficient exact algorithms for scheduling problems, capable of solving to optimality problem instances up to a certain size and computing both non trivial lower bounds and good solutions to harder instances, taking short time and using scarce memory resources. Concretely, this research has focused on the Job Shop Scheduling Problem and the recent Job Shop Scheduling Problem with Operators, a generalization of the first one that considers complex constraints derived from the presence of human workers in production environments. By means of heuristic search and constraint-based reasoning techniques, we have contributed to this field in several ways.

Firstly, we evaluate two variants of depth-first search algorithms in the context of the Job Shop Scheduling Problem, and propose a partially informed depth-first search algorithm that exploits the knowledge from the problem domain effectively in order to guide the search towards promising regions of the search space. This algorithm takes profit from an original heuristic that exploits a constraint propagation method.

Secondly, we propose a new search strategy that extends *Iterative Deepening A** to effectively cope with scheduling and constraint optimization problems. The new strategy, termed *Intensified Iterative Deepening A**, uses constraint propagation in a non conservative way, updates the cost threshold liberally between consecutive iterations and yields limited depth-first searches from some states in order to sample different regions of the search space. This technique has been shown to be effective in solving the Job Shop Scheduling Problem.

Finally, we design several heuristic search components for the Job Shop Scheduling Problem with Operators: a definition of a search space based on a new schedule generation scheme termed $OG\&T$, two heuristic estimations and a dominance relation used to prune states. Besides, we propose a new search strategy, termed A*DFS, that combines the A* algorithm and depth-first search in an original way, obtaining very good results.

Índice de Contenidos

Capítulo 1

INTRODUCCIÓN

Se puede definir *scheduling* como un proceso de toma de decisiones sobre cómo organizar la ejecución de un conjunto de tareas teniendo en cuenta recursos disponibles y restricciones de diversa naturaleza con el fin de optimizar una determinada métrica.

Bajo una definición tan general, no es difícil entrever que este proceso constituye una parte fundamental en multitud de actividades en ámbitos industriales, administrativos o científicos. Cubrir turnos de trabajo con personal, organizar la producción en una fábrica, optimizar el uso de quirófanos y cirujanos en un hospital o definir los horarios y aulas de clase de las asignaturas en la Universidad son ejemplos de problemas de scheduling. Las aplicaciones prácticas y las implicaciones económicas son enormes; en muchas ocasiones, la calidad de una planificación puede ser determinante para el éxito de una actividad.

Tradicionalmente, este ha sido un proceso costoso realizado por expertos humanos, y se ha considerado necesario el empleo de inteligencia para llevarlo a cabo. Encontrar una solución que satisfaga todas las restricciones impuestas no es casi nunca una tarea trivial, y puede requerir evaluar un número exponencial de posibles opciones, para lo que es necesario invertir mucho tiempo y esfuerzo. Por este motivo, disponer de herramientas automáticas de scheduling tiene un gran interés, ya que permite ahorrar tiempo y a la vez contar con planificaciones más adecuadas.

La dificultad intrínseca de muchos de estos problemas, así como su gran campo de aplicación, ha suscitado mucho interés en campos científicos como la inteligencia artificial, la investigación operativa o las matemáticas aplicadas. Ya en los inicios de la Teoría de la Complejidad Computacional, algunos problemas de este tipo se encontraban en la lista de problemas NP-completos conocidos [25].

Hacer una descripción exhaustiva de los distintos tipos de problemas de scheduling es un objetivo fuera del alcance de esta memoria. A este respecto, el lector interesado puede consultar libros como [10], [11], [13] o [57], o revistas científicas como *Journal of Scheduling*.

Esta tesis doctoral se centra en una clase de problemas de scheduling de la que el problema Job Shop Scheduling (JSS), presente en multitud de situaciones reales, es un paradigma.

En concreto, un *job shop* (taller de trabajos) es un modelo de producción en el que, utilizando un conjunto limitado de recursos, se fabrican productos especializados, cada uno con una configuración propia y por tanto un proceso de fabricación diferente. Es un caso opuesto a la producción en serie, donde se fabrica un gran número de productos idénticos siguiendo el mismo proceso. Por ejemplo, se puede pensar en una carpintería donde podría haber una máquina para cortar madera, una mesa utilizada para ensamblar distintas piezas, una máquina para barnizar, y otros recursos. Los trabajos serían los pedidos de los clientes, y constarían de una secuencia propia de tareas con un tiempo de procesamiento determinado en cada una de las máquinas; así, construir un mueble podría requerir cortar madera durante 30 minutos, después barnizar las partes durante 50 minutos y por último ensamblar piezas durante 10 minutos. Programar la producción de un conjunto de pedidos de modo que se tarde el menor tiempo posible en completarse es una instancia del problema JSS. En la misma línea del ejemplo anterior, el JSS podría aparecer en la organización de una lavandería, una imprenta, un taller de reparación de vehículos, una empresa de fabricación de componentes especiales o un restaurante. Este modelo no sólo se restringe a determinados sectores productivos, sino que tiene aplicación en diversas actividades administrativas (una oficina donde se atienden distintos tipos de solicitudes), científicas (organización experimentos en un laboratorio), empresariales (organización de proyectos), etc.

Formalmente, el problema JSS requiere planificar un conjunto de n trabajos $\{J_1,..., J_n\}$ en un conjunto de m máquinas o recursos $\{R_1,..., R_m\}$. Cada trabajo J_i consta de una secuencia de operaciones o tareas $(\theta_{i1},..., \theta_{im})$, donde cada tarea θ_{il} necesita el uso de un recurso $R_{\theta_{il}}$ durante un tiempo de procesamiento $p_{\theta_{il}}$. El objetivo es asignar a cada operación un tiempo de inicio $st_{\theta_{il}}$ de modo que se optimice una determinada función objetivo satisfaciendo tres tipos de restricciones: i) Las operaciones de un mismo trabajo deben procesarse en el orden dado en la secuencia, ii) las máquinas tienen capacidad para procesar una única operación en un instante dado y iii) no se permite interrumpir el procesamiento de una operación una vez que ha comenzado.

En la literatura, la función objetivo es habitualmente la minimización de una métrica de la planificación. Las más estudiadas son las siguientes:

- ***Makespan***, o tiempo de finalización de todos los trabajos.
- ***Tiemplo de flujo total***, definido como la suma de los tiempos de finalización de todos los trabajos. En ocasiones, se considera adicionalmente un peso para cada trabajo, denominándose en este caso la función objetivo como *tiempo de flujo total ponderado*.
- ***Lateness máximo***, definido como el máximo retraso de los trabajos con respecto a su límite preestablecido (*due date*).
- ***Lateness total***, o suma de los retrasos de los trabajos con respecto a su *due date*. Si un trabajo se completa antes de su límite, su retraso cuenta como negativo.
- ***Tardiness total***, definido como la máxima tardanza de los trabajos con respecto a su *due date*. En este caso, si un trabajo se completa antes de su *due date*, su tardanza cuenta como 0, y no como un número negativo.

En esta tesis, nos centramos en el problema JSS con la minimización del makespan y del tiempo de flujo total. Estas variantes del problema se denotan respectivamente como $J||C_{max}$ y $J||\sum C_i$ en la notación $\alpha|\beta|\gamma$ propuesta en [32].

El problema JSS es un paradigma de los problemas de optimización con restricciones (*constraint optimization*), y destaca como uno de los más difíciles de la clase de problemas NP-hard. Por este motivo, ha sido objeto de un estudio exhaustivo y en las últimas décadas se han propuesto numerosas técnicas de distinta naturaleza para su resolución.

Debido a su gran dificultad ha sido común emplear metaheurísticas que, a pesar de que no garantizan encontrar una solución óptima ni pueden demostrar cotas inferiores, suelen proporcionar muy buenas planificaciones en un tiempo reducido. Una buena introducción a este tipo de algoritmos se puede encontrar en [73]. Algunos ejemplos notables son algoritmos genéticos que emplean esquemas de codificación de cromosomas y operadores de cruce sofisticados [9], así como algoritmos de decodificación basados en esquemas de generación de planificaciones como el algoritmo *G&T* [27]. Otro tipo de algoritmos muy efectivos son los métodos de búsqueda local. Estos algoritmos explotan estructuras de vecindad muy eficientes basadas en ideas originales propuestas en [74] y [20]. Actualmente, los algoritmos de búsqueda tabú presentados en [54] y [81] se encuentran entre los más efectivos para el problema JSS. En los últimos años, se han logrado éxitos muy notables aplicando estrategias híbridas. Por ejemplo, en [75] se propone un algoritmo memético (que combina un algoritmo genético con búsqueda local) para una variante del problema JSS que considera tiempos de puesta a punto (*setup*) dependientes de la secuencia, obteniendo los mejores resultados conocidos para el problema.

También ha habido mucho interés en resolver el problema de forma exacta. A este respecto destacan los algoritmos de búsqueda y de razonamiento basado en restricciones (*constraint-based reasoning*). Una buena introducción a estos se puede encontrar en [5]. La efectividad de los métodos exactos recae en una combinación apropiada de técnicas de búsqueda con métodos de propagación de restricciones, que tienen la capacidad de reducir el espacio de búsqueda en gran medida. Algunos ejemplos de este tipo de métodos son *edge-finding*, *timetabling* o *energetic reasoning* [17, 21, 41]. En estos algoritmos, una estrategia de búsqueda muy empleada ha sido *backtracking search*, y se han propuesto heurísticos de ordenación de variables y valores eficientes, como los basados en texturas [8] o los basados en holguras [68]. En este contexto, el algoritmo de ramificación y poda propuesto por P. Brucker et al. en [12] destaca como uno de los más eficaces para resolver de forma óptima instancias del problema de tamaño pequeño y medio. Este algoritmo ha sido utilizado como método base para desarrollar nuevas técnicas. Por ejemplo, en [71] el problema JSS se plantea como una serie de problemas de decisión, y se propone un orden adecuado para resolver dichos problemas empleando el algoritmo de P. Brucker en cada uno de ellos. También, en [70], se utiliza para calcular soluciones óptimas de instancias aleatorias en un estudio sobre la estructura del espacio de soluciones del problema. En [26], se propone un método de cálculo de cotas inferiores que emplea algoritmo de P. Brucker para resolver una relajación no polinómica del problema.

Los métodos exactos han servido de base de algunas técnicas metaheurísticas. Algunos ejemplos relevantes son el procedimiento *Shifting Bottleneck* [1], o la técnica *Iterative Flattening Search* [18]. También, en los últimos años, se han propuesto algoritmos que combinan algoritmos exactos con metaheurísticas, como por ejemplo los presentados en [69] y [79].

Muchos de los métodos anteriores, tanto exactos como aproximados, emplean una representación del problema basada en el modelo de grafo disyuntivo [60]. En la Figura 1.1 se muestra un ejemplo para una instancia con tres trabajos y tres máquinas. En el grafo disyuntivo, hay un nodo por cada operación, además de dos nodos ficticios (*start* y *end*) conectados a las primeras y a las últimas tareas de cada trabajo respectivamente. Las restricciones de precedencia entre operaciones del mismo trabajo se representan mediante arcos conjuntivos (en línea continua en la Figura 1.1), y las restricciones de capacidad se representan mediante

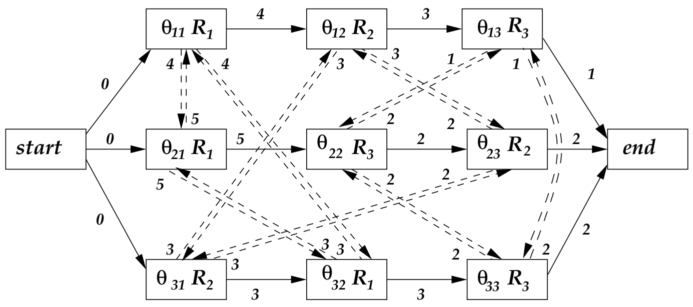


Figura 1.1: Grafo disyuntivo de una instancia con 3 trabajos y 3 máquinas.

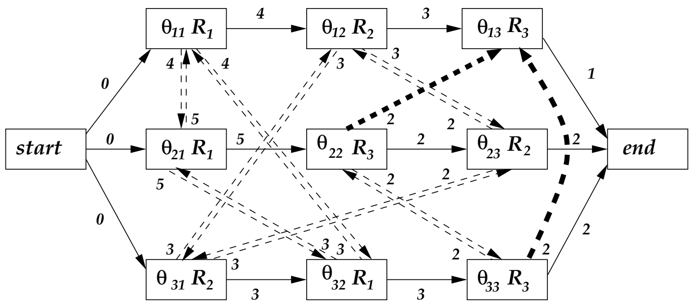


Figura 1.2: Grafo disyuntivo de una planificación parcial para la instancia de la Figura 1.1.

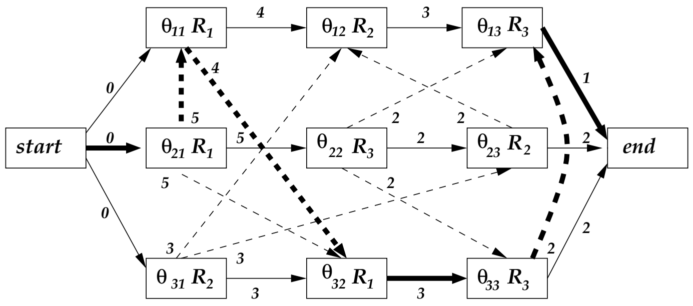


Figura 1.3: Grafo disyuntivo de una planificación factible para la instancia de la Figura 1.1.

arcos disyuntivos (en línea discontinua en la Figura 1.1).

La resolución del problema radica en decidir el orden de procesamiento de las operaciones en cada máquina. Esto se realiza fijando arcos disyuntivos en el grafo. Por ejemplo, en la Figura 1.2 se muestra una planificación parcial para la instancia de la Figura 1.1. En ella, están fijados los arcos (θ_{22}, θ_{13}) y (θ_{33}, θ_{13}). Algunos métodos constructivos necesitan calcular *cabezas* y *colas* de las operaciones en una planificación parcial, que son, respectivamente, cotas inferiores del tiempo más temprano posible de procesamiento de una operación y del tiempo que transcurre desde el fin de una operación hasta que se completa toda la planificación. En un grafo disyuntivo, las cabezas y las colas se obtienen eficientemente mediante algoritmos de cálculo del camino más largo desde el nodo *start* hasta cada operación y desde cada operación al nodo *end*.

Fijando arcos disyuntivos sucesivamente sin generar ciclos en el grafo, se llega a una solución factible. La Figura 1.3 muestra una planificación para la instancia de la Figura 1.1, alcanzable desde la planificación parcial representada en la Figura 1.2. En una planificación factible, las cabezas de las operaciones representan sus tiempos de inicio. En esta figura se marcan los arcos del camino de mayor coste desde el nodo *start* hasta el nodo *end*. Este recibe el nombre de *camino crítico*. En el caso del *makespan*, el coste de un camino crítico es igual al *makespan* de la planificación. También tiene importancia el concepto de *bloque crítico*, definido como una subsecuencia maximal de operaciones de un camino crítico que requieren el mismo recurso. Por ejemplo, en la Figura 1.3 hay dos bloques críticos, $(\theta_{21}, \theta_{11}, \theta_{32})$ y $(\theta_{33}, \theta_{13})$. Algunos esquemas de ramificación de algoritmos exactos, así como la mayoría de las estructuras de vecindad utilizadas en los algoritmos de búsqueda local, se basan en cambiar el orden de las operaciones de un bloque crítico.

Esta tesis se enmarca en las técnicas exactas de optimización. En concreto, siguiendo el paradigma de la búsqueda heurística [56] y apoyándose en los métodos conocidos, hemos planteado el objetivo de diseñar algoritmos exactos que sean capaces de certificar soluciones óptimas para instancias de hasta un cierto tamaño y de calcular cotas inferiores y soluciones de calidad para instancias más difíciles, todo ello con un consumo de tiempo y memoria razonables.

El interés de las cotas inferiores se puede justificar fácilmente: en la práctica, las cotas inferiores pueden dar una medida de la calidad de las soluciones de que se dispone, y ayudar así en la toma de decisiones. Por otra parte, su cálculo equivale a demostrar la no existencia de una solución a un problema de satisfacción de restricciones. Estos casos tienen una gran importancia, ya que puede ser muy interesante saber que dado un problema, no se puede llegar a una solución que esté por debajo de la cota inferior. Asimismo, su cálculo puede ser muy difícil computacionalmente, por lo que constituye un buen objetivo para mejorar las técnicas de resolución conocidas.

Asimismo, siguiendo una tendencia de la comunidad investigadora en scheduling, se considerarán variantes del problema más cercanas a casos reales, además del problema Job Shop Scheduling clásico.

Esta tesis doctoral se presenta como un compendio de publicaciones. El resto de la memoria se estructura de la siguiente forma: En el capítulo 2 se describen los objetivos de la presente tesis. El capítulo 3 contiene una discusión de las contribuciones realizadas y de los resultados obtenidos. En el capítulo 4 se presentan las conclusiones y se esbozan algunas líneas de trabajo futuro. El capítulo 5 contiene una copia original de los artículos presentados como compendio de publicaciones. En el capítulo 6 se presenta un informe sobre el factor de impacto de las publicaciones. Por último, se lista la bibliografía utilizada en esta memoria.

Capítulo 2

OBJETIVOS

En los últimos años, la investigación en técnicas exactas de resolución de problemas de optimización con restricciones, en particular de problemas de scheduling, se ha centrado en el uso de estrategias de búsqueda capaces de hallar buenas soluciones en un tiempo limitado. Para ello, se ha sacrificando en gran medida la capacidad de los algoritmos de calcular cotas inferiores y de demostrar la optimalidad de una solución. Nuestra hipótesis de partida es que, mediante un estudio riguroso de las particularidades de este tipo de problemas y del modo de explotarlas eficazmente, es posible diseñar algoritmos de búsqueda eficientes capaces de calcular tanto cotas inferiores como soluciones de calidad en un tiempo y con un consumo de memoria razonables. De este modo, el objetivo general de esta tesis es contribuir a la mejora de las técnicas exactas de búsqueda y su aplicación a problemas de scheduling de distinta naturaleza. De forma más concreta, proponemos los siguientes objetivos específicos:

- Identificar las limitaciones y las ventajas que tienen las técnicas de búsqueda heurística y de razonamiento basado en restricciones para los problemas de scheduling.
- Estudiar modos eficaces de explotar las particularidades de este tipo de problemas para mejorar los algoritmos exactos de resolución.
- Diseñar estrategias generales de búsqueda que sean eficaces tanto en el cálculo de soluciones como de cotas inferiores en un tiempo limitado y con recursos de memoria razonables.
- Diseñar algoritmos exactos específicos para el problema Job Shop Scheduling con minimización del makespan. Este problema es un paradigma de los problemas de scheduling y está considerado como uno de los más complejos de la clase NP-hard, por lo que constituye un buen banco de prueba para las técnicas desarrolladas.
- Diseñar algoritmos eficaces de búsqueda heurística para problemas que incluyan restricciones complejas derivadas de ámbitos de aplicación reales. En este caso, consideraremos el problema Job Shop Scheduling con Operarios. Al ser un problema nuevo en la literatura, será necesario diseñar cada una de las componentes específicas de los algoritmos de búsqueda (espacio de búsqueda, estimaciones heurísticas, estrategias de poda, etc.).

Capítulo 3

DISCUSIÓN DE RESULTADOS

En este capítulo se discuten los principales resultados obtenidos en esta tesis. Como ya hemos adelantado, hemos considerado dos problemas: el Job Shop Scheduling clásico (JSP) y el Job Shop Scheduling con Operarios (JSO) que es una extensión del anterior propuesta recientemente en la literatura. Algunas contribuciones constituyen métodos específicos para dichos problemas. Otras, como la definición de nuevas estrategias de búsqueda heurística, tienen un carácter más general, pero han sido evaluadas sobre los citados problemas. Por esta razón, el capítulo se divide en dos secciones, cada una de ellas dedicada a un problema.

3.1. El problema Job Shop Scheduling

La primera parte de la investigación se ha centrado en el diseño de nuevas estrategias de búsqueda heurística y su aplicación al problema JSS. Así, en esta sección, consideramos en primer lugar la aplicación de dos variantes de la estrategia de búsqueda en profundidad al problema JSS y describimos el heurístico denominado h_{IS} que constituye una de las principales aportaciones de la tesis. Posteriormente presentamos una nueva estrategia de búsqueda, denominada *Intensified Iterative Deepening A**, que es otra de las aportaciones importantes de la tesis, y su aplicación al mismo problema.

3.1.1. Búsqueda en profundidad parcialmente informada y el heurístico h_{IS}

Antecedentes e hipótesis

Debido a su naturaleza, la resolución exacta de problemas de scheduling y de optimización con restricciones se ha abordado tradicionalmente mediante algoritmos de búsqueda en árboles, cuyas hojas representan soluciones potenciales y se encuentran a una profundidad acotada, en ocasiones conocida de antemano.

La inexistencia de ramas de longitud infinita en estos espacios de búsqueda favorece enormemente la aplicación de estrategias basadas en búsqueda primero en profundidad (*Depth-First Search*, o DFS). La búsqueda en profundidad es muy simple conceptualmente y tiene un consumo de memoria lineal en la máxima profundidad del árbol de búsqueda, lo que posibilita que, en la práctica, se ejecute durante largos periodos de tiempo. Además, en el contexto de los problemas de optimización, se trata de una estrategia de tipo *anytime*, es decir, puede disponer de una solución sub-óptima en un momento temprano de la búsqueda. Posteriormente, a medida que ésta avanza, se van encontrando mejores soluciones hasta haber recorrido todo el árbol, momento en que finaliza certificando una solución óptima.

El hecho de disponer de soluciones a lo largo de la búsqueda no sólo presenta evidentes ventajas cuando hay un tiempo limitado para la resolución de un problema, sino que puede ser explotado para mejorar la eficiencia de los algoritmos de búsqueda. Así, una práctica común es emplear el coste de la mejor solución encontrada hasta el momento (UB) para podar los nodos tales que una cota inferior de la mejor solución alcanzable desde ellos sea mayor o igual que UB. Esto constituye un elemento distintivo de las técnicas de ramificación y poda. Asimismo, los métodos de propagación de restricciones, que permiten simplificar el problema representado por cada nodo del árbol de búsqueda mediante razonamiento deductivo, suelen ser más eficaces cuanto menor sea UB. Por estos motivos, DFS es la base de muchas otras estrategias de búsqueda muy conocidas, como *Iterative Deepening A** (IDA*) [38], *Limited Discrepancy Search* (LDS) [36] y sus variantes, o estrategias basadas en DFS con reinicios y aleatoriedad de los heurísticos, como la exitosa *Solution Guided Search* [7].

Un elemento distintivo de las técnicas de inteligencia artificial es su capacidad para explotar el conocimiento del dominio del problema mediante el uso de información heurística. Es en este aspecto donde DFS presenta su mayor inconveniente. Para preservar un consumo de memoria tan bajo, sólo puede emplear heurísticos a nivel local, en la decisión de qué sucesor de un nodo dado se explorará primero. Aún así, este uso de información heurística puede mejorar en gran medida la efectividad de este tipo de algoritmos.

En la literatura, la mayor parte de las propuestas de resolución exacta de problemas de scheduling proviene de los campos de la investigación operativa y del razonamiento basado en restricciones (*constraint-based reasoning*). En ellas ha sido muy común el empleo de una variante de la búsqueda en profundidad conocida como *backtracking search*. Ésta se caracteriza por que en cada paso de la búsqueda, en lugar de generar todos los sucesores de un nodo n, sólo genera uno de ellos, almacenando la información necesaria para generar los demás una vez que se ha desarrollado todo el subárbol del que n es raíz. En estos casos, la elección del sucesor a explorar primero en la búsqueda es el resultado de un análisis del propio nodo n y de las reglas que dan lugar a los sucesores, empleando generalmente reglas de prioridad o heurísticos más sofisticados. Ejemplos relevantes de este tipo de heurísticos para el problema Job Shop Scheduling son los basados en holguras [68] o los basados en texturas [62, 8].

La hipótesis de partida de esta investigación es que, aprovechando el funcionamiento y la interacción entre las distintas componentes que conforman los algoritmos modernos de búsqueda en árboles, se puede explotar el conocimiento del dominio del problema más eficazmente en el marco de la búsqueda heurística clásica. Asimismo, una mayor compresión y mejora de la estrategia de búsqueda en profundidad en el contexto de los problemas de scheduling puede conllevar una mejora de otras estrategias basadas en ella.

Contribuciones y resultados

El punto de partida, de la línea de trabajo encaminada a mejorar las estrategias de búsqueda en profundidad, fue el conocido método de ramificación y poda propuesto en [12, 11] por P. Brucker et al. para resolver el problema Job Shop Scheduling con minimización de makespan. Éste constituye un buen ejemplo de algoritmo de ramificación y poda, al contar con la mayor parte de los elementos comúnmente empleados en la resolución de problemas complejos de optimización. A pesar de haber sido propuesto hace ya bastante tiempo, sigue siendo uno de los más eficaces para resolver de forma óptima instancias de tamaño pequeño y medio. En su ejecución busca soluciones en un árbol n-ario que representa el espacio de búsqueda de las planificaciones parciales. En dicho espacio de búsqueda, cada nodo se corresponde con un grafo disyuntivo en el que algunos arcos disyuntivos están fijados estableciendo relaciones de precedencia entre operaciones que requieren la misma máquina. La resolución del problema representado por un nodo consiste en añadir sucesivamente nuevos arcos disyuntivos sin producir ciclos en el grafo hasta completar una planificación. Su éxito radica en un uso equilibrado de las siguientes componentes:

- Un algoritmo eficiente de cálculo de soluciones que se aplica a partir de cada nodo del árbol de búsqueda. En concreto, se emplea una versión del conocido algoritmo $G\&T$ [27] que construye en tiempo polinómico una solución factible al problema respetando los arcos disyuntivos fijados en cada nodo.

- Un esquema de ramificación que explota eficazmente propiedades del problema para calcular estados sucesores. En concreto, a partir de cada nodo n, emplea el algoritmo $G\&T$ para obtener una solución factible S_n y calcula un camino crítico de S_n y los bloques críticos contenidos en él. Posteriormente, considera todos los posibles movimientos que conducen a que una operación se ejecute antes o después que todas las demás operaciones del mismo bloque crítico. Cada opción da lugar a un nuevo sucesor al que se añaden los arcos disyuntivos correspondientes a los que ya están fijados en n.

- Un método de cálculo de cotas inferiores basado en la relajación de las restricciones de no interrupción y de las restricciones de capacidad de todas las máquinas salvo de una. Cada máquina da lugar a una instancia del problema One Machine Sequencing (OMS) con interrupción, que se resuelve de forma óptima en tiempo polinómico mediante la conocida *Jackson's Preemptive Schedule* (JPS) [14]. La cota inferior se calcula como el máximo coste de las soluciones óptimas de los distintos problemas relajados derivados de cada máquina.

- Un método muy eficaz de propagación de restricciones conocido como *selección inmediata* [15, 16, 17]. Dado un nodo n del árbol de búsqueda y una cota superior, UB, este método es capaz de deducir relaciones de precedencia entre operaciones de una misma máquina que deben cumplirse en todas las soluciones que, desde n, tengan un makespan menor que UB. La selección inmediata fue el primer método de propagación de restricciones de tipo *edge-finding* propuesto en la literatura. Otros algoritmos similares pueden encontrarse en [21], [41] y [77].

- Una estrategia de búsqueda de ramificación y poda con backtracking search, que explota las componentes anteriores eficazmente. Hace uso de una pila donde almacena la rama actual en la que se encuentra la búsqueda. La elección del sucesor a generar primero se realiza en función de las cabezas y colas de las operaciones que dan lugar a los sucesores. Una vez generado un sucesor, se aplica el método de selección inmediata y se calcula una cota inferior, LB. Si LB no es menor que el coste de la mejor solución alcanzada hasta el momento, el nodo se poda.

Como hemos comentando, los algoritmos de búsqueda en árboles suelen ser más eficientes cuanto mejores sean las soluciones de que disponen. La forma natural de obtener soluciones de calidad en este contexto es explotar la información del dominio del problema para guiar la búsqueda de forma efectiva hacia regiones prometedoras.

Búsqueda en profundidad parcialmente informada

La primera contribución viene de la observación de que, a este respecto, la estrategia de backtracking search desaprovecha gran parte de la información disponible. Por una parte, elegir qué rama explorar primero en función de una estimación parcial de cómo serán los sucesores a partir de la información del nodo padre y del esquema de ramificación, implica una capacidad limitada de discriminar entre las posibles opciones. Por otra parte, se invierte un tiempo considerable en el cálculo de cotas inferiores que sólo se emplean para podar nodos cuya cota inferior permita demostrar que no conducen a mejorar la solución actual. La búsqueda heurística clásica se caracteriza por el empleo de este tipo de información como elemento central en la toma de decisiones. En este campo, la estrategia de búsqueda en profundidad parcialmente informada [56] constituye una buena alternativa a backtracking search.

Así, se ha empleado esta estrategia para recorrer el espacio de búsqueda generado por el algoritmo de ramificación y poda de P. Brucker. Para ello fue necesario formalizar sus distintas componentes en el marco de la búsqueda heurística.

En primer lugar, la estimación heurística del coste de la mejor solución alcanzable desde un estado n, $f(n)$, la hemos considerado, como es habitual, como la suma $f(n) = g(n) + h(n)$. Donde $g(n)$ representa el coste para llegar hasta el estado n y $h(n)$ una estimación del coste para llegar al objetivo más próximo a n. Hemos definido la función $g(n)$ como el coste de un camino crítico en el grafo disyuntivo de la planificación parcial representada por el estado n. Esta definición presenta la ventaja de que $g(n) = g^*(n)$, es decir es el coste mínimo para llegar al estado n, lo que tiene implicaciones interesantes, como se verá más adelante en esta sección. Asimismo, este modo de calcular $g(n)$ se puede extender a una clase más general de problemas de scheduling con minimización de makespan donde la representación mediante grafos disyuntivos resulte adecuada. La cota inferior de la mejor solución alcanzable desde un nodo n calculada por el algoritmo de Brucker se denota como $h_{JPS}(n)$, ya que requiere el cálculo de varias JPSs y proporciona una estimación consistente, y por tanto admisible, pues proviene de una relajación del problema resuelta de forma óptima.

Posteriormente, analizamos el modo más apropiado de integrar el método de selección inmediata en esta estrategia. Hay dos puntos durante la búsqueda donde es posible aplicar un método de propagación de restricciones: justo antes de la expansión de un nodo o justo después de generarlo. Debido a la potencial capacidad de estos métodos de deducir una gran cantidad de información sobre el sub-problema representado por cada estado, es razonable aprovechar dicha información en la decisión de qué nodo expandir primero. Por lo tanto, hemos optado por emplear el método de selección inmediata justo después de generar un nodo s; una vez aplicado, se calcula $f(s)$, con lo que la nueva información deducida se encuentra reflejada en la estimación heurística.

Por lo tanto, la estrategia de búsqueda en profundidad parcialmente informada funciona del siguiente modo: cuando se expande un nodo n, se generan todos sus sucesores. A partir de cada sucesor s de n, se obtiene un estado simplificado mediante el método de selección inmediata, es decir, $s = IS(s, UB)$. Posteriormente, se calcula $f(s) = g(s) + h_{JPS}(n)$ y, si $f(s) \geq UB$ el estado s se poda. Una vez generados todos los sucesores, se insertan al inicio de la lista ABIERTA ordenados localmente de forma no decreciente de los valores de f. El algoritmo extrae el primer nodo de ABIERTA y repite el mismo procedimiento hasta que esta lista se queda vacía.

Esta estrategia presenta una sobrecarga en la generación de los sucesores pues, a diferencia de backtracking search, es necesario copiar la información contenida en el estado n en cada sucesor s. También presenta un consumo de memoria ligeramente superior, al almacenar los estados sucesores a lo largo de la rama actual, en vez de anotaciones sobre cómo generarlos. Sin embargo, el uso de memoria sólo se multiplica por una constante, siendo lineal en la profundidad del árbol de búsqueda. A cambio, emplea el conocimiento heurístico de un modo más eficaz, tanto en el cálculo de soluciones como en el de cotas inferiores globales.

Justo después de expandir un nodo, se cumple que el menor $f(n)$ de los estados n contenidos en ABIERTA es una cota inferior del coste de la solución óptima. Un resultado teórico importante demostrado en este trabajo, es que si $g(n) = g^*(n)$ y $h(n)$ es monótono, la secuencia de cotas inferiores dadas por el menor $f(n)$ de ABIERTA es no decreciente en el tiempo. Por tanto, en un punto intermedio de la ejecución, basta con realizar este cálculo una vez para obtener la mejor cota inferior calculada hasta el momento.

El heurístico h_{IS}

Una contribución importante ha sido la formalización de un nuevo heurístico, denotado como h_{IS}, que explota la propagación de restricciones a modo de demostración por reducción al absurdo. Esta idea, esbozada originalmente en [16], se ha aplicado con anterioridad para el cálculo de cotas inferiores iniciales del problema, dado el alto tiempo computacional de los métodos de propagación de restricciones. Además, ha inspirado la técnica de *shaving* [45], que conduce a reducciones dramáticas en las ventanas de tiempo de las operaciones en una planificación parcial (cabezas y colas).

El cálculo del heurístico se basa en el siguiente razonamiento: dado un nodo del espacio de búsqueda n y una cota superior ficticia P, la aplicación del procedimiento de selección inmediata añade nuevos arcos disyuntivos a n, produciendo un nuevo estado $n_P = IS(n, P)$. Si n_P es inconsistente, significa que no existe desde n una solución con coste menor que P, luego P sería una cota inferior de n. El nodo n_P es inconsistente si su grafo disyuntivo contiene algún ciclo o si $f_{JPS}(n_P) \geq P$.

La estimación heurística $f_{IS}(n)$ se define como el mayor valor P tal que $IS(n, P)$ produce un estado n_P inconsistente. Por definición, el valor $P = f_{JPS}(n)$ da lugar a un estado inconsistente y, al provenir n de la aplicación del método de selección inmediata con la cota UB, el valor $P = UB$ produce el mismo estado n consistente. La demostración formal de que si $n_P = IS(n, P)$ es consistente, entonces el estado $n_{P-1} = IS(n, P-1)$ también lo es para cualquier estado n del espacio de búsqueda, conduce al cálculo eficiente de $f_{IS}(n)$ por medio de una búsqueda dicotómica en el intervalo $[f_{JPS}(n), UB-1]$.

En un estudio formal se ha demostrado que el nuevo heurístico $h_{IS}(n)$, definido como $h_{IS}(n) = f_{IS}(n) - g(n)$, cumple la propiedad de monotonía. Esta propiedad, además de presentar la ventaja para el cálculo de cotas inferiores globales descrita anteriormente, permite restringir aún más la búsqueda dicotómica al intervalo $[\text{máx}\{f_{IS}(p), f_{JPS}(n)\}, UB-1]$, con $n \in SUC(p)$, es decir, n es un sucesor de p, haciendo así su cálculo más eficiente.

Aunque empíricamente se ha observado que h_{IS} es un estimador más eficaz que h_{JPS} independientemente de la profundidad del árbol de búsqueda, también es más costoso computacionalmente. Por ello, hemos propuesto un modo de combinarlos para sacar partido de las ventajas de cada uno. En concreto, proponemos emplear el heurístico h_{IS} hasta una profundidad determinada del espacio de búsqueda, y a partir de ahí, utilizar h_{JPS}.

Evaluación

Un estudio experimental extenso sobre instancias del problema de uso común obtenidas de la OR-Library [6] y otras propuestas por Taillard en [72], ha mostrado que la búsqueda en profundidad parcialmente informada supera ampliamente a backtracking search en la capacidad de certificar soluciones óptimas para instancias de tamaño pequeño y medio, y de calcular cotas inferiores y soluciones aproximadas para instancias grandes a igual tiempo de computación. Se ha observado que el empleo del heurístico más informado h_{IS} es muy beneficioso para esta estrategia, mientras que conduce a un deterioro de la efectividad de backtracking search.

En cuanto la combinación de los dos heurísticos, se ha observado que a medida que el tamaño y dificultad de las instancias crece, es más efectivo emplear h_{IS} hasta una mayor profundidad del árbol de búsqueda. En las instancias más grandes la mejor opción es emplear h_{IS} en todo el espacio de búsqueda. En ellas, la búsqueda en profundidad parcialmente informada dirigida por h_{IS} es capaz de reducir el error medio de las soluciones alrededor del $50\,\%$ y el error de las cotas inferiores en un $30\,\%$ con respecto al algoritmo de ramificación y poda de P. Brucker.

En el capítulo 5 se incluyen las siguientes publicaciones donde se detallan las contribuciones discutidas en esta sección:

- Carlos Mencía, María R. Sierra and Ramiro Varela. Partially Informed Depth-First Search for the Job Shop Problem. *In Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS '10)*. Toronto, Canada. AAAI Press. pp: 113-120. 2010

- Carlos Mencía, María R. Sierra and Ramiro Varela. Depth-first heuristic search for the job shop scheduling problem. *Annals of Operations Research*. 2012. DOI 10.1007/s10479-012-1296-x.

3.1.2. Una nueva estrategia: Intensified Iterative Deepening A*

Antecedentes e hipótesis

La búsqueda en profundidad parcialmente informada explota el conocimiento heurístico disponible de un modo más eficaz que la estrategia de backtracking search, lo que le permite alcanzar mejores soluciones y certificar cotas inferiores más ajustadas a igual tiempo de computación.

Sin embargo, presenta una debilidad común a todas las variantes de la estrategia general de búsqueda en profundidad: su carácter sistemático hace que una decisión equivocada acerca de qué sucesor expandir primero le pueda conducir a estancarse en una región mediocre del árbol de búsqueda, lo que tiene efectos muy negativos en su rendimiento. Al elegir un sucesor s que no conduce a buenas soluciones, el algoritmo de búsqueda necesita explorar todo el subárbol por debajo de s antes de tener la posibilidad de elegir otro sucesor que represente una opción mejor. El problema se agrava por el hecho de que los heurísticos suelen presentar una capacidad muy limitada de discriminación en niveles bajos del árbol de búsqueda, donde cada sucesor es la raíz de un subárbol posiblemente de gran tamaño. Este fenómeno limita la capacidad de DFS de mejorar las soluciones y las cotas inferiores globales en el tiempo; en árboles de búsqueda grandes, la mejor cota inferior que se puede calcular en un tiempo razonable suele ser el valor de la función f para algún sucesor del estado inicial aún no expandido.

En la literatura se han propuesto numerosas técnicas basadas en búsqueda en profundidad encaminadas a mitigar este inconveniente. Algunos ejemplos destacados son *Limited Discrepancy Search* (LDS) [36] ó la búsqueda en profundidad con reinicios y aleatoriedad en los heurísticos [28]. Ambas tratan de diversificar la búsqueda entre regiones prometedoras, para lo que necesitan realizar reinicios y reexpandir algunos estados con el fin de mantener un consumo de memoria bajo. LDS explora primero las ramas del árbol con menor discrepancia heurística. Para ello, ejecuta una serie de búsquedas en profundidad limitadas por una máxima discrepancia heurística, que se incrementa al principio de cada reinicio. Algunas variantes como *Improved Limited Discrepancy Search* (ILDS) [39] o *Depth-Bounded Limited Discrepancy Search* (DBLDS) [78], mejoran su efectividad evitando reexpandir algunos nodos (ILDS) o priorizando ramas con discrepancias heurísticas a niveles bajos del árbol de búsqueda (DBLDS). La búsqueda en profundidad con reinicios y aleatoriedad en los heurísticos realiza una serie de ejecuciones de búsqueda en profundidad, cada una de ellas limitada por un número máximo de ramas exploradas definido por una estrategia de reinicios.

La diversificación de la búsqueda permite a las estrategias anteriores hallar soluciones de alta calidad en un tiempo corto, lo que las hace apropiadas para abordar la resolución de instancias de gran tamaño que no se puedan resolver de forma óptima en un tiempo razonable. Sin embargo, debido a los reinicios, suelen ser menos efectivas que DFS en su capacidad de certificar soluciones óptimas o de calcular cotas inferiores globales. A este respecto, se han propuesto técnicas como *Interleaved Depth-First Search* (IDFS) [53], que intercala la ejecución de distintas búsquedas en profundidad, o la incorporación de estrategias de *no-good recording* en la búsqueda en profundidad con reinicios, que almacenan nodos cuyo subárbol se ha explorado completamente, evitando ser expandidos de nuevo en cualquier otro reinicio del método. IDFS multiplica el consumo de memoria de DFS por una constante igual al número de búsquedas en profundidad activas, y se ha observado experimentalmente que es efectiva cuando hay un cambio en la función heurística desde el primer nivel al segundo del árbol, lo cual no se suele cumplir en los problemas de scheduling. Por otra parte, la técnica de no-good recording presenta un consumo de memoria más elevado y una sobrecarga en el tiempo de computación derivada de su gestión.

Una estrategia con mucho éxito en la resolución de problemas de optimización combinatoria es *Iterative Deepening A** (IDA*) [38]. Diseñada originalmente con el objetivo de limitar el consumo de memoria del algoritmo A*, emplea de manera iterativa la búsqueda en profundidad para recorrer el espacio de búsqueda a modo de búsqueda *primero el mejor* (*best-first search*). Su funcionamiento consiste en realizar una secuencia de ejecuciones de DFS limitadas por un umbral de coste (CT, *cost threshold*), considerando inicialmente $CT = f(ini)$. En cada iteración, busca en profundidad expandiendo los nodos tales que $f(n) \leq CT$, y descartando aquellos nodos n con $f(n) > CT$. Si encuentra un nodo que representa una solución, el algoritmo termina, habiendo certificado una solución óptima, con coste $C^* = CT$. Si por el contrario recorre todo el espacio de búsqueda sin haber encontrado una solución, el mínimo valor de f de los nodos descartados constituye una cota inferior global al problema; toma dicho valor como CT para la siguiente iteración, y reinicia la búsqueda.

IDA* es una estrategia adecuada para problemas de optimización combinatoria de tipo *shortest-path*, como puzles o problemas de planning, que definen espacios de búsqueda con estructura de grafo donde las soluciones se encuentran a una profundidad no conocida y presentan ciclos y diversos caminos que conducen a cada nodo desde el estado inicial. En el contexto de la búsqueda en árboles, no explota la estructura del espacio de búsqueda, por lo que el hecho de necesitar expandir todos los nodos n con $f(n) < C^*$ y alguno con $f(n) = C^*$ antes de llegar a una solución, hace que en la práctica no sea apropiada para resolver problemas de cierta dificultad. Sin embargo, al resolver de forma secuencial una serie de problemas de decisión, puede retornar cotas inferiores no triviales en un tiempo de computación inferior a técnicas como DFS, lo cual es una propiedad deseable en un algoritmo exacto.

En base a lo anterior, hemos partido de la hipótesis de que aprovechando el funcionamiento del algoritmo IDA* e integrando en él distintos elementos específicos de la resolución de problemas de optimización con restricciones, es posible obtener una estrategia que calcule buenas cotas inferiores y al mismo tiempo diversifique la búsqueda entre regiones prometedoras, logrando así obtener buenas soluciones en un tiempo limitado.

Contribuciones y resultados

En [51] proponemos una nueva estrategia de búsqueda que extiende a IDA* para abordar eficazmente problemas de búsqueda en árboles. La nueva estrategia, llamada *Intensified Iterative Deepening A** (*i*IDA*) ha sido diseñada con el objetivo de alcanzar soluciones de alta calidad y, al mismo tiempo, calcular cotas inferiores ajustadas en un tiempo limitado. Para ello, extiende a IDA* mediante la introducción de tres elementos: la aplicación de un método de propagación de restricciones, una estrategia liberal de actualización del umbral del coste y la combinación con búsquedas en profundidad limitadas por un número de expansiones.

Aplicación no conservativa de propagación de restricciones

En su descripción original, IDA* no contemplaba el uso de métodos de propagación de restricciones. Sin embargo, como se ha comentado anteriormente, este tipo de métodos constituyen una de las claves en la resolución de problemas de scheduling y de optimización con restricciones.

A este respecto, *i*IDA* aplica la propagación de restricciones tomando como cota del método el valor de CT en cada iteración. De este modo, al ser CT una cota inferior de C^*, el método de propagación de restricciones es mucho más efectivo reduciendo el espacio de búsqueda que si empleara una cota superior para tal cometido. Sin embargo, salvo en la última iteración, donde CT es también cota superior de C^*, es posible que algunos nodos que conducen a alguna solución óptima se poden mediante este método, por lo que *i*IDA* no conserva la propiedad de que el menor valor de f de entre los nodos descartados durante la búsqueda sea una cota inferior global al problema. En consecuencia, para garantizar la admisibilidad del método, al finalizar una iteración intermedia sólo puede actualizar la cota inferior global a $CT + 1$, y CT a $CT + 1$ para la siguiente iteración.

Por lo tanto, en principio *i*IDA* debe realizar más iteraciones que IDA*, pero en cada una de ellas el espacio de búsqueda se reduce en mayor medida que si se tomara una cota superior de C^* en el método de propagación de restricciones. Experimentalmente, se ha observado que en el problema Job Shop Scheduling con minimización del makespan, la mayor parte de las veces la estrategia IDA* clásica incrementa el valor de CT en una unidad entre dos iteraciones consecutivas, por lo que en la práctica, *i*IDA* no requiere de un número mucho mayor de iteraciones. En cualquier caso, el siguiente elemento de *i*IDA* contribuye a reducirlo en gran medida.

Una nueva estrategia de actualización del umbral de coste

Antes de cada iteración, IDA* actualiza el valor de CT a una cota inferior de C^*. Así, garantiza que la primera solución encontrada durante la búsqueda sea óptima y presenta la ventaja de que nunca expande estados cuyo valor de f supere a C^*. Sin embargo, debido a los reinicios, en cada iteración debe volver a expandir los nodos desarrollados en la iteración anterior, lo que introduce una sobrecarga en términos de tiempo de computación.

Con el fin de reducir el trabajo repetido, *i*IDA* actualiza el valor de CT de un modo liberal entre dos iteraciones consecutivas. En concreto se basa en la estrategia DFS* [76]: Comenzando con $CT = f(ini)$, siendo *ini* el estado inicial, cuando una iteración intermedia finaliza sin haber hallado una solución duplica el valor de CT para la siguiente iteración. Puede ocurrir que CT llegue a superar el valor de C^*, por lo que cuando en una iteración se alcanza una solución al problema ésta puede no ser óptima. Por esta razón, para poder certificar soluciones óptimas, después de encontrar una solución, *i*IDA* continúa el recorrido del espacio de búsqueda mediante una estrategia clásica de ramificación y poda hasta el final.

En los problemas de scheduling, C^* suele ser mucho menor que $2 \times f(ini)$, por lo que en la práctica aplicar esta estrategia no presenta ventajas (salvo cuando $f(ini) = C^*$). Al cambiar a modo de ramificación y poda de forma prematura, el árbol de búsqueda puede ser de gran tamaño. Una contribución del trabajo ha sido la propuesta de una nueva estrategia de actualización del umbral de coste más adecuada para estos problemas que, aunque hace crecer a CT más lentamente, garantiza un número máximo de iteraciones de *i*IDA* logarítmico en la diferencia $C^* - f(ini)$. Con esta estrategia, cuando CT llega a superar a C^*, la diferencia no es tan grande como en el caso anterior, lo que hace que la última iteración de *i*IDA* se lleve a cabo en un árbol de búsqueda más reducido.

Combinación con búsquedas en profundidad limitadas

La estrategia IDA* no encuentra una solución al problema hasta la última iteración. Esto constituye su principal desventaja con respecto a otras técnicas de búsqueda en árboles, ya que si el tamaño de la instancia es grande, puede ocurrir que IDA* no sea capaz de calcular una solución en un tiempo razonable.

El tercer elemento que caracteriza a *i*IDA* resuelve este problema. Atendiendo al funcionamiento de IDA*, en cada iteración intermedia, las hojas del árbol de búsqueda efectivo (aquellas que no tienen ningún sucesor s con $f(s) \leq CT$), definen una frontera a lo ancho del árbol de búsqueda. Estos estados representan zonas del espacio de búsqueda similarmente prometedoras, pues en muchas ocasiones tienen valores de f iguales o muy similares. Por lo tanto, muestreando algunas de estas regiones en busca de soluciones, se obtiene un modo efectivo de diversificar la búsqueda haciendo posible que se encuentren soluciones de calidad rápidamente.

Para este cometido, *i*IDA* ejecuta búsquedas en profundidad limitadas por un número máximo de expansiones desde algunos de los estados de la frontera en cada iteración. Estas ejecuciones de DFS se denominan *intensificaciones*. En ellas, DFS utiliza el valor de la mejor solución encontrada hasta el momento tanto en el método de propagación de restricciones como para podar nodos n con $f(n) \geq UB$. Con el fin de que no consuman demasiado tiempo de ejecución, las intensificaciones están limitadas por un número de expansiones igual al doble de la distancia desde cada nodo a la primera hoja. Así, se asegura que puedan llegar a la máxima profundidad del árbol y compensar dicho esfuerzo permaneciendo a dicha profundidad un tiempo equivalente con la esperanza de mejorar la solución actual. Además, para avanzar en las distintas iteraciones del método y diversificar la búsqueda más eficazmente, *i*IDA* equilibra los dos modos de búsqueda llevando cuenta del número de nodos expandidos en cada uno de ellos. Sólo lanza una intensificación cuando alcanza un estado en la frontera y se han expandido menos estados en modo de intensificación que en modo IDA*.

Evaluación

La estrategia *i*IDA* ha sido evaluada en el problema Job Shop Scheduling con minimización de makespan. Se ha implementado un algoritmo de este tipo que toma como base el algoritmo de búsqueda en profundidad parcialmente informada propuesto en [48, 49], discutido en la sección anterior.

Una primera evaluación de los componentes de *i*IDA* muestra que los tres nuevos elementos contribuyen a la eficiencia de la nueva estrategia, y su combinación presenta un efecto sinérgico. En una comparación experimental, se ha mostrado que *i*IDA* es capaz de encontrar soluciones óptimas a instancias de tamaño medio y de certificar su optimalidad en un tiempo muy corto, menor que DFS, LDS e IDA*. En instancias grandes, *i*IDA* calcula soluciones significativamente mejores que las tres estrategias anteriores en el mismo tiempo de computación (1 hora). Asimismo, es capaz de superar a DFS y a LDS en el cálculo de cotas inferiores, siendo estas comparables a las de la estrategia IDA*.

Por todo ello, *i*IDA* es una estrategia de búsqueda que cubre uno de los objetivos principales de la tesis doctoral. Mediante una combinación original de los distintos elementos de la búsqueda se ha obtenido una estrategia que diversifica la búsqueda para hallar soluciones de alta calidad en un tiempo limitado. Para tal efecto, no sólo no sacrifica la capacidad de DFS de calcular cotas inferiores y de certificar soluciones óptimas para instancias hasta cierto tamaño, como hacen otras estrategias, sino que la mejora sustancialmente.

En el capítulo 5, se encuentra la siguiente publicación, donde se recogen de forma detallada las contribuciones presentadas en este apartado:

- Carlos Mencía, María R. Sierra and Ramiro Varela. Intensified iterative deepening A* with application to job shop scheduling. *Journal of Intelligent Manufacturing* 2013. DOI 10.1007/s10845-012-0726-6

3.2. El problema Job Shop Scheduling con Operarios

Antecedentes e hipótesis

El interés práctico del problema JSS es indudable, ya que numerosas actividades se desarrollan en entornos de producción de tipo job shop. Por ejemplo, en [46], se modela una planta de producción de acero como un job shop simple y se propone un algoritmo de búsqueda tabú para calcular planificaciones. Sin embargo, la simpleza de su formulación obliga a asumir como insignificantes características específicas y restricciones presentes en diversos entornos reales, como tiempos de *setup* dependientes de la secuencia [75], incertidumbre en los tiempos de procesamiento de las operaciones [31, 59] o restricciones de bloqueo entre los trabajos [55]. La incorporación de dichas características en el modelo puede tener un gran impacto en la calidad y significancia de las planificaciones. Por este motivo, extender el problema JSS y otros problemas relacionados a versiones más generales ha sido una prioridad para la comunidad de investigación en scheduling en las últimas décadas.

Recientemente, ha crecido el interés en la consideración de características derivadas de la presencia de operarios humanos en los entornos productivos. Estos representan un recurso crítico en numerosas situaciones [82] y dan lugar a restricciones complejas que pueden incrementar la dificultad de los problemas significativamente. Por ejemplo, en [33] se considera el problema de planificar turnos de trabajadores especializados que participan en un entorno productivo de tipo job shop. El objetivo es minimizar los costes de asignar trabajadores a turnos y a máquinas dentro de cada turno de modo que el makespan de la solución al JSS subyacente no supere una cota predefinida. En [80] se consideran problemas de scheduling donde las operaciones requieren el uso de diversos recursos simultáneamente, algunos de ellos trabajadores humanos. En [61] se modela un JSS real en la industria farmacéutica, donde los recursos son técnicos con habilidades diferentes y sólo un subconjunto de ellos puede trabajar en paralelo.

Un objetivo de la presente tesis doctoral es contribuir a la resolución de problemas de scheduling más cercanos a casos reales. A este respecto se ha optado por abordar el problema Job Shop Scheduling con Operarios, presentado en [Agnetis et al. 2011]. Este problema extiende el JSS contemplando un número limitado de operarios humanos que deben asistir el procesamiento de las operaciones en las máquinas.

Formalmente, está definido por un conjunto de n trabajos $\{J_1, ..., J_n\}$, un conjunto de m máquinas $\{R_1, ..., R_m\}$ y un conjunto de p operarios $\{O_1, ..., O_p\}$. El trabajo J_i consta de una secuencia de v_i operaciones $(\theta_{i1}, ..., \theta_{iv_i})$, donde cada tarea θ_{ij} requiere el uso ininterrumpido de un recurso R_{ij} y la asistencia de cualquiera de los operarios durante una duración positiva entera p_{ij}. El objetivo es asignar a cada tarea u un tiempo de inicio st_u y un operario O_u satisfaciendo las siguientes restricciones de modo que se minimice una función objetivo determinada: i) las operaciones de un trabajo deben ejecutarse en el orden dado en la secuencia, ii) las máquinas no tienen capacidad para procesar más de una operación al mismo tiempo, iii) cada operación necesita la asistencia un operario para su ejecución, y un operario no puede asistir a más de una operación a la vez y iv) no se permite la interrupción de una operación en ningún momento intermedio de su procesamiento.

En [2] el problema se denota como JSO(n, p), y se considera como función objetivo la minimización del makespan. Se demuestra que pertenece a la clase de problemas NP-hard incluso en el caso particular en donde se fijen $n = 3$ y $p = 2$, y se propone un método exacto de programación dinámica y dos algoritmos heurísticos para resolverlo. El problema JSS es un caso particular del JSO(n, p) siempre que $p \geq \min n, m$.

El JSO(n, p), puede verse como un JSS en donde no puede haber más de p operaciones procesándose en paralelo en ningún momento de la planificación.

Durante el desarrollo de la investigación se ha estudiado el problema JSO(n, p) considerando tanto el makespan como el tiempo de flujo total como funciones objetivo. Cada una es representativa de una clase general de funciones objetivo: el makespan es una función de tipo máximo o *cuello de botella*, mientras que el tiempo de flujo total, definido como la suma de los tiempos de finalización de los trabajos, es aditiva. En la literatura se ha estudiado con mucha más profusión el primer caso, dado que presenta un evidente interés práctico y su naturaleza de tipo máximo hace más fácil explotar la estructura del problema de una forma eficaz. Sin embargo, en diversos entornos productivos, las funciones de tipo suma pueden ser más útiles desde un punto de vista práctico [24, 43]. A la vez, minimizar el tiempo de flujo total suele incrementar la dificultad de los problemas de scheduling [30, 13].

En esta sección, se detallan los resultados obtenidos en la minimización del tiempo de flujo total y se mencionan brevemente algunas contribuciones para el makespan.

El problema JSO es nuevo en la literatura, por lo que es necesario diseñar por primera vez cada uno de los distintos elementos que contribuyen a su resolución. Una metodología común empleada en el campo de scheduling consiste en extender ideas y técnicas eficientes para problemas conocidos como el JSS, de modo que se puedan manejar nuevas restricciones presentes en versiones más generales del problema.

De este modo, nuestra hipótesis de partida es que un estudio riguroso del problema JSO(n, p) y de las semejanzas y diferencias con otros problemas conocidos de scheduling, puede conducir al diseño de algoritmos exitosos en su resolución. Por lo tanto, en primer lugar se ha partido de ideas conocidas para el problema JSS con el fin de diseñar componentes de los algoritmos de búsqueda. Asimismo, partiendo de las propuestas discutidas en la sección anterior, se ha diseñado una nueva estrategia de búsqueda encaminada a calcular soluciones y cotas inferiores de calidad en un tiempo razonable.

Contribuciones y resultados

En primer lugar se ha propuesto un modelo de grafo disyuntivo para el problema. Este es una generalización del utilizado para el JSS, e incluye un nodo ficticio para cada operario y nuevo tipo de arcos que representan asignaciones de operarios a tareas y la secuencia de operaciones que asiste cada operario. Disponer de tal representación de las planificaciones parciales permite abordar el estudio del problema bajo el formalismo y rigor de la Teoría de Grafos, facilitando la definición de las componentes de los algoritmos de resolución, así como las pruebas formales de sus propiedades. Este modelo ha servido para dar una definición recursiva eficiente del cálculo de cabezas y colas de las operaciones en una planificación de orden parcial, las cuales constituyen un elemento indispensable en la resolución del problema.

Componentes de los algoritmos de búsqueda heurística

A partir del modelo disyuntivo, hemos definido los tres elementos siguientes que constituyen la base de un algoritmo de búsqueda exacto: la definición de un espacio de búsqueda mediante un esquema de generación de planificaciones, heurísticos específicos y reglas de poda basadas en relaciones de dominancia entre estados.

***Espacio de búsqueda. El algoritmo** $OG\&T$*

El problema Job Shop Scheduling destaca por ser uno de los más difíciles de la clase NP-hard. Incluso para instancias con 10 trabajos y 10 máquinas, existe tal cantidad de posibles planificaciones que es imposible enumerarlas en la práctica en un tiempo razonable. El problema JSO (n, p) puede sufrir de dicha explosión combinatoria en mayor medida, ya que además de secuenciar las operaciones en las máquinas, se deben tener en cuenta todas las posibles asignaciones de operarios a tareas. Una práctica común para mitigar este fenómeno es el uso de esquemas de generación de planificaciones, que definen un modo efectivo de enumerar un subconjunto de las planificaciones factibles. Estos métodos pueden emplearse en diversos contextos: Por ejemplo, como base de algoritmos voraces [12], generadores de espacios de búsqueda de métodos exactos [67] o como decodificadores en algoritmos genéticos [29]. Un ejemplo notable es el conocido algoritmo $G\&T$ [27] para el problema Job Shop Scheduling. Éste define el conjunto de las planificaciones activas, que cumplen la propiedad de que en ellas no es posible adelantar el tiempo de inicio de una operación sin retrasar el de otra.

Siguiendo una metodología empleada con anterioridad para el problema JSS con tiempos de *setup* dependientes de la secuencia [4], se ha propuesto un nuevo esquema de generación de planificaciones que extiende al algoritmo $G\&T$ considerando las nuevas restricciones del problema JSO(n, p). El nuevo esquema se ha denotado como $OG\&T$ (Operators $G\&T$). En su funcionamiento, las operaciones se planifican de una en una respetando el orden dado en cada trabajo, es decir, maneja un tipo de planificaciones parciales en las que hay algunas tareas planificadas (tienen un tiempo de inicio y un operario asignado). En ellas, se cumple que las predecesoras en el trabajo de cualquier operación planificada también están planificadas. En cada paso, $OG\&T$ determina un conjunto de operaciones candidatas a ser planificadas a continuación, y elige una de ellas asignándole un tiempo de inicio y un operario. Para tal fin, dada una planificación parcial, comienza considerando el conjunto canónico A de todas las tareas cuya predecesora en el trabajo, si existe, ya ha sido planificada. En un segundo paso, construye un conjunto reducido A', que contiene todas las operaciones de A que puedan empezar su procesamiento antes del tiempo de fin más temprano posible de cualquier tarea de A. Finalmente, mediante un análisis del número de operarios disponibles y del número de máquinas requeridas en cada intervalo de tiempo, reduce el número de posibilidades a un nuevo conjunto B, garantizando que en pasos futuros todas las combinaciones descartadas puedan volver a considerarse. En [66], se demuestra formalmente que el algoritmo $OG\&T$ cumple la propiedad de dominancia, es decir, el conjunto de planificaciones que enumera contiene al menos una solución óptima para cualquier instancia.

La propiedad de dominancia hace que el algoritmo $OG\&T$ sea adecuado para definir un espacio de búsqueda que puede ser explorado por métodos exactos. En concreto, cada estado n representa una planificación parcial, y el operador de generación de sucesores, $SUC(n)$, se obtiene a partir del conjunto B calculado por el algoritmo $OG\&T$. Cada operación de B, da lugar a un nuevo estado sucesor $s \in SUC(n)$ en el que la operación se planifica a continuación.

En el mismo trabajo [66], se propone un algoritmo de tipo A* que emplea dicho espacio de búsqueda, además de heurísticos obtenidos de relajaciones del problema y reglas de poda por dominancia para minimizar el makespan. Se ha observado experimentalmente que el algoritmo $OG\&T$ reduce el tamaño del espacio de búsqueda de un modo exponencial. Como resultado, el algoritmo A* es capaz resolver de forma óptima instancias del problema en un tiempo menor en un orden de magnitud que el algoritmo de programación dinámica presentado en [2]. El algoritmo $OG\&T$ también ha sido empleado como decodificador en un algoritmo genético para el problema JSO(n, p) con minimización del makespan [52]. Este algoritmo obtiene resultados sustancialmente mejores que los algoritmos heurísticos propuestos en [2].

Heurísticos

Como se ha comentado en la sección anterior, las estimaciones heurísticas constituyen un elemento clave en la eficiencia de los algoritmos de búsqueda. En esta tesis, hemos diseñado dos heurísticos específicos para el problema JSO(n, p) con minimización del tiempo de flujo total. Ambos se obtienen a partir de relajaciones del problema, por lo que presentan la propiedad de monotonía.

El primero, denotado $h_{PS}(n)$, proviene del problema JSS [67], y se obtiene relajando todas las restricciones de operarios, las de no interrupción y las restricciones de capacidad de todas las máquinas salvo de una de ellas. Cada máquina da lugar a un problema OMS con interrupción y minimización del tiempo de flujo total, que se resuelve de forma óptima en tiempo polinómico mediante la regla de prioridad STP (*Shortest Processing Time*). El heurístico $h_{PS}(n)$, se calcula como el máximo tiempo de flujo total de entre las planificaciones óptimas calculadas para cada problema definido por cada máquina. Este heurístico es un buen estimador en situaciones en donde haya un número elevado de operarios disponibles; es decir, cuando el JSO(n, p) se asemeja al JSS.

El segundo heurístico se denota $h_{OP}(n)$, y se obtiene relajando todas las restricciones salvo las de operarios, además de las cabezas de las operaciones, que se asumen iguales a 0. En [47] se propone un algoritmo de tiempo polinómico que resuelve dicho problema de forma óptima. Al centrarse en las restricciones de operarios, el heurístico h_{OP} es eficaz en los casos en donde hay pocos operarios disponibles.

Los dos heurísticos anteriores se combinan definiendo $h(n) = \text{máx}\, h_{PS}(n), h_{OP}(n)$, obteniendo un efecto sinérgico. Al calcularse como el máximo de dos heurísticos monótonos, el heurístico h es también monótono.

Relaciones de dominancia entre estados

A diferencia de lo que se hace en las estrategias descritas en la sección anterior para el problema JSS, en los algoritmos de búsqueda ideados para el problema JSO(n,p) no hemos utilizado métodos de propagación de restricciones. La razón principal es que aun no se conocen este tipo de métodos para el problema. Además, en [64], se observó que los métodos de propagación de restricciones diseñados para el JSS con minimización del makespan no son muy eficaces al ser aplicados al tipo de planificaciones parciales definidas por el algoritmo $G\&T$. Además, la propagación de restricciones presenta una capacidad deductiva pobre en problemas cuya función objetivo es aditiva, y apenas existen algoritmos eficientes en la literatura, con alguna excepción como el método presentado en [40] para problemas de scheduling con recursos de capacidad unaria y minimización del tiempo de flujo total ponderado.

Como alternativa, se ha utilizado una técnica basada en explotar relaciones de dominancia entre estados que permite realizar podas durante la búsqueda [37]. Dados dos estados n_1 y n_2, se dice que n_1 domina a n_2 si y sólo si $f^*(n_1) \leq f^*(n_2)$. Computacionalmente, el cálculo de $f^*(n)$, definido como el coste de la mejor solución alcanzable desde el nodo n, es tan difícil como resolver el problema JSO(n, p). Sin embargo, en ocasiones se pueden establecer condiciones suficientes que permitan comparar dos estados y determinar la propiedad de dominancia.

Así, se ha definido una relación de dominancia para el problema JSO(n, p) que extiende a la que se propone en [67] para el problema JSS. Dados dos nodos del espacio de búsqueda con las mismas operaciones planificadas, la nueva relación identifica casos de dominancia en función de las cabezas de las operaciones no planificadas, del tiempo de flujo total acumulado en cada estado y de la disponibilidad de operarios en cada caso.

Asimismo, se ha demostrado formalmente una condición necesaria para el cumplimiento de la relación anterior. Esta condición establece que si se emplean heurísticos obtenidos mediante relajaciones del problema y $f(n_1) > f(n_2)$, entonces las condiciones de la relación de dominancia definida no se cumplen. Esto permite una aplicación eficiente de esta técnica para detectar estados dominados.

Estrategias de búsqueda

Las estrategias de búsqueda presentadas en la sección anterior no contemplan la utilización de relaciones de dominancia entre estados. Como se ha comentado, esta técnica puede conducir a importantes reducciones del espacio de búsqueda, por lo que es conveniente realizar un estudio de cómo explotarla adecuadamente en una estrategia de búsqueda. La aplicación reglas de poda por dominancia radica en detectar situaciones en las que un nuevo estado generado sea dominado por algún otro expandido previamente. De este modo, es necesario almacenar nodos expandidos durante la búsqueda e implementar un modo eficiente de comprobar la condición suficiente de dominancia.

A pesar de que el espacio de búsqueda definido por el algoritmo $OG\&T$ puede contener estados duplicados, se han empleado estrategias que lo recorren a modo de árbol. Esto no supone ningún inconveniente, ya que las duplicaciones representan situaciones de dominancia mutua y son uno de los casos detectados por las reglas de poda.

En una primera aproximación hemos diseñado un algoritmo de tipo A* para el problema JSO(n, p) [65]. Esta estrategia de búsqueda, propuesta originalmente en [35] es un paradigma de búsqueda *primero el mejor*. En cada paso, explota el conocimiento heurístico disponible de un modo global expandiendo el estado con menor valor de f entre todos los que han sido generados y aún no expandidos, almacenados en una lista ABIERTA. En el contexto de búsqueda en árboles, no es necesario almacenar los nodos expandidos durante la búsqueda, salvo en los casos en que sea necesario reconstruir la solución desde el estado objetivo hasta el inicial. Sin embargo, con el fin de explotar adecuadamente la regla de poda por dominancia este algoritmo mantiene una lista CERRADA con dichos estados. Tras generar un nuevo estado n, éste se compara con los nodos de CERRADA que tienen las mismas operaciones planificadas. Si alguno de dichos estados domina a n, éste se poda conduciendo a una reducción efectiva del espacio de búsqueda. Con el fin de aplicar esta técnica eficientemente, CERRADA se implementa como una tabla hash donde las claves son vectores de bits que representan el conjunto de las operaciones planificadas en cada estado. De este modo, es muy rápido determinar el conjunto de estados que podrían dominar a uno dado.

El algoritmo A* anterior se combina con un voraz para calcular soluciones desde algunos estados durante la búsqueda, y aunque emplea una versión preliminar de los heurísticos presentados anteriormente, es competente para resolver instancias pequeñas del problema JSO(n, p). Se ha observado experimentalmente que el algoritmo $OG\&T$ define un espacio de búsqueda de un tamaño exponencialmente menor comparado con el espacio completo de las planificaciones factibles. Asimismo, los heurísticos y la relación de dominancia incrementan significativamente la eficacia del algoritmo. Sin embargo, debido a su excesiva necesidad de memoria, este algoritmo sólo es apropiado para resolver instancias de tamaño pequeño. Para instancias grandes, es capaz de calcular cotas inferiores no triviales, pero no llega a buenas soluciones antes de agotar la memoria disponible.

Con el fin de reducir el consumo de memoria y resolver instancias de mayor tamaño, hemos diseñado un algoritmo de búsqueda en profundidad parcialmente informada [47] que integra la relación de dominancia para reducir el espacio de búsqueda. Ésta se aplica según un esquema sencillo pero eficaz. El algoritmo de búsqueda en profundidad mantiene en CERRADA los nodos expandidos hasta el momento, que son utilizados para identificar estados dominados. Los estados expandidos se insertan en CERRADA mientras no se alcance un límite de memoria predefinido, y a partir de dicho momento, CERRADA deja de crecer y sólo se utiliza para podar nuevos nodos generados. Esta limitación de memoria conduce a una explotación parcial de las reglas de poda, pero permite que la búsqueda se pueda ejecutar durante más tiempo. El algoritmo de búsqueda en profundidad parcialmente informada es eficiente en memoria y calcula mejores soluciones que A* en instancias grandes. Sin embargo, no es tan eficaz en el cálculo de cotas inferiores ni certificando soluciones óptimas en instancias pequeñas.

*La nueva estrategia A*DFS*

Una contribución importante ha sido el diseño de una nueva estrategia híbrida de búsqueda heurística que combina las dos estrategias anteriores explotando las ventajas de cada una. La nueva estrategia, propuesta en [50], se denota A*DFS y ha sido ideada con el objetivo principal de la tesis, de calcular buenas soluciones y cotas inferiores en un tiempo y con un consumo de memoria razonables. Consta de dos fases:

En primer lugar, recorre el árbol de búsqueda siguiendo la estrategia A* hasta que llega a un límite de memoria predefinido. Para tal cometido emplea una lista CERRADA y una lista ABIERTA-A*. Cada cierto número de expansiones, lanza una búsqueda en profundidad parcialmente informada limitada por un número de expansiones definido como el doble de la longitud de la rama hasta llegar a la primera hoja. Esta idea se había aplicado previamente en la estrategia *i*IDA*. De este modo, el algoritmo muestrea distintas regiones prometedoras del espacio de búsqueda, con lo que incrementa su capacidad para hallar soluciones de calidad. Cada búsqueda en profundidad opera sobre una lista ABIERTA particular, que se vacía después de cada ejecución.

Una vez que alcanza el límite de memoria, la estrategia A*DFS ejecuta una búsqueda en profundidad exhaustiva desde cada nodo de ABIERTA-A*, y continua hasta que dicha lista se vacía, resolviendo la instancia de forma óptima. En esta fase, las búsquedas en profundidad operan sobre la lista ABIERTA-A*. Cada nodo de dicha lista representa un subárbol del espacio de búsqueda. Al estar ordenados de un modo no decreciente de los valores de f, se desarrollan primero los árboles más prometedores, lo que es beneficioso para la efectividad del método.

Por lo tanto, A*DFS es una estrategia de búsqueda eficiente en memoria que explota la capacidad de A* para sacar partido del conocimiento heurístico y de las reglas de poda por dominancia y la capacidad de DFS para intensificar la búsqueda en regiones concretas.

Se ha implementado un propotipo del algoritmo A*DFS para el problema, y ha sido evaluado sobre un conjunto extenso de instancias del problema generadas a partir de un benchmark obtenido de la OR-library [6]. Al no existir en la literatura propuestas previas para el JSO(n, p) con minimización del tiempo de flujo total, ha sido comparado con un modelo de *constraint programming* implementado en la herramienta comercial *IBM ILOG CPLEX CP Optimizer*. Esta herramienta está especializada en scheduling [42], y obtiene resultados muy competitivos para problemas con restricciones disyuntivas y recursos de capacidad múltiple (los p operarios se modelan como un recurso de capacidad p). Por este motivo, ha sido empleado previamente en la literatura para comparar nuevas propuestas y, en ocasiones, CP Optimizer obtiene resultados difíciles de superar, como en el problema Job Shop Scheduling con restricciones de bloqueo [55]. En el caso del problema JSO(n, p) abordado en esta tesis, los resultados son muy favorables a A*DFS. No sólo supera a A* y a DFS en la calidad de las soluciones obtenidas y en las cotas inferiores, sino que obtiene mejores soluciones que CP Optimizer empleando mucho menos tiempo (5 minutos contra 1 hora).

En el capítulo 5, se encuentra la siguiente publicación, donde se recogen de forma detallada las contribuciones presentadas en este apartado:

- Carlos Mencía, María R. Sierra and Ramiro Varela. An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators. *International Journal of Production Research* 2013. DOI 10.1080/00207543.2013.802389

Capítulo 4

CONCLUSIONES

En este capítulo se resumen las aportaciones de la tesis y se describen brevemente algunas líneas de trabajo futuro.

4.1. Aportaciones de la tesis

Uno de los objetivos principales que nos habíamos propuesto al inicio de esta tesis era el diseño de métodos exactos eficientes para problemas de scheduling, entendiendo que una propiedad deseable de este tipo de algoritmos es su capacidad para certificar soluciones óptimas en instancias de un cierto tamaño y de calcular tanto cotas inferiores no triviales como buenas soluciones en un tiempo razonable para instancias más difíciles.

Siguiendo una metodología de trabajo basada en un análisis riguroso del funcionamiento de los algoritmos de búsqueda actuales y de las características propias de los problemas, ha sido posible diseñar estrategias de búsqueda heurística originales que cumplen la propiedad anterior.

En primer lugar, hemos estudiado cómo explotar de un modo efectivo el conocimiento del domino del problema en el marco de la estrategia de búsqueda primero en profundidad. Se ha observado que la estrategia clásica de *backtracking search*, ampliamente empleada en la literatura, puede ser claramente superada por la búsqueda en profundidad parcialmente informada. Esta estrategia expande los estados completamente y aprovecha la información deducida por los métodos de propagación de restricciones y el conocimiento heurístico en la decisión acerca de qué rama explorar primero. Partiendo del algoritmo de ramificación y poda de P. Brucker [12, 11], hemos diseñado un algoritmo de búsqueda en profundidad parcialmente informada para el problema Job Shop Scheduling con minimización del makespan. Para este problema, hemos propuesto un heurístico original, denotado h_{IS}, que se basa en el método de propagación de restricciones conocido como *selección inmediata* [16]. Aunque h_{IS} es costoso computacionalmente, resulta ser muy efectivo guiando la búsqueda hacia regiones prometedoras, lo que permite al algoritmo de búsqueda en profundidad parcialmente informada calcular mejores soluciones y cotas inferiores empleando menos tiempo que el algoritmo de ramificación y poda de P. Brucker.

Otra contribución importante ha sido el diseño de la nueva estrategia de búsqueda *Intensified Iterative*

*Deepening A** (iIDA*), ideada con el fin de mitigar el problema que sufre la búsqueda en profundidad de estancarse en una región del árbol de búsqueda. Esta estrategia extiende la técnica clásica IDA* [38] con algunas ideas inspiradas en los problemas de optimización de restricciones, y en particular en los problemas de scheduling. En concreto incorpora tres nuevos elementos: La aplicación de métodos de propagación de restricciones para reducir el tamaño del espacio de búsqueda efectivo, el empleo de una estrategia novedosa de actualización del umbral de coste entre cada dos iteraciones consecutivas, reduciendo así el número de reexpansiones, y la combinación con búsquedas en profundidad limitadas con el fin de muestrear distintas regiones prometedoras del árbol de búsqueda. Esta nueva estrategia es capaz de diversificar la búsqueda para encontrar buenas soluciones en un tiempo limitado y, al mismo tiempo, es más efectiva que otras estrategias en el cálculo de cotas inferiores. Una evaluación de la nueva técnica en el problema Job Shop Scheduling ha permitido observar que iIDA* supera a otras estrategias como búsqueda en profundidad, *Limited Discrepancy Search* o IDA*.

Por último, hemos abordado el problema Job Shop Scheduling con Operarios (JSO) propuesto recientemente en la literatura [2], que extiende el problema JSS considerando un número limitado de operarios humanos que deben atender el procesamiento de las operaciones en las máquinas. Hemos considerado la minimización de tiempo de flujo total, la cual es una función objetivo de gran interés en diversos entornos productivos. En un primer paso, hemos diseñado elementos específicos para su resolución, concretamente: un esquema de generación de planificaciones dominante, denominado $OG\&T$, que ha permitido la definición de un espacio de búsqueda completo, dos heurísticos monótonos basados en relajaciones del problema, y una relación de dominancia entre estados que permite reducir notablemente el espacio de búsqueda. La combinación de estos elementos ha dado lugar a una nueva estrategia de búsqueda, denominada A*DFS, eficiente en consumo de memoria y capaz de diversificar la búsqueda. Así, puede obtener buenas soluciones y buenas cotas inferiores al mismo tiempo.

En el desarrollo del trabajo, han aparecido numerosas cuestiones e ideas que aún no han sido estudiadas en profundidad y que constituyen posibles líneas de investigación para el futuro. Algunas de ellas se mencionan brevemente en la sección siguiente.

4.2. Trabajo futuro

Numerosos algoritmos de optimización muy eficaces en el cálculo de soluciones de calidad provienen del campo de las metaheurísticas. Estos algoritmos, como las búsquedas locales o las estrategias evolutivas, constituyen actualmente el estado del arte de numerosos problemas de scheduling. Recientemente, ha habido algunas propuestas en la literatura basadas en combinaciones de algoritmos exactos con metaheurísticas. Por ejemplo, en el marco del problema Job Shop Scheduling, en [69], se combina el algoritmo de ramificación y poda de P. Brucker con un algoritmo de búsqueda local. Otro ejemplo se puede encontrar en [79], donde se combina la estrategia *Solution Guided Search* con el algoritmo de búsqueda tabú i-TSAB [54], siendo actualmente el algoritmo que encuentra mejores soluciones para este problema. Asimismo, los algoritmos de búsqueda exacta pueden ser empleados para definir estructuras de vecindad de algoritmos de búsqueda local. Partiendo de una solución factible, esta se relaja deshaciendo algunas decisiones con el fin de obtener una solución parcial. Dicha solución parcial se puede resolver de forma óptima aplicando algoritmos exactos. Esta idea fue propuesta originalmente bajo el nombre de *shuffling* en [3] para el problema JSS, y es la base de la técnica *Large Neighborhood Search* [63] y de metaheurísticas como *Iterative Flattening Search* [18]. Así, por una parte, emplearemos los algoritmos desarrollados en la tesis para diseñar estructuras de vecindad eficientes siguiendo la idea anterior. Gracias a su capacidad para certificar soluciones óptimas para instancias pequeñas y de tamaño medio, esperamos poder obtener algoritmos de búsqueda local muy eficientes. Por otra parte, estudiaremos modos eficaces de integrar métodos de búsqueda local dentro de las estrategias de búsqueda heurística, con el fin de incrementar su capacidad para alcanzar buenas soluciones en un tiempo corto.

Una segunda posible línea de investigación se deriva del estudio de modos más eficaces de explotar las relaciones de dominancia. Como hemos visto, el principal inconveniente del uso de reglas de poda por dominancia es su elevado consumo de memoria para explotadas de forma exhaustiva. En el campo del razonamiento basado en restricciones, suelen emplearse reglas de detección de simetrías en el espacio de búsqueda, que son en realidad un caso particular de las relaciones de dominancia consideradas en la tesis. En [23], se propone el método denominado *Symmetry Breaking via Dominance Detection* (SBDD), que permite explotar de forma completa las reglas de detección de simetrías en algoritmos de búsqueda en profundidad con un consumo de memoria lineal en la máxima profundidad del árbol de búsqueda. Así, trataremos de generalizar SBDD para ser empleado con reglas de dominancia más generales que las de detección de simetrías. Para ello, necesitaremos definir relaciones de dominancia más generales que permitan comparar estados a distinta profundidad del espacio de búsqueda.

Por último, consideraremos problemas aún más cercanos a situaciones reales y nuevas funciones objetivo. Los problemas de scheduling con recursos humanos, como el JSO, constituyen una buena oportunidad para estudiar métricas como la robustez [19, 58], entendida como la validez de la planificación ante distintos eventos posibles, como por ejemplo una avería en una máquina o la baja de un trabajador. Algunos resultados preliminares sobre esta cuestión se pueden ver en [22]. También consideraremos problemas de scheduling multiobjetivo, y estudiaremos modos de adaptar nuestras técnicas a dicho contexto partiendo de los algoritmos propuestos en [34] y [44]. Por otra parte, en cuanto a los problemas, dado el impacto de la presencia de trabajadores humanos en casi cualquier actividad, abordaremos versiones extendidas del problema JSO, por ejemplo considerando habilidades específicas de cada operario o restricciones derivadas de la normativa laboral.

Capítulo 5

COMPENDIO DE PUBLICACIONES

En este capítulo se incluyen los artículos que dan cuerpo a la tesis doctoral. En primer lugar, se presenta un artículo publicado en el prestigioso congreso ICAPS. Posteriormente se encuentran tres artículos publicados en revistas indexadas en el JCR (2011).

5.1. Partially Informed Depth-First Search for the Job Shop Problem

Se presenta la siguiente publicación:

- Carlos Mencía, María R. Sierra and Ramiro Varela. Partially Informed Depth-First Search for the Job Shop Problem. *In Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS '10)*. Toronto, Canada. AAAI Press. pp: 113-120. 2010
    - Estado: **Publicado.**

Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

Partially Informed Depth-First Search for the Job Shop Problem

Carlos Mencía and **María R. Sierra** and **Ramiro Varela**
Department of Computer Science,
University of Oviedo, 33271 Gijón (Spain)
e-mail: {uo156612, sierramaria, ramiro}@uniovi.es

Abstract

We propose a partially informed depth-first search algorithm to cope with the Job Shop Scheduling Problem with makespan minimization. The algorithm is built from the well-known P. Brucker's branch and bound algorithm. We improved the heuristic estimation of Brucker's algorithm by means of constraint propagation rules and so devised a more informed heuristic which is proved to be monotonic. We conducted an experimental study across medium and large instances. The results show that the proposed algorithm reaches optimal solutions for medium instances taking less time than branch and bound and that for large instances it reaches much better lower and upper bounds when both algorithms are given the same amount of time.

Introduction

The Job Shop Scheduling Problem (JSSP) is a paradigm of optimization and constraint satisfaction problems which has interested researchers over the last decades, due to its difficulty and its proximity to real-life problems. It is NP-hard in the strong sense (Garey and Johnson 1979) and is considered as one of the most intractable problems known so far.

In this paper, we focus on the JSSP with makespan minimization and propose a partially informed depth-first search algorithm to solve it. This algorithm is built from the well-known Peter Brucker's branch and bound algorithm (Brucker, Jurisch, and Sievers 1994), which is, as far as we know, the most effective exact method to cope with this problem. We devised a new heuristic obtained from a lower bound calculation method proposed in (Carlier and Pinson 1990), which is based on constraint propagation rules, and we prove it is monotonic.

We have conducted an experimental study across conventional medium and large instances to compare our approach with Brucker's algorithm. The results show that for medium-size instances our approach reduces the time required to solve them by about 15%. For the largest instances, that in most of the cases cannot be optimally solved by any of them, our approach is able to reach much better lower and upper bounds when both algorithms are given the same time. In this case, our approach reduces the error with respect to the best known solutions by about 40%.

The remainder of the paper is structured as follows: In section 2 a formulation of the JSSP is given. Section 3 outlines the main characteristics of Brucker's algorithm. In section 4 we describe the proposed partially informed depth-first search algorithm. In section 5 we present a heuristic based on a constraint propagation method and prove its monotonicity. Section 6 reports the results of the experimental study. Finally, in section 7 we summarize the main conclusions and propose some ideas for future research.

The Job Shop Scheduling Problem

The Job Shop Scheduling Problem (JSSP) requires scheduling a set of N jobs $\{J_1, \dots, J_N\}$ on a set of M resources or machines $\{R_1, \dots, R_M\}$. Each job J_i consists of a set of tasks or operations $\{\theta_{i1}, \dots, \theta_{iM}\}$ to be sequentially scheduled. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, a fixed duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. The JSSP has three constraints: precedence, capacity and non-preemption. Precedence constraints translate into linear inequalities of the type: $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$. Capacity constraints translate into disjunctive constraints of the form: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, if $R_v = R_w$. Non-preemption requires assigning machines to operations without interruption during their whole processing time. The objective is to come up with a feasible schedule such that the completion time, i.e. the *makespan*, is minimized. This problem is denoted as $J||C_{max}$ in the $\alpha|\beta|\gamma$ notation.

Below, a problem instance will be represented by a directed graph $G = (V, A \cup E)$. Each node in the set V represents an actual operation, with the exception of the dummy nodes *start* and *end*, which represent operations with processing time 0. The arcs of A are called *conjunctive arcs* and represent precedence constraints and the arcs of E are called *disjunctive arcs* and represent capacity constraints. E is partitioned into subsets E_i with $E = \cup_{\{i=1,\dots,M\}} E_i$. E_i includes an $arc\ (v, w)$ for each pair of operations requiring R_i. The arcs are weighted with the processing time of the operation at the source node. Node *start* is connected to the first operation of each job and the last operation of each job is connected to node *end*.

A feasible schedule S is represented by an acyclic subgraph G_S of G, $G_S = (V, A \cup H)$, where $H = \cup_{i=1,\dots,M} H_i$, H_i being a processing ordering for the operations requiring R_i. The makespan is the cost of a *critical*

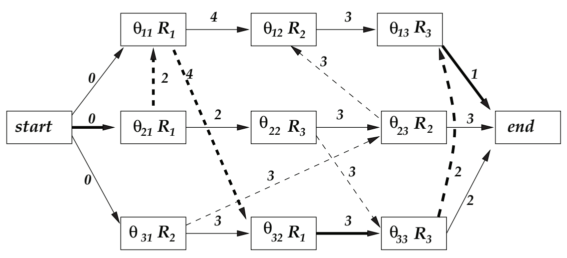


Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines. Bold face arcs show a critical path whose length (makespan) is 12

path and it is denoted as $L(S)$. A critical path is a longest path from node *start* to node *end*. A critical path contains a set of *critical blocks*. A critical block is a maximal sequence, with length at least two, of consecutive operations in a critical path requiring the same machine. Figure 1 shows a solution graph for an instance with 3 jobs and 3 machines.

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v, i.e. it is the value of st_v. The *tail* q_v is defined so as the value $q_v + p_v$ is the cost of the longest path from v to *end*. Hence, $r_v + p_v + q_v$ is the makespan if v is in a critical path, otherwise, it is a lower bound. PM_v and SM_v denote the predecessor and successor of v respectively on the machine sequence and PJ_v and SJ_v denote the predecessor and successor operations of v respectively on the job sequence.

A partial schedule is given by a subgraph of G where some of the disjunctive arcs are not fixed yet. In such a schedule, heads and tails can be estimated as

$$r_v = \max\{\max_{J \subseteq P(v)}\{\min_{j \in J} r_j + \sum_{j \in J} p_j\}, r_{PJ_v} + p_{PJ_v}\} \quad (1)$$

$$q_v = \max\{\max_{J \subseteq S(v)}\{\sum_{j \in J} p_j + \min_{j \in J} q_j\}, p_{SJ_v} + q_{SJ_v}\} \quad (2)$$

with $r_{start} = q_{end} = 0$ and where $P(v)$ denotes the disjunctive predecessors of v, i.e. operations requiring machine R_v which are scheduled before than v. Analogously, $S(v)$ denotes the disjunctive successors of v.

Brucker's Algorithm

As far as we know, the best exact method proposed so far to cope with the $J||C_{max}$ problem is the branch and bound algorithm due to P. Brucker et al. (Brucker, Jurisch, and Sievers 1994). This algorithm starts from the constraint graph for a given problem instance and proceeds to fix disjunctive arcs subsequently. The key feature of the algorithm is a smart branching scheme that relies on fixing arcs either in the same or in the opposite direction as they appear in a critical block of a feasible solution. Moreover, the algorithm exploits powerful methods to obtain accurate lower and upper bounds. Lower bound calculation is based on preemptive one machine sequencing problem relaxations. The optimal solution to the simplified problem is given by the so-called Jackson's Preemptive Schedule (JPS), which is obtained in polynomial time by the algorithm proposed in (Carlier 1982). Upper bounds are obtained by means of the greedy $G\&T$ algorithm proposed in (Giffler and Thomson 1960). Finally, Brucker's algorithm exploits a constraint propagation method termed *immediate selection* (Carlier and Pinson 1989). This method allows to fix additional disjuntive arcs so as the effective search tree is dramatically reduced. Brucker's algorithm can easily solve almost any instance up to 10 jobs and 10 machines as well as many larger instances. In fact, the famous set of instances selected in (Applegate and Cook 1991) are considered very hard to solve due to the fact that the great majority of them are not solved by this algorithm.

A Partially Informed Depth-First Search Algorithm

In this section, we present a partially informed depth-first search algorithm which looks for optimal schedules in the same search space as Brucker's algorithm. We describe the state representation, the way of computing heuristic estimations and upper bounds, the expansion mechanism, a constraint propagation method that reduces the search space and the search strategy used by the algorithm.

State Representation

A search state n is given by a partial solution graph $G_n = (V, A \cup FD_n)$, where FD_n denotes the disjunctive arcs fixed in n. In the initial state $FD_n = \oslash$. Heads and tails in n are calculated by expressions (1) and (2). The cost of the best path from the initial state to n, denoted by $g(n)$, is given by the largest cost path between nodes $start$ and end in G_n. As $g(n)$ is computed from n, it is always optimal no matter what path led to it. Hence, $g(n) = g^*(n)$.

Heuristic Estimation

For each generated state n, a heuristic estimation $f(n)$ of the cost of the best solution reachable from n (i.e. the cost of the best path from the initial state to a goal constrained to pass through n) is computed. This estimation is based on problem splitting and constraint relaxations. Let $\mathbf{O}$ be the set of operations requiring the machine m. Scheduling operations in $\mathbf{O}$ accordingly to their heads and tails in n so as the value $max_{v \in \mathbf{O}}(st_v + p_v + q_v)$ is minimized is known as the One Machine Sequencing Problem (OMSP). The optimal solution to this problem for n is clearly a lower bound of $f^*(n) = g^*(n) + h^*(n)$, where $h^*(n)$ is the optimal cost from n to its nearest goal. However, the OMS problem is still NP-hard. So, a new relaxation is required in order to obtain a polynomially solvable problem. To do that, it is common to relax the non-preemption constraint. An optimal solution to the preemptive OMS problem is given by the Jackson's Preemptive Schedule (JPS) (Carlier 1982). The JPS is calculated by the following algorithm: at any time t given by a head or the completion of an operation, from the minimum r_v until all operations are completely scheduled, schedule the ready operation with the largest tail on machine m. Calculating the JPS has a complexity of $O(k \times \log k)$, where

Algorithm 1 $G\&T$(state n). Calculates a heuristic solution S from a state n. O denotes the set of all operations and SC the set of scheduled operations

```
0. $SC = \{start\}$;
while ($SC \neq O$) do
    1. $A = \{v \in O \setminus SC; P(v) \cup \{PJ_v\} \subset SC\}$;
    2. $v^* = \arg\min\{r_u + p_u; u \in A\}$;
    3. $B = \{v \in A; R_v = R_{v^*}$ and $r_v < r_{v^*} + p_{v^*}\}$;
    4. $C = \{v \in O \setminus SC; R_v = R_{v^*}\}$;
    5. $w^* = \arg\min\{$makespan of JPS($C \setminus \{w\}$) after schedule $w$; $w \in B\}$;
    6. Schedule $w^*$ in $S$ at a time $r_{w^*}$;
    7. Add $w^*$ to $SC$ and update heads of operations not in $SC$;
end while
8. return $S$ and its makespan;
```

k is the number of operations. Finally, $f(n)$ is taken as the largest JPS over all machines. So, $h(n) = f(n) - g(n)$ is an optimistic estimate of $h^*(n)$. Below, $f_{JPS}(n)$ and $h_{JPS}(n)$ denote these heuristic estimations for n.

Upper Bounds

As Brucker's algorithm does, we introduce upper bound calculations in the depth-first search counterpart. To do that, a variant of the well-known $G\&T$ algorithm proposed in (Giffler and Thomson 1960) is used. The algorithm is issued from each expanded node so as it builds a schedule that includes all disjunctive arcs fixed in that state. Note that the disjunctive arcs fixed to obtain the schedule do not remain fixed in that node. $G\&T$ is a greedy algorithm that produces a schedule in a number of $N * M$ steps. Algorithm 1 shows the $G\&T$ algorithm adapted to obtain upper bounds from a search state n. Remember that $P(v)$ denotes the disjunctive predecessors of operation v in state n. In each iteration, the algorithm considers the set A comprising all operations v that can be scheduled next, i.e. all operations such that $P(v)$ and PJ_v are already scheduled (initially only operation $start$ is scheduled). The operation v^* in A with the earliest completion time if it is scheduled next is determined, and a new set B is obtained with all operations v in A requiring the same machine as v^* that can start at a time lower than the completion time of v^*. Any of the operations in B can be scheduled next and the selected one is w^* if it produces the least cost JPS for the remaining unscheduled operations on the same machine. Finally, the algorithm returns the built schedule S and the value of its makespan.

The best upper bound computed so far, with makespan UB, is maintained so as generated states n having $f(n) \geq UB$ are pruned from the search space as they do not lead to better solutions.

Expansion Mechanism

The expansion mechanism is based on the following theorem (Brucker, Jurisch, and Sievers 1994).

Theorem 1. *Let S and S' be two schedules. If $L(S') < L(S)$, then one of the two following conditions holds:*

1) at least one operation v in a critical block B in G_S, different from the first operation of B, is processed in S' before all operations of B.

2) at least one operation v in a critical block B in G_S, different from the last operation of B, is processed in S' after all operations of B.

Now, let us consider a feasible schedule S being compatible with the disjunctive arcs fixed in state n. Of course, S might be the schedule calculated by Algorithm 1. The solution graph G_S has a critical path with critical blocks $B_1, \ldots, B_k$. For block $B_j = (u_1^j, \ldots, u_{m_j}^j)$ the sets of operations

$$E_j^B = B_j \setminus \{u_1^j\} \text{ and } E_j^A = B_j \setminus \{u_{m_j}^j\}$$

are called the $before$-$candidates$ and $after$-$candidates$ respectively. For each before-candidate (after-candidate) a successor s of n is generated by moving the candidate before (after) the corresponding block. An operation $l \in E_j^B$ is moved before B_j by fixing the arcs $\{l \to i; i \in B_j \setminus \{l\}\}$. Similarly, $l \in E_j^A$ is moved after B_j by fixing the arcs $\{i \to l; i \in B_j \setminus \{l\}\}$.

This expansion strategy is complete as it guaranties that at least one optimal solution is contained in the search graph. However, this strategy can be improved by fixing additional arcs so as the search space is a complete tree. Let us consider a permutation $(E_1, \ldots, E_{2k})$ of all sets E_j^B and E_j^A. This permutation defines an ordering for successors generation. When a successor is created from a candidate E_t, we can assume that all solutions reachable from n by fixing the arcs corresponding to the candidates $E_1, \ldots, E_{t-1}$ will be explored from the successors associated to these candidates. So, for the successor state s generated from E_t the following sets of disjunctive arcs can be fixed: $F_j = \{u_1^j \to i; i = u_2^j, \ldots, u_{m_j}^j\}$, for each $E_j^B < E_t$ and $L_j = \{i \to u_{m_j}^j; i = u_1^j, \ldots, u_{m_j-1}^j\}$, for each $E_j^A < E_t$ in the permutation above. So the successors of a search tree node n generated from the permutation $(E_1, \ldots, E_{2k})$ are defined as follows. For each operation $l \in E_j^B$ generate a search tree node s by fixing the arcs $FD_s = FD_n \cup S_j^B$, provided that the resulting partial solution graph has no cycles, with

$$S_j^B = \bigcup_{E_i^B < E_j^B} F_i \cup \bigcup_{E_i^A < E_j^B} L_i \cup \{l \to i : i \in B_j \backslash \{l\}\}. \tag{3}$$

And for each operation $l \in E_j^A$ generate a search tree node s by fixing the arcs $FD_s = FD_n \cup S_j^A$ with

$$S_j^A = \bigcup_{E_i^B < E_j^A} F_i \cup \bigcup_{E_i^A < E_j^A} L_i \cup \{i \to l : i \in B_j \backslash \{l\}\}. \tag{4}$$

Fixing Additional Arcs by Constraint Propagation

After adjusting heads and tails, new disjunctive arcs can be fixed by the constraint propagation method due to Carlier and Pinson (Carlier and Pinson 1989), termed immediate selection. Below, I denotes the set of operations requiring a

Algorithm 2 PROCEDURE Select

```
for all $c, j \in I, c \neq j$ do
    if $r_c + p_c + p_j + q_j \geq UB$ then
        fix the arc $(j \rightarrow c)$;
    end if
end for
```

given machine. For each operation $j \in I$, r_j and q_j denote the head and tail respectively of the operation j in the state n.

Theorem 2. *Let $c, j \in I, c \neq j$. If*

$$r_c + p_c + p_j + q_j \geq UB, \tag{5}$$

j has to be processed before c in every solution reachable from state n that improves UB.

The arc $(j \rightarrow c)$ is called $direct\ arc$ and the procedure Select given in Algorithm 2 calculates all direct arcs for the state n in a time of order $O(|I|^2)$.

The procedure Select can be combined with the method due to Carlier and Pinson that allows to improve heads and tails. This method is based on the following result.

Theorem 3. *Let $c \in I$ and $J \subseteq I \backslash \{c\}$.*

1) If

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB, \tag{6}$$

then in all solutions reachable from state n improving UB, the operation c has to be processed after all operations in J.

2) If

$$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J \cup \{c\}} q_j \geq UB, \tag{7}$$

then in all solutions reachable from state n improving UB, the operation c has to be processed before all operations in J.

If condition 1) of the theorem above holds, then the arcs $\{j \rightarrow c; j \in J\}$ can be fixed. These arcs are called $primal\ arcs$ and the pair (J, c) is called $primal\ pair$. Similarly, if condition 2) holds, the $dual\ arcs$ $\{c \rightarrow j; j \in J\}$ can be fixed and (c, J) is a $dual\ pair$.

In (Brucker, Jurisch, and Sievers 1994), an efficient method is derived to calculate all primal and dual arcs. This method is based on the following ideas. If (J, c) is primal pair, the operation c cannot start at a time lower than

$$r_J = \max_{J' \subseteq J} \{\min_{j \in J'} r_j + \sum_{j \in J'} p_j\}. \tag{8}$$

So, if $r_c < r_J$, we can set $r_c = r_J$ and then the procedure Select fixes all primal arcs $\{j \rightarrow c; j \in J\}$.

This fact leads to the following problem.

Definition 1 (Primal problem). *Let $c \in I$. Does there exist a primal pair (J, c) such that $r_c < r_J$? If it exists, find*

$$r_{J^*} = \max\{r_J; (J, c)\, is\, a\, primal\, pair\}. \tag{9}$$

In (Carlier and Pinson 1994), Carlier and Pinson propose an algorithm which is also based on JPS calculations to solve the primal problem in a time $O(k \log k)$, with $k = |I|$. So, all primal pairs can be computed in $O(k^2 \log k)$. We refer the interested reader to Carlier and Pinson's or Brucker et al.'s papers.

Analogously, if (c, J) is dual pair, q_c cannot be lower than

$$q_J = \max_{J' \subseteq J} \{\sum_{j \in J'} p_j + \min_{j \in J'} q_j\}. \tag{10}$$

So, if $q_c < q_J$, we can set $q_c = q_J$ and then the procedure Select fixes all dual arcs $\{c \rightarrow j; j \in J\}$. All dual pairs can be obtained similarly.

Finally, the algorithm used to fix additional disjunctive arcs proceeds as follows:

1) calculation of all primal arcs for all machines,
2) calculation of new heads and tails,
3) calculation of all dual arcs for all machines,
4) calculation of new heads and tails.

As new heads and tails are computed in steps 2 and 4 due to the additional arcs fixed in steps 1 and 3, steps 1-4 should be repeated as long as new disjunctive arcs are fixed.

Search Strategy

Brucker's algorithm uses a backtracking procedure and a simple dispatching rule for selecting the next successor node. When exploring a node, it computes all before and after candidates in a critical path, it then generates only one successor at a time which is explored next, giving priority to before (after) candidates with the smallest heads (tails resp.). By doing so, it only takes profit from the heuristic estimations (lower bounds) for pruning those states n having $f(n) \geq UB$. According to (Pearl 1984), the main advantage of this search strategy is the low use of memory.

We propose here to use a depth-first search strategy in which successors of a given state are sorted by non-decreasing values of $f(\cdot)$, so as the most promising of them are expanded first. So, all successors of a state n are generated and stored before exploring any of them, i.e. n is expanded. In this way, the selection of the next node to explore is based on more knowledge from the problem domain than that of a single dispatching rule. In our algorithm this rule is used for breaking ties for nodes with the same value of $f(\cdot)$ as it has given better results than other methods such as considering $g(\cdot)$ values or the number of arcs fixed in the node.

Guiding the search by knowledge from the problem domain when traversing the search tree has important benefits. The sooner good upper bounds are found, the more effective the immediate selection procedure is, what leads to reduce the search space, compute more accurate heuristic estimations, reach good upper bounds sooner and so on.

This search strategy needs more memory resources than backtracking, as all successors of the states along the current branch are stored. Nevertheless, these requirements are still low, due to the depth-first search.

Heuristic Improvement by Constraint Propagation

To improve the heuristic estimations, we use the following strategy which is inspired in some ideas taken from (Carlier and Pinson 1990) for computing lower bounds for the JSSP.

If we have a heuristic estimation $f(n)$ for a state n and we can prove in any way that a solution with cost not greater than $P \geq f(n)$ may not be reachable from state n, then we can improve the heuristic estimation up to $P+1$. In order to do that, we apply immediate selection to n considering the upper bound $P+1$. This way, we are supposing that there exists a solution with makespan not greater than P (lower than $P+1$) reachable from n. So, additional disjunctive arcs get fixed and the resulting situation is a state, denoted n_r, that in general is not a node of the search tree. However, any solution with cost not greater than P reachable from n must include all arcs fixed in state n_r. Hence, if the partial solution graph G_{n_r} is inconsistent, such a solution cannot exist and then the estimation can be improved to $P+1$. The inconsistency of G_{n_r} can be established if one of the two following conditions holds:

1) G_{n_r} contains a cycle.
2) A lower bound greater than P can be derived from G_{n_r}.

In the first case the inconsistency is clear as all arcs fixed in G_{n_r} should be included in the final solution graph and a solution graph must be acyclic. In the second case, even if G_{n_r} has no cycles, there is an inconsistency as a solution with cost lower than or equal to P containing the arcs fixed in G_{n_r} cannot exist. In order to check this condition we use the lower bound given by $f_{JPS}(n_r)$.

To compute the improved heuristic a dichotomic search in the interval $[f_{JPS}(n), UB-1]$ is made. The new heuristic estimation, termed f_{IS}, is computed as $f_{IS}(n) = P$, where P is the smallest value in the interval that produces a graph G_{n_r} without inconsistencies (after applying the immediate selection procedure with the upper bound $P+1$). Note that the value $P = f_{JPS}(n) - 1$ would produce an inconsistent graph G_{n_r} as at least a lower bound equal to $f_{JPS}(n)$ would be derived from it. Analogously, $P = UB - 1$ would generate a graph $G_{n_r} = G_n$, without any inconsistency, as the immediate selection method is applied to n with upper bound UB after n is generated and before computing this heuristic estimation for it. So, the new heuristic, denoted h_{IS}, is obtained as $h_{IS}(n) = f_{IS}(n) - g(n)$.

Monotonicity of h_{IS}

It is clear that $h_{IS}(n) \geq h_{JPS}(n)$, for every state n. We now prove that h_{IS} is monotonic. In this section, $r_i(n)$ denotes the head of operation i in the search state n. Analogously, $q_i(n)$ refers to its tail. $SCS(n)$ is the set containing all successors of n.

Lemma 1. *Let n and n' be two states such that $n' \in SCS(n)$. Then, $\forall i, r_i(n) \leq r_i(n')$ and $q_i(n) \leq q_i(n')$.*

Proof. Generating n' from n requires fixing at least one disjunctive arc in n' without producing any cycle in $G_{n'}$. So, every path from node $start$ to node i in G_n belongs to $G_{n'}$ as well. As the head of operation i is computed across the paths from $start$ to i it is clear that $\forall i, r_i(n) \leq r_i(n')$. Analogously for tails, so $q_i(n) \leq q_i(n')$. □

Lemma 2. *Let n and n' be two states such that $n' \in SCS(n)$. If an arc $i \to j$ gets fixed by immediate selection with an upper bound $P+1$ in n, then it will get fixed in n' with $P+1$ as well.*

Proof. The immediate selection procedure fixes an arc $i \to j$ in n if $r_j(n) + p_i + p_j + q_i(n) \geq P+1$. From Lemma 1, $\forall k$, $r_k(n') \geq r_k(n)$ and $q_k(n') \geq q_k(n)$, so $r_j(n') + p_i + p_j + q_i(n') \geq P+1$. Hence, $i \to j$ will be also fixed in n' by immediate selection. □

Corollary 1. *Let n and n' be two states such that $n' \in SCS(n)$. And let n_r and n'_r be the resulting states from applying immediate selection to states n and n' respectively considering the same upper bound. Then G_{n_r} is a subgraph of $G_{n'_r}$ and $\forall i, r_i(n_r) \leq r_i(n'_r)$ and $q_i(n_r) \leq q_i(n'_r)$*

Proof. It is trivial from Lemma 2 that G_{n_r} is a subgraph of $G_{n'_r}$, as $FD_n \subset FD_{n'}$ and every arc fixed in n by immediate selection is fixed in n' as well. Also, from similar reasoning as in Lemma 1, heads and tails in $G_{n'_r}$ are larger or at least equal than they are in G_{n_r} as the set of paths from $start$ to i and from i to end in G_{n_r} are subsets of the corresponding sets of paths in $G_{n'_r}$. □

Lemma 3. *Let n, n' be two search states such that $n' \in SCS(n)$. Then $f_{IS}(n) \leq f_{IS}(n')$.*

Proof. Let G_{n_r} and $G_{n'_r}$ be the resulting graphs of applying immediate selection with upper bound $P+1$ to n and n' respectively. To prove that $f_{IS}(n) \leq f_{IS}(n')$ it is enough to see that for any $P+1$ that G_{n_r} is inconsistent then $G_{n'_r}$ is inconsistent as well. We analyze separately the two conditions for inconsistency given previously.

1) As G_{n_r} is a subgraph of $G_{n'_r}$, if G_{n_r} contains a cycle, then $G_{n'_r}$ contains a cycle as well.
2) As $\forall i, r_i(n_r) \leq r_i(n'_r)$ and $q_i(n_r) \leq q_i(n'_r)$, any preemptive schedule for a machine in n'_r is a feasible preemptive schedule for the same machine in n_r (but not conversely), so the Jackson's Preemptive Schedule for every machine in n'_r is greater or equal than it is in n_r and so $f_{JPS}(n'_r) \geq f_{JPS}(n_r)$. Hence, if $f_{JPS}(n_r) > P$, then $f_{JPS}(n'_r) > P$.

So it follows that $f_{IS}(n) \leq f_{IS}(n')$. □

Theorem 4. *h_{IS} is monotonic.*

Proof. From Lemma 3 we know $f_{IS}(n) \leq f_{IS}(n') \forall n, n' \in SCS(n)$. This is equivalent to $g(n) + h_{IS}(n) \leq g(n') + h_{IS}(n')$. As $\forall s, g(s)$ is computed as the length of the largest path from $start$ to end in G_s, the cost of the path from the initial state to s is always known and equal to $g^*(s)$, no matter what path led to s. This implies that $g(n') - g(n) = c(n, n')$, so $h_{IS}(n) \leq c(n, n') + h_{IS}(n')$. Hence, h_{IS} is monotonic (equivalently consistent) and consequently admissible. □

Remark 1. *The results given by Lemma 2 and Corollary 1 assume that the states* n *and* $n' \in SCS(n)$ *are initially consistent with* $P+1$*, i.e., their associated graphs are acyclic and* $f_{JPS}(n) \leq f_{JPS}(n') < P+1$*. In other cases, it is trivial that Lemma 3 holds.*

Analysis of the Effectiveness of h_{IS}

As we have seen, the improved heuristic h_{IS} is more informed than the original one h_{JPS}, i.e. $h_{IS}(n) \geq h_{JPS}(n)$ for every state n, so it is expected that the number of nodes expanded by the partially informed depth-first search algorithm is smaller with h_{IS} than it is with h_{JPS}. The reason for this is that the values of the function $f(\cdot)$ are larger, so more nodes are pruned from the condition $f(n) \geq UB$ and, at the same time, the evaluation function guides the search towards more promising regions of the search space so as better upper bounds are reached quickly. However, it is also clear that computing h_{IS} takes more time than computing h_{JPS}, so we have to consider whether or not the increase in time consumed by the heuristic is compensated by the reduction of the effective search space. To do that we considered some of the instances with 10 jobs and 10 machines that in our experiments have required more search time (namely ORB01, ORB03, FT10 and LA20). For each of them we have analyzed the difference between the two heuristic estimations and the number of nodes visited at different levels of the search space. As the number of arcs fixed from a state to a successor is not constant, we have taken the number of disjunctive arcs fixed in a node as its level, instead of the length or the cost of the path from the initial state to it. The initial state has no disjunctive arcs fixed whereas the maximum number of arcs that can be fixed for an instance with N jobs and M machines is given by the expression

$$maxArcs(N, M) = M \times \frac{(N-1)^2 + (N-1)}{2}. \qquad (11)$$

States having such a number of disjunctive arcs fixed represent feasible schedules. However, the partially informed depth-first search algorithm rarely reaches this situation, due to the condition $f(n) \geq UB$ that allows to prune the node n.

Figure 2 shows the results from instance ORB01 (the results from ORB03, FT10 and LA20 are fairly similar). The x-axis represents the percentage of the disjunctive arcs fixed (with respect to $maxArcs(10, 10)$). And the y-axis represents the average improvement in percent of h_{IS} over h_{JPS}, computed for each node n as

$$100 \times \frac{h_{IS}(n) - h_{JPS}(n)}{h_{JPS}(n)}. \qquad (12)$$

As we can observe, the average improvement is about 30% and it is more or less uniform for different values of the number of arcs fixed in the states, with variations in only a very small fraction of the nodes at low and high levels of the search.

Figure 3 illustrates the distribution of the states evaluated with respect to the number of disjunctive arcs fixed. As we can observe they are normally distributed: the number of nodes evaluated at low levels is very small, then the number increases quickly for intermediate levels and finally it is very low again for levels close to the final states. This is quite reasonable, as at the end most of the nodes get pruned and at the beginning the number of states is lower than it is at intermediate levels. The results given in figures 2 and 3 correspond to the search space traversed by the algorithm using the heuristic h_{IS}. With h_{JPS} there are only small variations due to the differences in the number of evaluated nodes.

These results suggest us the possibility of using different heuristics at different levels. In particular we propose using h_{IS} at low levels where there are few nodes and the decisions are more critical. At intermediate levels maybe the use of a low cost heuristic such as h_{JPS} could be better as there are a very large number of states and the decisions are less critical. And finally, at the last levels where the decisions are much less critical and few nodes are visited the heuristic is not very relevant. So we propose using h_{IS} up to a given level of the search in order to take the most critical decisions and then use h_{JPS} in order to save time.

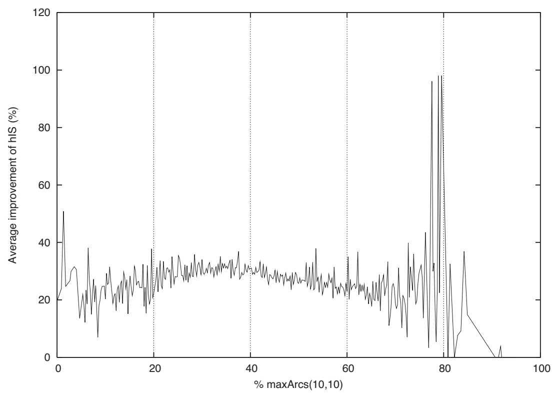


Figure 2: Distribution of heuristic improvements in the effective search space depending on the number of arcs fixed

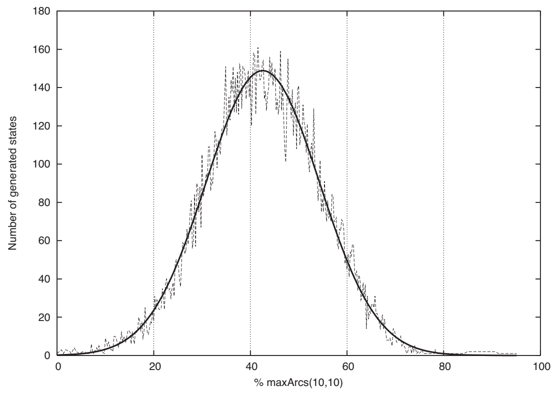


Figure 3: Distribution of states in the effective search space depending on the number of arcs fixed

Table 1: Results for instances of size 10×10

	BB	DF 0%	20%	35%	50%	65%	80%	100%
$Exp.$	100,00	69,45	67,71	67,42	68,21	68,63	68,46	68,41
$T.(s)$	100,00	86,41	86,41	97,09	119,42	138,83	147,57	148,54

Table 2: Results for selected instances

	BB	DF 0%	20%	35%	50%	65%	80%	100%
E_{UB}	100,00	81,17	66,50	60,76	51,94	49,40	52,14	52,39
E_{LB}	100,00	100,00	45,97	65,74	60,63	65,74	65,74	65,74

Table 3: Results for instances of size 20×15

	BB	DF 0%	20%	35%	50%	65%	80%	100%
E_{UB}	100,00	74,19	69,01	69,96	69,84	62,62	58,65	58,20
E_{LB}	100,00	100,00	70,84	70,84	70,84	70,84	70,84	70,84

Computational Results

As we have pointed, we have conducted an experimental study to compare the proposed partially informed depth-first search algorithm (DF) with the original Brucker's branch and bound algorithm (BB). We have considered three sets of benchmarks taken from the OR-library. Firstly, eighteen instances of size 10×10 (10 jobs and 10 machines): FT10, ABZ5-6, LA16-20 and ORB01-10. As all of these instances are easily solved by both algorithms, we report the average number of nodes expanded and time taken to reach an optimal schedule (and prove its optimality) in percentage with respect to those obtained by BB. Then, we considered the set of instances selected in (Applegate and Cook 1991) as very hard to solve: FT20 (size 20×5), LA21, LA24, LA25 (15×10), LA27, LA29 (20×10), LA38, LA40 (15×15), ABZ7, ABZ8 and ABZ9 (20×15). Finally, we considered the set of Taillard's instances of size (20×15) together with ABZ7, ABZ8 and ABZ9. As most of these instances are not optimally solved by any of the algorithms, we consider the quality of the upper and lower bounds reached by BB in average with respect to the best known lower and upper bounds (taken from (Zhang et al. 2008)). Then we report the average error in the upper and lower bounds (E_{UB} and E_{LB} respectively) reached by DF with respect to those reached by BB. For the three sets of instances we have considered a number of levels to apply the improved heuristic h_{IS}: 0, 20, 35, 50, 65, 80 and 100%. With level 0% the improved heuristic is not applied to any of the nodes. In the other cases, when the improved heuristic is not applied, we use the estimation $f(n) = \max(f(p), f_{JPS}(n))$, where p is the parent of state n. In all cases we report values averaged for all the instances of the set and normalized so as BB is given the value 100. The target machine was Linux (Ubuntu 9.04) on Intel Core 2 Quad Q9400 (2,66 GHz), 4 GB. RAM.

Table 1 reports the results from the 10×10 instances. The average time taken by the BB algorithm is $5,72$ seconds and the average number of expanded nodes is $7184,67$. The first remarkable result is that DF reduces the number of expanded nodes by about 30% with respect to BB. This number is almost the same disregarding the level of application of the improved heuristic (even if only h_{JPS} is used). This is a little bit surprising as the differences between the two heuristic estimations are about 30% in average, as we have seen in the previous section. In our opinion this is due to the fact that these instances are easy to solve and the single heuristic h_{JPS} is able to guide the search quite well. At the same time, the intensive use of h_{IS} only contributes to increase the time taken so as the overall time is even greater than the time taken by BB when h_{IS} is applied at a level beyond 35%. For these instances, applying h_{IS} at a level in $[0, 20]$ is the best choice and, in this case, the time is reduced by about 15% with respect to BB.

Table 2 shows the results from the second set of instances. In this case all algorithms were run for 1 hour. The average error reached by BB is $5,37\%$ for upper bounds and $3,42\%$ for lower bounds with respect to the best known bounds. As we can observe, DF is better than BB in all cases. When h_{IS} is used, the improvement in lower bounds is about 35% in all cases, independently of the level up to h_{IS} is applied, with exception of the level 20%. This is due to the fact that the instance $LA38$ is solved optimally in this case and so the lower bound is much better than in the remaining ones. The fact that the lower bound is almost the same for all levels is quite reasonable due to the depth-first search. The lower bound is the lowest $f(\cdot)$ of a node in the open list and, as the instances are very large, this value might correspond to a successor of the initial state due to the fact that the algorithm is not able to expand all of these successors after one hour. For the same reason the lower bound at level 0% is the same as that of BB. However, the quality of the upper bounds improves in direct ratio with the level up to h_{IS} is applied. In this case it is worth to exploit the improved heuristic beyond level 50%. In this case the improvement with respect to BB is about 50%. Moreover, BB does not reach an optimal solution to the instance FT20, whereas DF solves it in just a few seconds using h_{IS}.

Table 3 summarizes the results from the largest instances (size 20×15) obtained in 1 hour. These are extremely hard instances. The average error reached by BB is $13,01\%$ for upper bounds and $4,18\%$ for lower bounds with respect to the best known bounds. The results show similar tendency as those for the second set of instances. DF is always better than BB. In this case, the lower bounds reached by DF when h_{IS} is used at any level are always the same due to the fact that these instances are even harder to solve. The upper bounds improve in direct ratio with the level up to h_{IS} is used. However, in this case, it is worth to exploit the improved heuristic in the whole search space. Overall, the improvement in upper bounds quality with respect to BB is more than 40%.

So, we can draw two main conclusions from this experimental study. The first one is that DF is better than BB when both of them use the same heuristic information given by h_{JPS} and the second one is that DF is able to improve when it is given more informed and time consuming heuristics, such as h_{IS}, but in this case we have to be aware of the problem size and exploit this heuristic up to a level that depends on the problem size. In any case, for very large instances it is worth to exploit this heuristic during the whole search.

Conclusions

We have proposed a partially informed depth-first search algorithm to cope with the Job Shop Scheduling Problem with makespan minimization. This algorithm has been designed from the branch and bound algorithm proposed in (Brucker, Jurisch, and Sievers 1994), (Brucker 2004). We have also devised a new heuristic which is monotonic and it is more informed than the heuristic estimation used in the original Brucker's algorithm. We have conducted an experimental study across three sets of medium, large and very large instances from the OR-library. The results show that the proposed algorithm outperforms the original branch and bound algorithm in the three sets. The improvement is due to the fact that the partially informed depth-first search is able to exploit the heuristic knowledge much better than the single branch and bound strategy. We have done some experiments, not reported here, combining branch and bound with the improved heuristic and the results were not good, as the resulting algorithm takes much more time to reach the same solutions, or reaches worse solutions in a given time.

As future work we plan to design new heuristic estimations based on more powerful constraint propagation rules, such as those proposed in (Dorndorf, Pesch, and Phan-Huy 2000), and exploit other search strategies such as IDA^* (Korf 1985) or some combinations of depth-first and best-first strategies in order to improve the lower bounds. We will also try to adapt the algorithms to cope with objective functions other than the makespan, which are in general more interesting from the point of view of real-life problems and, for the largest instances, we will consider weighted and non-admissible heuristics in order to improve the efficiency at the cost of reaching suboptimal solutions.

Acknowledgments

We are grateful to the reviewers for their helpful comments. This research has been supported by the Spanish Ministry of Science and Education under research project MEC-FEDER TIN2007-67466-C02-01 and by the Principality of Asturias under grant FICYT-BP09105.

References

Applegate, D., and Cook, W. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing* 3:149–156.

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

Brucker, P. 2004. *Scheduling Algorithms*. Springer, 4th edition.

Carlier, J., and Pinson, E. 1989. An algorithm for solving the job-shop problem. *Management Science* 35(2):164–176.

Carlier, J., and Pinson, E. 1990. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research* 26:269–287.

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

Carlier, J. 1982. The one-machine sequencing problem. *European Journal of Operational Research* 11:42–47.

Dorndorf, U.; Pesch, E.; and Phan-Huy, T. 2000. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence* 122:189–240.

Garey, M., and Johnson, D. 1979. *Computers and Intractability*. Freeman.

Giffler, B., and Thomson, G. L. 1960. Algorithms for solving production scheduling problems. *Operations Research* 8:487–503.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Pearl, J. 1984. *Heuristics: Intelligent Search strategies for Computer Problem Solving*. Addison-Wesley.

Zhang, C. Y.; Li, P.; Rao, Y.; and Guan, Z. 2008. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research* 35:282–294.

5.2. Depth-first heuristic search for the job shop scheduling problem

Se presenta la siguiente publicación:

- Carlos Mencía, María R. Sierra and Ramiro Varela. Depth-first heuristic search for the job shop scheduling problem. *Annals of Operations Research*. 2012. DOI 10.1007/s10479-012-1296-x.
    - Estado: **Publicado online.**

Ann Oper Res
DOI 10.1007/s10479-012-1296-x

Depth-first heuristic search for the job shop scheduling problem

Carlos Mencía · María R. Sierra · Ramiro Varela

Abstract We evaluate two variants of depth-first search algorithms and consider the classic job shop scheduling problem as a test bed. The first one is the well-known branch-and-bound algorithm proposed by P. Brucker et al. which uses a single chronological backtracking strategy. The second is a variant that uses partially informed depth-first search strategy instead. Both algorithms use the same heuristic estimation; in the first case, it is only used for pruning states that cannot improve the incumbent solution, whereas in the second it is also used to sort the successors of an expanded state. We also propose and analyze a new heuristic estimation which is more informed and more time consuming than that used by Brucker's algorithm. We conducted an experimental study over well-known instances showing that the proposed partially informed depth-first search algorithm outperforms the original Brucker's algorithm.

Keywords Heuristic search · Depth-first search · Constraint propagation · Scheduling

1 Introduction

Over the last decades, scheduling problems have been a core issue for Artificial Intelligence and Operations Research. In this context, the job shop scheduling problem (JSSP) stands out as a paradigm of great interest, due to its proximity to real-life problems and its complexity.

We wish to thank the anonymous reviewers for their useful comments for improving this paper. This research has been supported by the Spanish Government under research project FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP09105.

C. Mencía · M.R. Sierra · R. Varela (✉)
Computing Technologies Group, Department of Computing, University of Oviedo,
Campus de Viesques, 33271 Gijón, Spain
e-mail: ramiro@uniovi.es

C. Mencía
e-mail: menciacarlos@uniovi.es

M.R. Sierra
e-mail: sierramaria@uniovi.es

Its decision version is NP-complete in the strong sense (Garey and Johnson 1979) and it is considered as one of the hardest problems in this class.

The JSSP and a variety of constraint optimization problems have been successfully solved by means of branch and bound algorithms, which interleave the execution of backtracking search and constraint propagation. These algorithms require few memory resources as only one of the successors of the current search state is generated and selected for continuing the search from in the next step. One smart and successful algorithm of this kind, for solving the JSSP with makespan minimization, is that proposed by Brucker et al. (1994a). However, generating only one successor at a time is the very reason of the weakness of backtracking algorithms as the information of the successor states is not fully available at the moment of deciding what path in the search space to follow. So, it can be expected that partially informed depth-first search with complete expansion is able to exploit the heuristic knowledge from the problem domain to guide the search better than backtracking search. Even though the first requires more memory to store the generated but not yet explored states, this amount of memory is limited. At the same time, it may be worth to use more accurate heuristics even though they are more computationally expensive than less informed ones.

In this paper, we start from the implementation of Brucker's algorithm given in Brucker et al. (1994b), termed BB in the following, and design a new exact partially informed depth-first search algorithm, termed DF. We also propose a new heuristic estimation which can be used in both algorithms. We have conducted an experimental study across conventional medium and large instances to compare DF with BB. The results show that DF is able to exploit the available heuristic estimation to a larger extent. For medium-size instances, the time required to solve them is reduced by about 15 %. For larger instances, that in most of the cases cannot be optimally solved by any of the algorithms, DF is able to reach much better lower and upper bounds when both algorithms are given the same time. In this case, the error with respect to the best known solutions is reduced by about 40 %.

The remainder of the paper is organized as follows: In Sect. 2, we give some background on heuristic search and review some related works. In Sect. 3, a formulation of the JSSP is given and some of the solving methods are revised. In Sect. 4, we describe the proposed DF algorithm for the JSSP. In Sect. 5, we present the new heuristic. We include a formal study to justify its properties and efficiency. Section 6 reports the results of the experimental study. Finally, in Sect. 7, we summarize the main conclusions of the paper and propose some ideas for future research.

The present paper is an extended version of Mencía et al. (2010).

2 Heuristic search

A search problem is given by a quintuple $\langle \mathbf{S}, \Gamma, s, SUC, c \rangle$, where $\mathbf{S}$ is the set of states or nodes, $s \in \mathbf{S}$ is the initial state, $\Gamma \subseteq \mathbf{S}$ is the set of goals, SUC is the transition operator such that for a state $n \in \mathbf{S}$, $SUC(n)$ returns the set of successor states of n, and each transition from n to $n' \in SUC(n)$ has a non negative cost $c(n, n')$. The states and the transitions between states conform a graph, which is called *search space*. $P_{s\text{-}n}$ denotes a path from s to n and $P^*_{s\text{-}n}$ denotes one with minimum cost. $g_P(n)$ denotes the cost from s to n through a path P, and $g^*(n) = g_{P^*_{s\text{-}n}}(n)$.

Starting from s and applying systematically the operator SUC, a search algorithm looks for an optimal solution,[1] i.e., a path $P^*_{s\text{-}o}$ with $o = \arg\min\{g_{P^*_{s\text{-}o'}} | o' \in \Gamma\}$, the cost of this

[1] Without loss of generality, we consider minimization problems in this section.

solution is denoted C^*. For this purpose, a search algorithm maintains two lists: CLOSED and OPEN. The list CLOSED contains the states that have been already expanded, i.e., all their successors have been generated, whereas the list OPEN stores those states that have been generated but not yet expanded. At each step, the search algorithm selects a state for expansion from the list OPEN following a *search strategy*, having this decision implications in the completeness, admissibility and efficiency of the algorithm.

The performance of a search algorithm can be dramatically improved by exploiting the knowledge from the problem domain. This knowledge is usually represented by means of a heuristic evaluation function f, such that for each state n, $f(n)$ gives a measure of its promise. If f is used to select for expansion the most promising state in the whole list OPEN, the search strategy is called *best-first search*. The value $f(n)$ can be viewed as an estimation of $f^*(n) = g^*(n) + h^*(n)$, where $h^*(n)$ is the optimal cost from n to its nearest goal. So, it can be defined as $f(n) = g(n) + h(n)$ for all states n, where $g(n)$ is the cost of the best path from s to n found so far and $h(n)$ is a heuristic estimation for $h^*(n)$. In this case, we have the A^* algorithm proposed by Hart et al. (1968) and very well described by Nilsson (1980) and Pearl (1984).

The A^* algorithm has some interesting properties that depend on the function h. When $h(n) \le h^*(n)$ for all states n, the algorithm is admissible and then it guarantees that the first goal reached represents an optimal solution. Moreover, if we have two admissible heuristics h_1 and h_2 such that $h_1(n) < h_2(n)$ for all non goal states n, h_2 is more informed than h_1 and then the number of nodes expanded by A^* to reach a solution with h_2 is not greater than it is with h_1. Another interesting property is monotonicity, i.e., $h(n_1) \le h(n_2) + c(n_1, n_2)$, for all states n_1, n_2. This property is equivalent to consistency defined as $h(n_1) \le h(n_2) + k(n_1, n_2)$, for all states n_1 and n_2, where $k(n_1, n_2)$ denotes the weighted cost of the best path from n_1 to n_2. Two consequences of monotonicity are admissibility and that $g(n) = g^*(n)$ when n is expanded. This is quite important when the search space is not a tree, as it guarantees that none of the states needs to be reexpanded.

2.1 Tree search

For search spaces with tree structure where detection of duplicate states is not required, depth-first search is a strategy of common use due to the low amount of memory it consumes. In this case, the list CLOSED is not necessary and the list OPEN is managed as a LIFO stack. Whenever a leaf is reached the search backtracks and continues from a shallower state. This is an any-time algorithm and so a sequence of improving solutions is obtained along the search. When a new state n is reached for the first time with $f(n) \ge UB$, where UB is the cost of the incumbent solution, the subtree rooted by n is pruned. The algorithm finishes when the list OPEN gets empty, in this case certifying the optimality of the solution, or after a given time returning a solution that might not be optimal. In this latter case, the lowest f-value in OPEN is a lower bound on the cost of the optimal solution.

A popular variant of depth-first search is backtracking search which generates only one successor $n' \in SUC(n)$ at a time and selects it for the next step. As it was pointed in Pearl (1984), the main advantage of backtracking is the low use of memory as the only information that has to be stored is the branch from the initial state to the current one, together with some annotations used to identify the next successor to be generated for each state. Also, a backtracking algorithm can be implemented very efficiently, as it does not have to duplicate the information of a state for all its successors.

One of the main drawbacks of depth-first search strategies is the big effort that they have to make to recover from wrong decisions. In any step of the search, when a "bad" successor (i.e., a state not leading to good solutions) is selected before a better one, the algorithm must traverse the whole subtree under this successor before proceeding the search from a better option. This problem becomes worse at shallow levels of the search tree. So, it is important to use the available knowledge from the problem domain in order to choose the most promising successor for continuing the search from. This is precisely what an informed DF algorithm does: when a state n is expanded, all its successors $n' \in SUC(n)$ are generated and stored at the beginning of the list OPEN, sorted according to a heuristic. Hence, DF can exploit the knowledge to a greater extent than backtracking to guide the search. In particular, it can apply constraint propagation rules to each successor before deciding what path to follow.

3 The job shop scheduling problem

In this section we present the job shop scheduling problem, the disjunctive graph model and introduce some notation. We also briefly review some of the most effective solving methods and some extensions of the problem based on real life characteristics.

3.1 Problem formulation

The job shop scheduling problem (JSSP) requires scheduling a set of N jobs $\{J_1, \dots, J_N\}$ on a set of M resources or machines $\{R_1, \dots, R_M\}$. Each job J_i consists of a set of tasks or operations $\{\theta_{i1}, \dots, \theta_{iM}\}$ to be sequentially scheduled. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, a fixed duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. The JSSP has three kinds of constraints: precedence, capacity and non-preemption. Precedence constraints translate into linear inequalities of the type: $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$. Capacity constraints translate into disjunctive constraints of the form: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, being v and w two operations requiring the shame machine. Non-preemption requires assigning machines to operations without interruption during their whole processing time. The objective is to come up with a feasible schedule such that the completion time of the last operation, i.e., the *makespan*, is minimized. This problem is denoted as $J\|C_{max}$ in the $\alpha|\beta|\gamma$ notation (Graham et al. 1979).

3.2 The disjunctive graph model

From now on, a problem instance will be represented by a disjunctive graph $G = (V, A \cup E)$ (Roy and Sussman 1964). Each node in the set V represents an actual operation, with the exception of the dummy nodes *start* and *end*, which represent operations with processing time 0. The arcs of A are called *conjunctive arcs* and represent precedence constraints and the arcs of E are called *disjunctive arcs* and represent capacity constraints. E is partitioned into subsets E_i with $E = \bigcup_{i=1,\dots,M} E_i$. E_i includes an $arc(v, w)$ for each pair of operations requiring R_i. The arcs are weighted with the processing time of the operation at the source node. Node *start* is connected to the first operation of each job and the last operation of each job is connected to node *end*. Figure 1(a) shows the disjunctive graph for an instance with 3 jobs and 3 machines.

A feasible schedule S is represented by an acyclic subgraph G_S of G, $G_S = (V, A \cup H)$, with $H = \bigcup_{i=1,\dots,M} H_i$, where H_i represents a linear processing ordering for the operations requiring R_i. For two operations u and v requiring R_i, the arc (u, v) is included in H_i iff u is processed before v in S. The makespan is the cost of a *critical path* and it is denoted as $L(S)$. A critical path is a longest path from node *start* to node *end* and contains a set of

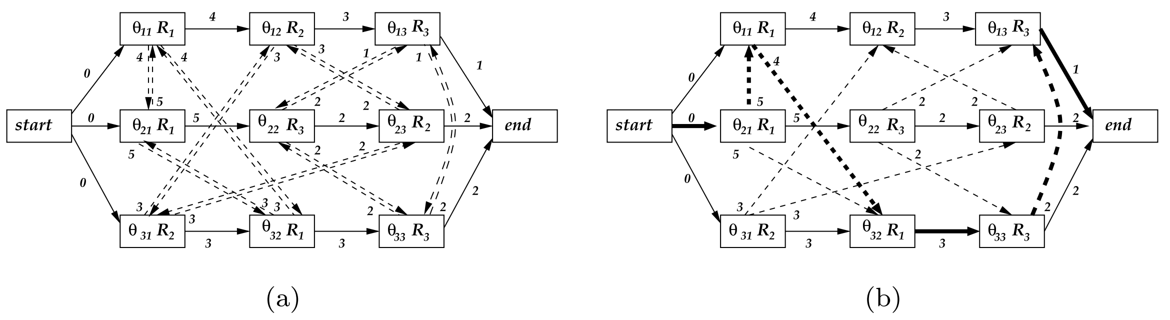


Fig. 1 (**a**) Disjunctive graph for a problem instance with 3 jobs and 3 machines and (**b**) a feasible schedule to this problem. *Bold face arcs* show a critical path with cost (makespan) 15

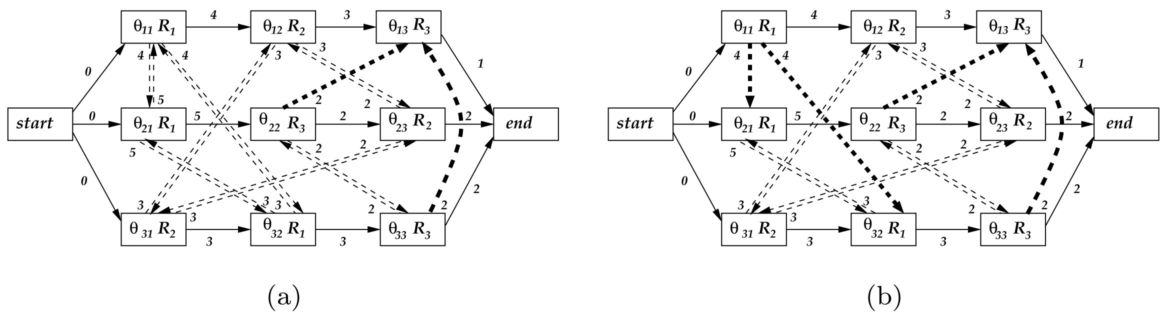


Fig. 2 Two partial solution graphs for the problem instance in Fig. 1(a)

critical blocks. A critical block is a maximal sequence of consecutive operations in a critical path requiring the same machine. Figure 1(b) shows a solution graph for the instance of Fig. 1(a).

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v, i.e., it is the value of st_v. The *tail* q_v is defined so as the value $q_v + p_v$ is the cost of the longest path from v to *end*. Hence, $r_v + p_v + q_v$ is the makespan if v is in a critical path, otherwise, it is a lower bound. JP_v and JS_v denote the predecessor and successor operation of v respectively on the job sequence.

A partial schedule is given by an acyclic subgraph of G where some of the disjunctive arcs are not fixed yet. Figure 2 shows two partial solution graphs for the instance of Fig. 1(a). In a partial schedule, heads and tails can be estimated as

$$r_v = \max\left\{\max_{J \subseteq DP_v}\left\{\min_{j \in J} r_j + \sum_{j \in J} p_j\right\}, r_{JP_v} + p_{JP_v}\right\} \tag{1}$$

$$q_v = \max\left\{\max_{J \subseteq DS_v}\left\{\sum_{j \in J} p_j + \min_{j \in J} q_j\right\}, p_{JS_v} + q_{JS_v}\right\} \tag{2}$$

with $r_{start} = q_{end} = 0$ and where DP_v denotes the set of disjunctive predecessors of v, i.e., operations requiring machine R_v which are scheduled before v. Analogously, DS_v denotes the disjunctive successors of v. For example, the head of operation θ_{13} is 9 in the partial schedule of Fig. 2(a), being $J = \{\theta_{22}, \theta_{33}\}$ the subset of disjunctive predecessors giving the largest value in expression (1), and the tail of operation θ_{11} is 11 in the partial solution of Fig. 2(b), being $\{\theta_{21}, \theta_{32}\}$ the subset of disjunctive successors giving the largest value in expression (2).

An efficient representation for this solution graph that enables easy operation on the problem data to obtain critical blocks and to perform transformations as reversing critical arcs is given in Blazewicz et al. (2000).

3.3 Existing solution methods

A broad variety of solving methods have been proposed for the $J||C_{max}$ problem along of more than four decades up to date. In Blazewicz et al. (1996), the authors review the history and proposed solving methods in the first three decades. Designed from seminal ideas proposed by Van Laarhoven et al. (1992) or Dell'Amico and Trubian (1993), sophisticated local search methods are considered the current state-of-the-art techniques; among them, the tabu search algorithms proposed by Nowicki and Smutnicki (2005), and Zhang et al. (2008) deserve special mention.

Also, effective exact methods have been developed within the constraint-based reasoning community. A recent review of general constraint reasoning techniques can be found in Meseguer (2012). Regarding scheduling problems, these methods combine search with powerful constraint propagation algorithms that are able to reduce the search space to a great extent by reasoning and deducing properties that hold in any improving schedule w.r.t. the incumbent solution. Some examples are timetabling, edge-finding and energetic reasoning (Dorndorf et al. 2000; Laborie 2003; Baptiste et al. 2001). Regarding search strategies, backtracking has been widely used in this domain, and a number of variable and value ordering heuristics, such as texture-based (Beck and Fox 2000) or slack-based heuristics (Smith and Cheng 1993), have been proposed. In this context, Brucker's branch and bound algorithm (Brucker et al. 1994a) is one of most effective methods for finding and proving optimal solutions to medium-size instances. Although it was proposed several years ago, it is still often chosen as a building method for developing new research ideas on. For example, it was used for testing an efficient query strategy (Streeter and Smith 2007) and also in Gharbi and Labidi (2010) to obtain lower bounds by solving non-polynomial relaxations. A similar branch and bound algorithm to Brucker's was used in Dorndorf et al. (2002) where the authors propose an edge-guessing procedure that exploits a number of constraint propagation rules in a preprocessing step in order to speed up the search to obtain near-optimal solutions.

For larger and more difficult instances that cannot be solved in a reasonable time, Solution Guided Multi-Point Constructive Search (SGMPS) (Beck 2007) stands out as the state-of-the-art constructive method in the quality of the solutions returned. Finally, hybrid algorithms combining exact and local search approaches have been proposed recently. Streeter and Smith (2006), combine an iterated local search procedure with Brucker's algorithm. Watson and Beck (2008) propose combining SGMPS with tabu search.

3.4 Brucker's algorithm

As we have pointed, one of the best exact methods[2] proposed so far to cope with the $J||C_{max}$ problem is the branch and bound algorithm due to Brucker et al. (1994a), Brucker (2004). This algorithm starts from the constraint graph for a given problem instance and proceeds to fix disjunctive arcs subsequently. The key feature of the algorithm is a smart branching scheme that relies on fixing arcs either in the same or in the opposite direction as they appear

[2]We mean *exact method* from a practical point of view, i.e., an algorithm capable of efficiently finding and proving optimal solutions to the problem, at least for medium-size instances.

in a critical block of a feasible solution. Moreover, the algorithm exploits powerful methods to obtain accurate lower and upper bounds.

Lower bound calculation is based on preemptive one machine sequencing problem relaxations. The optimal solution to the simplified problem is given by the so-called Jackson's Preemptive Schedule (JPS), which is obtained in polynomial time by the algorithm proposed in Carlier (1982). Upper bounds are obtained by means of the greedy $G\&T$ algorithm (Giffler and Thompson 1960).

Finally, Brucker's algorithm exploits a constraint propagation method termed *immediate selection* (Carlier and Pinson 1989). This method fixes additional disjunctive arcs so as the effective search tree is dramatically reduced. Brucker's algorithm can easily solve almost any instance up to 10 jobs and 10 machines as well as many larger instances.

3.5 Real life job shop scheduling

Some real life scheduling problems have been modeled as a JSSP. For example in Meeran and Morshed (2011), a work shop at a steel mill company is modeled as a classical JSSP and the problem is solved by means of a tabu search algorithm. However, in many cases some additional characteristics taken from real scenarios are considered. For example, scheduling problems with sequence-dependent setup times were faced by some researches by means of branch and bound algorithms (Brucker and Thiele 1996; Artigues and Feillet 2008) or memetic algorithms (Vela et al. 2010). The JSSP with lateness minimization is confronted in Balas et al. (2008) combining the shifting bottleneck heuristic with guided local search and in González et al. (2012) by means of tabu search. The JSSP with operators is proposed in Agnetis et al. (2011) and solved with dynamic programming and some heuristics. This problem is solved in Mencía et al. (2011) by means of a genetic algorithm and in Sierra et al. (2011) by means of an enhanced A* algorithm that minimizes total flow time. A variant of this problem is considered in Guyon et al. (2012). Also, a real JSSP from a pottery company with uncertain processing times have been considered in Petrovic et al. (2008) and solved by multiobjective memetic algorithms. Similar problems have been confronted in González Rodríguez et al. (2008, 2009) by means of local search and genetic algorithms.

4 Partially informed depth-first search algorithm for the JSSP

In this section, we present the proposed partially informed depth-first search algorithm. Its general structure is given in Algorithm 1. In each iteration the algorithm takes the first state n in the *OPEN* list and one of the solutions, S, reachable from n is computed (see Sect. 4.3). Then n is expanded. Each successor q_i of n is improved by constraint propagation (procedure *IS*) to obtain a new state q_i' having more disjunctive arcs fixed. If the resulting state q_i' is feasible, it is considered for expansion. The new states are sorted by non decreasing f-values and appended at the beginning of *OPEN*. The algorithm finishes when *OPEN* is empty, in this case returning an optimal solution, or after a given time returning the incumbent solution and a lower bound on the optimal solution.

In the next subsections, we describe each one the main components of this algorithm, some of which are borrowed from Brucker's algorithm.

4.1 States representation

A search state n is given by a partial solution graph $G_n = (V, A \cup FD_n)$, where FD_n denotes the disjunctive arcs fixed in n. In the initial state $FD_n = \oslash$. Figure 2 shows two possible

Algorithm 1 *DF* (instance p). Calculates a solution for p with cost *UB* and a lower bound *LB* on the optimal solution of p

```
 1: $OPEN = Initial(p)$;
 2: while $((OPEN \neq \emptyset)$ and $(no\ time\ limit))$ do
 3:    $n = Pop(OPEN)$;
 4:    $S = G\&T(n)$; // Update $UB$ from $S$
 5:    $SUC = Expand(n)$; // $SUC = \{q_1, \ldots, q_p\}$
 6:    $SUC' = \emptyset$;
 7:    for $q_i \in SUC$ do
 8:       $q'_i = IS(q_i, UB)$; // Fixes additional arcs
 9:       if $q'_i$ is feasible, calculate $f(q'_i)$ and add $q'_i$ to $SUC'$ if $f(q'_i) < UB$;
10:    Sort $SUC'$ by non decreasing $f$-values;
11:    $OPEN = Append(SUC', OPEN)$;
12: return The incumbent solution with cost $UB$ and the lowest $f(q), q \in OPEN$;
```

search states. In the state of Fig. 2(a), only the arcs $(\theta_{33} \rightarrow \theta_{13})$ and $(\theta_{22} \rightarrow \theta_{13})$ are fixed, while in the state of Fig. 2(b) there are four arcs fixed: $(\theta_{33} \rightarrow \theta_{13})$, $(\theta_{22} \rightarrow \theta_{13})$, $(\theta_{11} \rightarrow \theta_{21})$ and $(\theta_{11} \rightarrow \theta_{32})$.

As the cost of a solution S is given by a longest path in G_S, we have opted to take $g^*(n)$ as the cost of the longest path from node *start* to node *end* in G_n. Consequently, $g_{P_{s\text{-}n}}(n) = g^*(n)$ for all paths $P_{s\text{-}n}$ and so $g(n) = g^*(n)$ when node n is reached for the first time. In the examples given in Fig. 2, these values are 9 and 13 respectively. This definition of g^* presents a small drawback as $g^*(initial) > 0$. However, this could be easily avoided by defining a predecessor to the initial state of the form $(V, \oslash)$, i.e., a state in which the conjunctive arcs were not included.

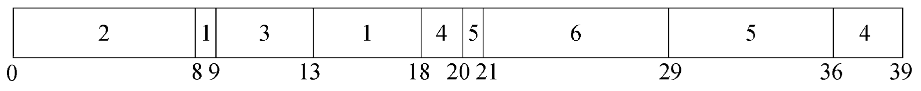
Heads and tails in n are calculated by expressions (1) and (2). In Brucker et al. (1994b), algorithms for computing these values in polynomial time are given. For instance, the algorithm for computing the head of an operation v in a state n is the following: the disjunctive predecessors of v, $DP_v(n)$, are sorted by non decreasing heads as $(v_1, \ldots, v_k)$, and the head of v is initialized as $r_v = r_{v_1}$. The predecessors are then visited in order and for each v_i, the head of v is updated as $r_v = \max\{r_v, r_{v_i}\} + p_{v_i}$. To compute the new heads for all operations, the algorithm above is applied to each operation following a topological order in G_n. A similar algorithm is used to compute tails. The overall algorithm is termed *Computing new Heads and Tails*.

4.2 Heuristic estimation

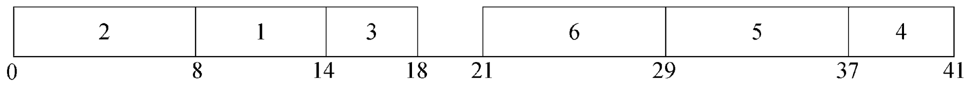
For each generated state n, a heuristic estimation $f(n)$ of the cost of the best solution reachable from n (i.e., the cost of the best path from the initial state to a goal constrained to pass through n) is computed. This estimation is based on problem splitting and constraint relaxations in the following way. Let I be the set of operations requiring the machine m. Scheduling operations in I accordingly to their heads and tails in n so as the value $\max_{v \in I}(st_v + p_v + q_v)$ is minimized is known as the One Machine Sequencing problem (OMS). The optimal solution to this problem is clearly a lower bound of $f^*(n)$. However, the OMS is still NP-hard. So, a new relaxation is required in order to obtain a polynomially solvable problem. To do that, it is common to relax the non-preemption constraint. An optimal solution to the preemptive OMS is given by the Jackson's Preemptive Schedule (JPS) (Carlier 1982).

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	13	16

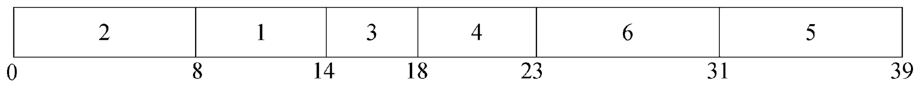
a) An OMS problem instance



b) A JPS with makespan 49 given by completion time of job 5 (36 + 13)



c) An Optimal non Preemptive Solution with makespan=50 given by completion time of job 4 (41+9)



d) A Suboptimal Solution (UB) without idle times with cost 52 given by completion time of job 5 (39+13)

Fig. 3 The Jackson's Preemptive Schedule for an OMS problem instance

The JPS is calculated by the following algorithm: at any time t given by a head or the completion of an operation, from the minimum r_v until all operations are completely scheduled, schedule the ready operation with the largest tail on machine m. Calculating the JPS has a complexity of $O(|I| \times \log |I|)$. Finally, $f(n)$ is taken as the largest JPS over all machines. So, $h(n) = f(n) - g(n)$ is an optimistic estimate of $h^*(n)$. From now on, $f_{JPS}(n)$ and $h_{JPS}(n)$ denote these heuristic estimations.

Figure 3 shows an OMS instance, its optimal solution, a suboptimal solution and a JPS for it. As we can observe in the JPS, some of the operations are processed in two or more time intervals due to allowing preemption. The cost of the JPS is given by the value $\max\{C_i + q_i\}$, where C_i denotes the completion time of the operation i in the JPS.

4.3 Upper bounds

As in Brucker's algorithm, we carry out upper bound calculations in the depth-first search counterpart. To do that, a variant of the well-known $G\&T$ greedy algorithm proposed in Giffler and Thompson (1960) is used. The algorithm is issued from each expanded node so as it builds a schedule that includes all disjunctive arcs fixed in that state. Note that the disjunctive arcs fixed to obtain the schedule do not remain fixed in that node. $G\&T$ is a greedy algorithm that produces a schedule in a number of $N * M$ steps. Algorithm 2 shows the $G\&T$ algorithm adapted to obtain upper bounds from a search state n. In each iteration, the algorithm considers the set A comprising all operations v that can be scheduled next, i.e., those whose disjunctive predecessors DP_v and conjunctive predecessor JP_v are already scheduled (initially only operation *start* is scheduled). The operation v^* in A with the earliest completion time if it is scheduled next is determined. Then, a new set B is obtained with all operations v in A requiring the same machine as v^* that can start at a time lower than $r_{v^*} + p_{v^*}$. Any operation in B can be scheduled next and the selected one is w^* if it produces the least cost JPS for the remaining unscheduled operations on the same machine. Finally, the algorithm returns the built schedule S and its makespan. From these values the incumbent solution and its cost UB are updated.

Algorithm 2 *G&T* (state n). Calculates a heuristic solution S from a state n. O denotes the set of all operations and SC the set of scheduled operations

```
1:  $SC = \{start\}$;
2:  while ($SC \neq O$) do
3:      $A = \{v \in O \setminus SC; DP_v \cup \{JP_v\} \subset SC\}$;
4:      $v^* = \arg\min\{r_u + p_u; u \in A\}$;
5:      $B = \{v \in A;\ R_v = R_{v^*}$ and $r_v < r_{v^*} + p_{v^*}\}$;
6:      $C = \{v \in O \setminus SC;\ R_v = R_{v^*}\}$;
7:      $w^* = \arg\min\{$makespan of JPS$(C \setminus \{w\})$ after scheduling $w;\ w \in B\}$;
8:      Schedule $w^*$ in $S$ at time $r_{w^*}$;
9:      Add $w^*$ to $SC$ and update heads of operations not included in $SC$;
10: return $S$ and its makespan;
```

4.4 Expansion mechanism

As we have pointed, the expansion of a state n is done by fixing new disjunctive arcs in the partial solution graph G_n. These arcs are selected from a critical path of some schedule compatible with G_n. The schedule built by Algorithm 2 is chosen for that purpose. Broadly speaking, for generating the successors of a given state n, the expansion mechanism follows the next steps:

1. Compute a feasible schedule S compatible with G_n by means of Algorithm 2.
2. Compute a critical path P in S and the critical blocks in P.
3. A successor node is created by moving an operation in a critical block B before or after the remaining operations in B. In this process, a number of disjunctive arcs get fixed as it is explained below.

This expansion mechanism is based on the following result given in Brucker et al. (1994a).

Theorem 1 *Let S and S' be two schedules. If $L(S') < L(S)$, then one of the following conditions holds*:

(1) *at least one operation v in a critical block B in G_S, different from the first operation of B, is processed in S' before all operations of B*;
(2) *at least one operation v in a critical block B in G_S, different from the last operation of B, is processed in S' after all operations of B.*

The details of the expansion mechanism together with some examples are given in Appendix A.

4.5 Fixing additional arcs by constraint propagation

After adjusting heads and tails, new disjunctive arcs can be fixed by the constraint propagation method due to Carlier and Pinson (1989), termed immediate selection,[3] which is based on the following results. Let I be the set of operations requiring a given machine. For each operation $j \in I$, r_j and q_j denote the head and tail respectively of the operation j in a given state.

[3]The immediate selection procedure was the first of the so-called edge-finding methods.

Algorithm 3 PROCEDURE Select

```
1: for all $c, j \in I, c \neq j$ do
2:     if $r_c + p_c + p_j + q_j \geq UB$ then
3:         fix the arc $(j \rightarrow c)$;
```

Algorithm 4 Immediate Selection

```
1: Calculation of all primal arcs for all machines;    {Algorithm Improving Heads + Procedure Select;}
2: Calculation of new heads and tails;    {Algorithm Computing New Heads and Tails;}
3: Calculation of all dual arcs for all machines;    {Algorithm Improving Tails + Procedure Select;}
4: Calculation of new heads and tails;    {Algorithm Computing New Heads and Tails;}
```

Theorem 2 *Let $c, j \in I, c \neq j$. If $r_c + p_c + p_j + q_j \geq UB$, then j has to be processed before c in every solution reachable from the state n that improves UB.*

The arc $(j \rightarrow c)$ is called *direct arc*. The procedure *Select* given in Algorithm 3 calculates all direct arcs for the state n.

Theorem 3 *Let $c \in I$ and $J \subseteq I \backslash \{c\}$.*

(1) *If*

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB, \tag{3}$$

then in all solutions reachable from state n improving UB, the operation c has to be processed after all the operations in J.

(2) *If*

$$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J \cup \{c\}} q_j \geq UB, \tag{4}$$

then in all solutions reachable from state n improving UB, the operation c has to be processed before all the operations in J.

If condition (1) of the theorem above holds, then the arcs $\{j \rightarrow c;\ j \in J\}$ can be fixed. These arcs are called *primal arcs* and the pair (J, c) is called *primal pair*. Similarly, if condition (2) holds, the *dual arcs* $\{c \rightarrow j;\ j \in J\}$ can be fixed and (c, J) is called a *dual pair*. In Brucker et al. (1994a), the procedure *Immediate Selection* (*IS*) is derived to calculate all primal and dual arcs. This method is given in Algorithm 4 and it is based on the algorithms *Improving Heads* and *Improving Tails* which are described in Appendix B.

Steps (1) and (3) improve heads and tails respectively and fix disjunctive arcs by Procedure *Select*. Then new heads and tails can be obtained in (2) and (4) due to the new heads and tails. So, in principle steps (1) to (4) should be repeated as long as new disjunctive arcs get fixed, until a fix point is reached. However, after (2) or (4) an inconsistent situation might appear. Let n be a state and $r_i(n)$ and $q_i(n)$ the head and tail respectively of operation i after steps (2) or (4), n is inconsistent if one of the two following conditions holds

(1) The partial solution graph G_n contains a cycle
(2) A lower bound larger or equal than UB can be derived from n.

In order to check the second condition we will use the lower bound given by $f_{JPS}(n)$. Cycles in G_n are detected by the algorithm *Computing New Heads and Tails*. As we have pointed in Sect. 4.1, this algorithm starts computing a topological ordering of G_n and such an order does not exist if the graph contains some cycle. To represent a call to this procedure we will use the notation $n_{UB} = IS(n, UB)$, where n is a state just after expansion as before or after candidate and UB is the current upper bound. If n_{UB} is consistent, n is substituted by n_{UB} as all arcs fixed in n_{UB} must also be fixed in any solution reachable from n having cost lower than UB; otherwise, n is pruned. In order to simplify notation, in the following, n will represent a search state after immediate selection.

4.6 Search strategy

As we have pointed, Brucker's algorithm uses a backtracking search strategy and a dispatching rule for selecting the next successor node. When exploring a node, it computes the before-candidates and after-candidates in a critical path, it then generates only one successor at a time which is explored next, giving priority to before-candidates (after-candidates) with the smallest heads (tails). By doing so, it only takes profit from the heuristic estimations (lower bounds) for pruning those states n having $f(n) \geq UB$.

We propose here to use a partially informed depth-first search strategy. In this way, the selection of the state to be explored next is based on more heuristic knowledge than that of previous dispatching rule. We have chosen to use that rule just for breaking ties in f-values as it has given better results than other methods such as considering g-values or the number of arcs fixed in the node.

As it was pointed in Sect. 2, the algorithm terminates when the list OPEN becomes empty or after a given time returning the incumbent solution with cost UB and a lower bound of the cost of the optimal solution LB. In the first case $UB = LB$ as an optimal solution was reached, while in the second case $UB \geq LB$ as the solution reached is not guaranteed to be optimal.

5 Heuristic improvement by constraint propagation

To improve the heuristic estimations, we use the following strategy taken from Carlier and Pinson (1990) for computing lower bounds for the JSSP. It has also inspired the effective *shaving* technique proposed in Martin and Shmoys (1996) for reducing the time window of each operation in a search state.

If it can be proved that a solution with cost lower than $P > f(n)$ may not be reachable from a state n, then the heuristic estimation $f(n)$ can be updated to P. In order to do that, we assume a tentative upper bound P and apply the procedure IS to obtain a modified state denoted as n_P, i.e., $n_P = IS(n, P)$. It is clear that n_P might not be a proper state of the search tree; however, a solution S with cost lower than P can be reached from state n iff S is also reachable from n_P. So, if n_P is inconsistent, no solution with cost lower than P can be obtained from n and P is then a lower bound for the cost of the best solution that can be found from the search state n.

Therefore, the best lower bound that can be established by this method is the largest P such that n_P is inconsistent. So the new estimation, termed f_{IS}, is defined as

Definition 1 For every state $n \in \mathbf{S}$

$$f_{IS}(n) = \max\left\{P' \middle| n_{P'} \text{ is inconsistent}\right\}. \tag{5}$$

Now, we will try to devise an efficient algorithm to compute $f_{IS}(n)$. The calculation of $f_{IS}(n)$ can be restricted to an interval from the following observations: n_P is trivially inconsistent for $P = f_{JPS}(n)$. Also, as n is obtained just after immediate selection from the current upper bound UB, then n_P is consistent for $P = UB$. Then, it follows that

Proposition 1 $f_{IS}(n) \in [f_{JPS}(n), UB - 1]$.

To speed up the search across this interval, we will prove that if n_P is inconsistent then n_{P-1} is also inconsistent. Therefore, the search for the largest P such that n_P is inconsistent could be reduced to a dichotomic search. In order to prove that we will see that if $IS(n, P)$ produces an inconsistency at a given iteration k, then $IS(n, P - 1)$ produces an inconsistency at iteration $k' \leq k$ as well.

Let n_1 and n_2 be two states fulfilling the following two conditions, where $\subseteq_{SG}$ denotes "is a subgraph of" relation.

$$r_i(n_1) \leq r_i(n_2) \quad \text{and} \quad q_i(n_1) \leq q_i(n_2), \quad \forall i \tag{6}$$

$$G_{n_1} \subseteq_{SG} G_{n_2}. \tag{7}$$

We start proving that if we apply any of the four steps of the Algorithm *Immediate Selection* to n_1 and n_2, with tentative upper bounds P_1 and $P_2 \leq P_1$ respectively, these two conditions are preserved. This result will also be used in the next subsection for proving the monotonicity of h_{IS}.

Lemma 1 *If the procedure Improving Heads is applied to n_1 and n_2 with tentative upper bounds P_1 and $P_2 \leq P_1$ respectively, and then new arcs are fixed by the procedure Select, previous conditions* (6) *and* (7) *hold.*

Proof See Appendix C. □

An analogous result can be proved regarding the algorithm *Improving Tails* followed by Procedure *Select*. We prove now that both conditions hold after Algorithm *Computing New Heads and Tails*.

Lemma 2 *If the algorithm Computing New Heads and Tails is applied to states n_1 and n_2, conditions* (6) *and* (7) *are preserved.*

Proof Condition (7) trivially holds as the graphs are not modified by the algorithm. To prove that condition (6) holds it is enough to observe that for any operation v the set of disjunctive predecessors in n_1, denoted $DP_v(n_1)$, is a subset of $DP_v(n_2)$ and that $r_i(n_1) \leq r_i(n_2)$ for all $i \in DP_v(n_1)$. So, the new heads for v in n_1 and n_2 respectively, computed by the algorithm given in Sect. 4.1, fulfill $r_v(n_1) \leq r_v(n_2)$. Analogous reasoning may be done for the new tails. So, condition (6) is also preserved after *Computing New Heads and Tails*. □

Now, let us consider the application of *Immediate Selection* to states n_1 and n_2 with tentative upper bounds P_1 and $P_2 \leq P_1$ respectively.

Lemma 3 *If $IS(n_1, P_1)$ produces an inconsistency after one of the steps* (2) *or* (4) *at iteration k, then if $IS(n_2, P_2)$ has not produced an inconsistency in an iteration $k' < k$, then it will produce an inconsistency in iteration k.*

Proof Let n_{1P_1} be the state at the time an inconsistency is produced when running $IS(n_1, P_1)$, i.e., the state in iteration k just after one of the steps (1)–(4). Let us assume that $IS(n_2, P_2)$ has not produced any inconsistency and so let n_{2P_2} be the state generated by $IS(n_2, P_2)$ at this time. From Lemmas 1 and 2, states n_{1P_1} and n_{2P_2} fulfill conditions (6) and (7). Hence, if $G_{n_{1P_1}}$ has a cycle, then $G_{n_{2P_2}}$ has a cycle as well, as $G_{n_{1P_1}}$ is a subgraph of $G_{n_{2P_2}}$. Moreover, due to condition (6), any feasible preemptive schedule for a machine in n_{2P_2} is a feasible preemptive schedule for the same machine in n_{1P_1} as well, then $f_{JPS}(n_{2P_2}) \geq f_{JPS}(n_{1P_1})$. So, if $f_{JPS}(n_{1P_1}) \geq P_1$ then $f_{JPS}(n_{2P_2}) \geq P_2$. □

Therefore, from Lemma 3 we have finally the following result considering $n_1 = n_2 = n$, $P_1 = P$ and $P_2 = P - 1$.

Proposition 2 *If n_P is inconsistent then n_{P-1} is inconsistent.*

Hence, the calculation of $f_{IS}(n)$ can be restricted to a dichotomic search in the interval $[f_{JPS}(n)+1, UB-1]$. The new heuristic, denoted h_{IS}, is obtained as $h_{IS}(n) = f_{IS}(n) - g(n)$.

5.1 Properties of h_{IS}

From the definition above, it follows that $h^*(n) \geq h_{IS}(n) \geq h_{JPS}(n)$, for every state n. So, it may be expected that f_{IS} prunes more states than f_{JPS} from the condition $f(n) \geq UB$ and that the search is guided towards better regions of the search space when f_{IS} is used in combination with a partially informed depth-first search algorithm.

We now prove that h_{IS} is monotonic. This property is evident in the case of h_{JPS} as it is obtained from an optimal solution to a relaxed problem (Pearl 1984).

In principle, no property for depth-first search algorithms can be derived from monotonicity, however it is well known that it is quite relevant when the heuristic is used in combination with exact algorithms such as A^* or IDA^*. In spite of that we will see that the monotonicity of h_{IS} has positive consequences on the efficiency of the depth-first search algorithm as well.

Let us start establishing an equivalent property for monotonicity in the case that for every node n, every path from the initial state to n has the same cost.

Lemma 4 *If $g_{P_{s-n}} = g^*(n)$, for all P_{s-n}, then the following conditions are equivalent*

1. *h is monotonic.*
2. *$f(n) \leq f(n')$, for all $n' \in SUC(n)$.*

Proof (1. ⇒ 2.) If h is monotonic, then $f(n) = g^*(n) + h(n) \leq g^*(n) + c(n, n') + h(n') = g^*(n') + h(n') = f(n')$.

(2. ⇒ 1.) If $f(n) \leq f(n')$, $g^*(n) + h(n) \leq g^*(n') + h(n')$ then $h(n) \leq g^*(n') - g^*(n) + h(n') = c(n, n') + h(n')$, hence h is monotonic. □

Let us now consider two states n and $n' \in SUC(n)$. It is clear that conditions (6) and (7) hold for $n_1 = n$ and $n_2 = n'$. So, from Lemma 3, considering $P_1 = P_2 = P$, it follows the next result.

Lemma 5 *If n_P is inconsistent then n'_P is inconsistent.*

From this result and the definition of f_{IS}, the following corollary may be derived.

Corollary 1 $f_{IS}(n) \leq f_{IS}(n')$.

Which, due to Lemma 4, is equivalent to

Theorem 4 *h_{IS} is monotonic.*

5.2 Consequences of the monotonicity of h_{IS}

We can establish the following two consequences of monotonicity. The first is that the interval for searching the value of f_{IS} in expression (5) can be reduced. The second is that for any monotonic heuristic the sequence of lower bounds registered in the list OPEN, i.e., the minimum f-values, is non decreasing.

Proposition 3 *Let n' be a successor of n and UB the upper bound at the time n is expanded, then $f_{IS}(n') \in [\max(f_{JPS}(n'), f_{IS}(n)), UB - 1]$, and so the dichotomic search can be restricted to the interval $[\max(f_{JPS}(n'), f_{IS}(n)) + 1, UB - 1]$.*

Proof It follows from Lemma 4 as $f_{IS}(n') \geq f_{IS}(n)$. □

Proposition 4 *Let n be a node selected for expansion. If LB and LB' are the values of the expression $\min\{f(n')|n' \in OPEN\}$ just before and just after n is expanded respectively, then $LB' \geq LB$.*

Proof From Lemma 4, $f(n') \geq f(n)$ for all n' successor of n. If $LB < f(n)$ or $LB = f(n)$ and n is not the only node in OPEN such that $LB = f(n)$, then the lower bound does not change with the expansion of n and $LB' = LB$. Otherwise, if n is the only node in OPEN such that $LB = f(n)$, then clearly $LB' \geq LB$. □

Here it is worth to remark that the value $\min\{f(n')|n' \in \text{OPEN}\}$ might not be a proper lower bound, as this value might be larger than the cost of the incumbent solution, UB. However, in this case $UB = C^*$ and the algorithm will discard all nodes in OPEN due to $f(n) > UB$ and so it will terminate after OPEN gets empty.

5.3 Heuristic refinement

In accordance with some ideas given in Brinkkötter and Brucker (2001) we can consider a further refinement for calculating the heuristic estimation. We formalize these ideas in the following results.

Theorem 5 *If n_P is consistent then $f_{JPS}(n_P) \leq f^*(n)$.*

Proof If n_P is consistent then $f_{JPS}(n_P) < P$. We consider the following two cases:

(a) $P \leq f^*(n)$, then trivially $f_{JPS}(n_P) < P \leq f^*(n)$.
(b) $P > f^*(n)$, then there exists a solution from n with cost lower than P. All arcs fixed in n_P must be included in such a solution, so this solution is reachable from n_P, then $f_{JPS}(n_P) \leq f^*(n)$. □

From this result, we can define a new heuristic estimation f'_{IS} as

Definition 2 For all state n

$$f'_{IS}(n) = \max\{f_{JPS}(n_P) | n_P \text{ is consistent}\}. \tag{8}$$

So, we have to consider whether or not this estimation might be better than the one given in Definition 1. Let $LI = f_{IS}(n)$, then n_{LI+1} is consistent, i.e., $f_{JPS}(n_{LI+1}) < LI + 1$.

Lemma 6 *Let P_1 and P_2 be integers such that $LI + 1 \leq P_1 < P_2$ then $f_{JPS}(n_{P_1}) \geq f_{JPS}(n_{P_2})$.*

Proof As heads and tails of operations are larger or equal in n_{P_1} than they are in n_{P_2}, then by similar reasoning as in Lemma 2, every feasible JPS for a machine in n_{P_1} is a feasible JPS for that machine in n_{P_2} too, so $f_{JPS}(n_{P_1}) \geq f_{JPS}(n_{P_2})$. □

Theorem 6 $f_{IS}(n) \geq f'_{IS}(n)$, *for all state n.*

Proof From Lemma 6 it follows that $f'_{IS}(n) = f_{JPS}(n_{LI+1}) \leq LI = f_{IS}(n)$. □

Therefore the new heuristic estimation f'_{IS} could be definitively discarded as it cannot improve f_{IS}. Yet, Lemma 6 suggests a way to speed up the dichotomic search for calculating f_{IS}. When a consistent state n_P is reached with $f_{JPS}(n_P) = X$, we can consider n_X as inconsistent without calling $IS(n, X)$, as $X < P$ and then from Lemma 6 it follows that $f_{JPS}(n_X) \geq f_{JPS}(n_P)$. Then, if X is greater than the lower limit of the dichotomic search in that iteration, this lower limit could be increased up to X. The assessment of this improvement is left over to the experimental study.

6 Computational study

As we have pointed, we have conducted an experimental study to compare the proposed partially informed depth-first search algorithm (DF) with the original Brucker's branch and bound algorithm (BB). We have considered three sets of benchmarks taken from the OR-library (Beasley 1990). Firstly, eighteen instances of size 10×10 (10 jobs and 10 machines): FT10, ABZ5-6, LA16-20 and ORB01-10. Then, we considered the set of instances selected in Applegate and Cook (1991) as very hard to solve: FT20 (size 20×5), LA21, LA24, LA25 (15×10), LA27, LA29 (20×10), LA38, LA40 (15×15), ABZ7-9, (20×15). Finally, we considered the set of Taillard's instances of size (20×15) (Taillard 1993) together with ABZ7-9. These are very large instances and the optimal solution is not known for the great majority of them. The target machine was Linux (Ubuntu 9.04) on Intel Core 2 Quad Q9400 (2.66 GHz), 4 GB RAM.

Firstly, we report some preliminary results with the purpose of analyzing the efficiency and effectiveness of h_{IS} w.r.t. that of h_{JPS}. Then we report the results of the whole experimental study across the three sets of instances considered.

6.1 Analysis of the efficiency and effectiveness of h_{IS}

As we have seen, the improved heuristic h_{IS} is largely more informed than the original one h_{JPS}, i.e., $h_{IS}(n) \geq h_{JPS}(n)$ for every state n, so it is expected that the number of nodes expanded by the partially informed depth-first search algorithm is smaller with h_{IS} than it is with h_{JPS} due to the difference in f-values. Also, a larger number of nodes are expected to

Table 1 Performance of h_{IS} in three different conditions: dichotomic search in intervals I_0 and I_1 without heuristic refinement, and dichotomic search in I_1 with heuristic refinement

Interval	Refinement	Avg. $\#P$
I_0	no	4.90
I_1	no	2.10
I_1	yes	2.10

be pruned under the condition $f(n) \geq UB$ and the evaluation function will hopefully guide the search towards more promising regions of the search space, so as reaching better upper bounds quickly. However, it is also clear that computing h_{IS} takes more time than computing h_{JPS}, so we have to consider whether or not the increase in time consumed by the heuristic is compensated by the improvement in the quality of the heuristic information and, at the end, in the reduction of the effective search space.

In this section, we provide some preliminary results with the purpose of demonstrating the differences between heuristics h_{JPS} and h_{IS} from the viewpoints of efficiency and effectiveness.

6.1.1 *Efficiency of h_{IS}*

As it can be expected, the time taken by h_{IS} might be conditioned by the average number of values P in the interval $[\max(f_{JPS}(n'), f_{IS}(n)) + 1, UB - 1]$, with $n' \in SUC(n)$, that are evaluated to calculate $f_{IS}(n')$. Each time a value P is evaluated, a call to the procedure *IS* is issued. The time consumed by these operations is similar to the time required to generate a successor and evaluate it with h_{JPS}. In order to visualize the number of such evaluations, we have conducted the following experiment: 13 large instances of size 20×15 were considered (Tai11-20, ABZ7-9). The search algorithm was run for each one of them for 1 hour. Three different runs were done for each instance considering different methods for computing f_{IS}. The first one considering the interval $I_0 = [f_{JPS}(n') + 1, UB - 1]$ to calculate $f_{IS}(n')$ and the second one considering the interval $I_1 = [\max(f_{JPS}(n'), f_{IS}(n)) + 1, UB - 1]$. This way we can evaluate the influence of monotonicity on the efficiency of f_{IS}. In the third run we have considered the same interval I_1 and the refinement suggested in Sect. 5.3 to speed up the calculation.

Table 1 reports the results averaged for all the 13 instances. For each experiment we report the number of values P evaluated in a call to f_{IS} (Avg. $\#P$). As we can observe, there is a great difference between intervals I_0 and I_1. The average number of P values that are evaluated to compute f_{IS}-values is reduced from 4.90 to 2.10 (i.e., it is decreased by about 57.1 %). This is an important improvement that can be achieved due to h_{IS} being monotonic. From the last row in Table 1 we can observe that the improvement suggested in Sect. 5.3 has no effect in these experiments.

6.1.2 *Effectiveness of h_{IS}*

In this case, our aim is to assess the difference in quality between both heuristics, h_{JPS} and h_{IS}. To do that we considered a number of instances that can be solved to optimality in order to visualize the values of both heuristics across the whole search space. The selected instances were some of those of size 10×10 that have required more time in our experiments (namely ORB01, ORB03, FT10 and LA20). For each one of them, the search algorithm was run until completion and we have analyzed the difference between the two heuristic estimations and the number of nodes visited at different levels of the search space. As the

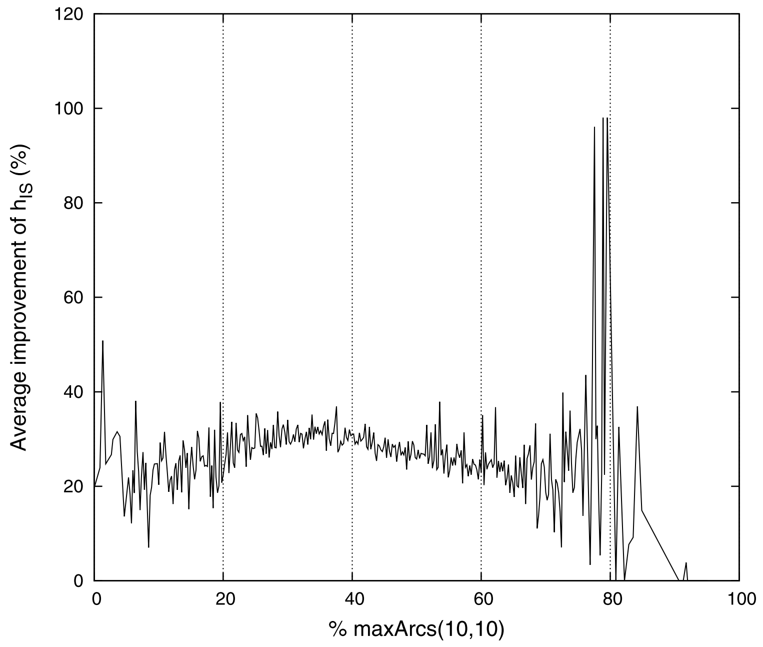


Fig. 4 Distribution of heuristic improvements in the effective search space depending on the number of arcs fixed

number of arcs fixed from a state to a successor is not constant, we have chosen to take the number of disjunctive arcs fixed in a node as its level, instead of the length or the cost of the path from the initial state to it. The initial state has no disjunctive arcs fixed whereas the maximum number of arcs that can be fixed for an instance with N jobs and M machines is given by the expression

$$\text{maxArcs}(N, M) = M \times \frac{(N-1)^2 + (N-1)}{2}. \tag{9}$$

States having such a number of disjunctive arcs fixed represent feasible schedules. However, the partially informed depth-first search algorithm rarely reaches this situation, due to the calculation of upper bounds and the condition $f(n) \geq UB$ that prunes the node n.

Figure 4 shows the results from instance ORB01 (the results from ORB03, FT10 and LA20 are fairly similar). The x-axis represents the percentage of the disjunctive arcs fixed (with respect to maxArcs(10, 10)). This value is represented by *% level* in the following. The y-axis represents the average improvement in percent of h_{IS} over h_{JPS}, computed for each node n as

$$100 \times \frac{h_{IS}(n) - h_{JPS}(n)}{h_{JPS}(n)}. \tag{10}$$

As we can observe, the average improvement is about 30 % and it is more or less uniform for different values of the number of arcs fixed in the states, with variations in only a very small fraction of the nodes at low and high levels of the search tree.

Figure 5 illustrates the distribution of the states evaluated with respect to the number of disjunctive arcs fixed. As we can observe they are normally distributed: the number of nodes evaluated at low levels is very small, then the number increases quickly for intermediate levels and finally it is very low again for levels close to the final states. This is quite reasonable, as at the end most of the nodes get pruned and at the beginning the number of states is lower than it is at intermediate levels. The results given in Figs. 4 and 5 correspond to the search space traversed by the algorithm using the heuristic h_{IS}. With h_{JPS} there are only small variations due to the differences in the number of evaluated nodes.

These results suggest the possibility of using different heuristics at different levels. In particular we propose using h_{IS} at shallow levels where there are few nodes and the decisions

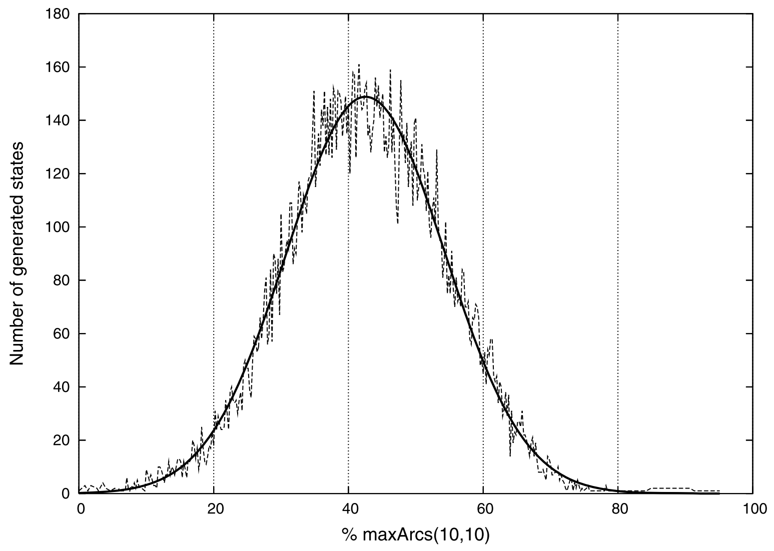


Fig. 5 Distribution of states in the effective search space depending on the number of arcs fixed

Table 2 Summary of results across instances of size 10 × 10. Columns 3 to 9 represent the number of expanded nodes and the time taken to reach the optimal solution, relative to BB

	BB	% *level* of use of h_{IS} in depth first						
		0	20	35	50	65	80	100
#Expanded nodes	100.00	69.45	67.71	67.42	68.21	68.63	68.46	68.41
Time (s)	100.00	86.41	86.41	97.09	119.42	138.83	147.57	148.54

are more critical. At intermediate levels, maybe the use of a low cost heuristic such as h_{JPS} could be better as there is a very large number of states and the decisions are less critical. And finally, at the last levels where the decisions are much less critical and few nodes are visited, the heuristic is not very relevant. So we propose using h_{IS} up to a given level of the search tree in order to take the most critical decisions and then use h_{JPS} in order to save time.

6.2 Summary of experimental results

In this section, we summarize the results of the whole experimental study. For the three sets of instances we have applied h_{IS} up to different levels of the search: 0, 20, 35, 50, 65, 80 and 100 %. With level 0 % the improved heuristic is not applied to any of the nodes. For deeper levels, we use the estimation $f(n) = \max(f(p), f_{JPS}(n))$, where p is the parent of state n. In all cases we report values averaged for all the instances of the set and normalized so as BB is given the value 100.

Table 2 reports the results from the 10 × 10 instances. As all of these instances are easily solved by both algorithms, we report the average number of nodes expanded and the time taken to reach an optimal schedule (and prove its optimality) in percentage with respect to those obtained by BB. The average time taken by the BB algorithm is 5.72 seconds and the average number of expanded nodes is 7184.67. The first remarkable result is that DF reduces the number of expanded nodes by about 30 % with respect to BB. This number is almost the same disregarding the level of application of the improved heuristic (even if only h_{JPS} is used). This is a little bit surprising as the differences between the two heuristic estimations are about 30 % in average, as we have seen in the previous section. In our opinion this is due to the fact that these instances are easy to solve and the single heuristic h_{JPS} is able to guide the search quite well. At the same time, the intensive use of h_{IS} only contributes to increase

Table 3 Summary of result across the Selected instances. Errors in percent of the upper bounds and the lower bounds w.r.t. the best known values. The time limit is 3600 s

	BB	% *level* of use of h_{IS} in depth first						
		0	20	35	50	65	80	100
E_{UB}	100.00	81.17	66.50	60.76	51.94	49.40	52.14	52.39
E_{LB}	100.00	100.00	45.97	65.74	60.63	65.74	65.74	65.74

Table 4 Summary of results across instances of size 20 × 15. Errors in percent of the upper bounds and the lower bounds w.r.t. the best known values. The time limit is 3600 s

	BB	% *level* of use of h_{IS} in depth first						
		0	20	35	50	65	80	100
E_{UB}	100.00	74.19	69.11	70.06	69.94	62.75	58.78	58.34
E_{LB}	100.00	100.00	71.21	71.21	71.21	71.21	71.21	71.21

the time taken so as the overall time is even greater than the time taken by BB when h_{IS} is applied at a level beyond 35 %. For these instances, applying h_{IS} at a level in [0, 20] is the best choice and, in this case, the time is reduced by about 15 % with respect to BB.

Instances other than 10×10 are much harder to solve and many of them are not solved by any of the algorithms. So, for these instances we have made two series of experiments giving the algorithms different time limits, one hour and five hours respectively. In this experiments we consider the quality of the upper and lower bounds reached by BB in average with respect to the best known lower and upper bounds, taken from Zhang et al. (2008). Then we report the average error in the upper and lower bounds (E_{UB} and E_{LB} respectively) reached by DF with respect to those reached by BB. The lower bound returned by BB is the minimum f-value from the set of nodes in the stack with at least one before-candidate or one after-candidate not yet explored. This is a small refinement on the original implementation of the Brucker's algorithm that returns $f(\mathit{initial})$.

Table 3 shows the results across the second set of instances with a time limit of one hour. The average error reached by BB is 5.37 % for upper bounds and 3.42 % for lower bounds w.r.t. the best known bounds. As we can observe, DF is better than BB in all cases. When h_{IS} is used, the improvement in lower bounds is about 35 % in all cases, independently of the level up to h_{IS} is applied, with exception of the level 20 %. This is due to the fact that the solution to instance *LA*38 is proved to be optimal in this case and so the lower bound is better than in the remaining ones. The fact that the lower bound is almost the same for all levels is quite reasonable due to the depth-first search. The lower bound is the lowest f-value of a node in the list OPEN and, as the instances are very large, this value might correspond to a successor of the initial state due to the fact that the algorithm is not able to expand all of these successors after one hour. For the same reason the lower bound at level 0 % is the same as that of BB. However, the quality of the upper bounds improves in direct ratio with the level up to h_{IS} is applied. In this case it is worth to exploit the improved heuristic beyond level 50 %. The improvement with respect to BB is about 50 %. Moreover, BB does not reach an optimal solution to the instance FT20, whereas DF solves it in just a few seconds using h_{IS}.

Table 4 summarizes the results from the largest instances (size 20 × 15) obtained in one hour. These are extremely hard instances. The average error reached by BB is 13.06 % for

Table 5 Summary of result across the Selected instances. Errors in percent of the upper bounds and the lower bounds w.r.t. the best known values. The time limit is 18000 s

	BB	% *level* of use of h_{IS} in depth first						
		0	20	35	50	65	80	100
E_{UB}	100.00	78.89	66.87	54.03	49.74	45.91	48.20	48.78
E_{LB}	100.00	122.94	86.55	86.55	86.55	86.55	86.55	86.55

Table 6 Summary of results across instances of size 20 × 15. Errors in percent of the upper bounds and the lower bounds w.r.t. the best known values. The time limit is 18000 s

	BB	% *level* of use of h_{IS} in depth first						
		0	20	35	50	65	80	100
E_{UB}	100.00	76.95	68.23	72.02	68.48	63.07	61.75	61.95
E_{LB}	100.00	100.00	71.21	71.21	71.21	71.21	71.21	71.21

upper bounds and 4.23 % for lower bounds with respect to the best known bounds. The results show similar tendency as those for the second set of instances. DF is always better than BB. In this case, the lower bounds reached by DF when h_{IS} is used at any level are always the same due to the fact that these instances are even harder to solve. The upper bounds improve in direct ratio with the level up to h_{IS} is used. In this case, it is worth to exploit the improved heuristic in the whole search space. Overall, the improvement in upper bounds quality with respect to BB is more than 40 %.

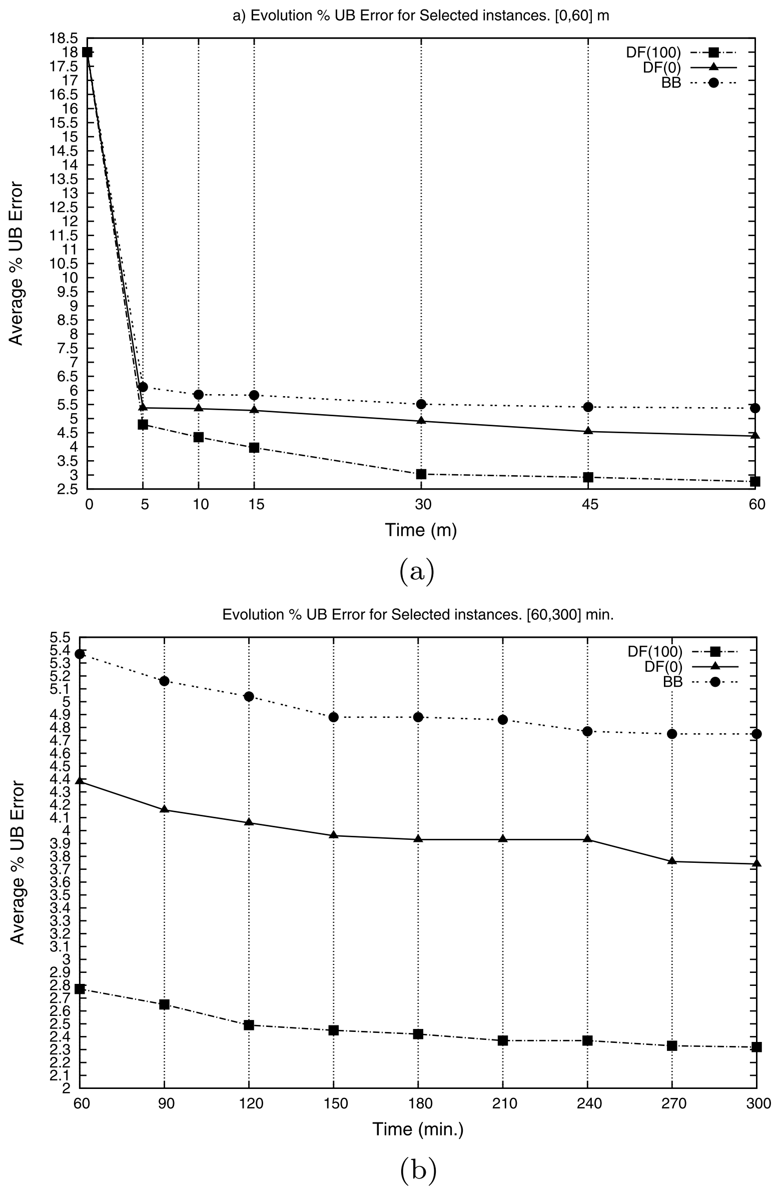
Let us now consider the results after five hours. Tables 5 and 6 summarize these results from the Selected and the 20 × 15 instances respectively. For the Selected instances, BB reaches an average error of 4.75 for upper bounds and 1.69 for lower bounds. For 20 × 15 instances these errors are 11.63 and 4.18 respectively.

As we can see, in all cases the best choice is exploiting h_{IS} across the whole search tree. If we compare these results with those given in Tables 3 and 4 we can make the following observations. For the Selected instances, the relative error of the upper bounds is now 48.78 %, while it was 52.39 % in a time limit of one hour. However, the error in the lower bounds gets worse: 86.55 %, versus 65.74 %. The reason for this has to do with the number of instances each algorithm can solve to optimality, as in these cases the error is null. When the time limit is one hour, both algorithms solve the instances *LA*24 and *LA*25, and DF also solves the instance *FT*20. While in five hours the instances *LA*21 and *LA*38 get solved by the two algorithms as well, and the instance *LA*40 is solved by BB. For the 20 × 15 instances, the error in the upper bound gets a little bit worse, 61.95 % versus 58.34 %, but it is the same for the lower bound. In this case, all the instances remain unsolved by both algorithms and due to the size of the search space, DF is not able to return a lower bound better than the heuristic estimation for the initial state.

Finally, Figs. 6 and 7 show the evolution of the upper bounds along the five hours for the Selected and 20 × 15 instances respectively. In both cases three algorithms were considered: BB, DF with heuristic h_{JPS} (DF(0)) and DF with heuristic h_{IS} (DF(100)). In this case, the values reported are the errors in percentage of the upper bound w.r.t. the best known solution. Figures 6(a) and 7(a) show the evolution along the first hour and Figs. 6(a) and 7(b) shows the evolution during the remaining four hours with a different scaling.

For the 20 × 15 instances, the final values are around 11.68, 8.99 and 7.23 for algorithms BB, DF with h_{JPS} and DF with h_{IS} respectively. So, the relative improvement of DF w.r.t. BB

Fig. 6 Evolution % UB Error for Selected instances. Time interval [0, 300] minutes

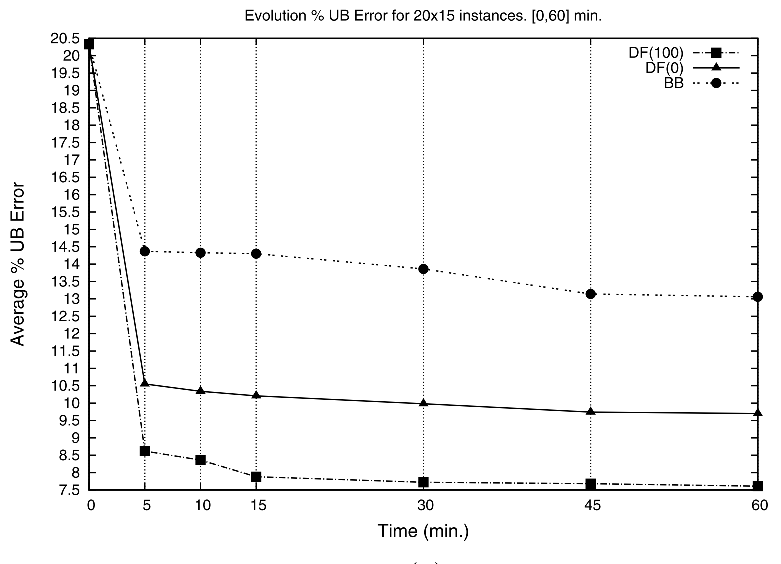


when both of them exploit h_{JPS} is 23.03 %, and the relative improvement of h_{IS} w.r.t. h_{JPS} when both of them are used in combination with DF is 19.58 %. The improvement of DF with h_{IS} w.r.t. BB is 38.1 %. Moreover, in Fig. 7 we can observe that the value reached by BB after five hours is reached by DF with h_{IS} in less than four minutes. Similar comments can be made regarding the results from the Selected instances. So, these results make it clear that DF with h_{IS} outperforms BB and that the superiority of DF over BB is even more remarkable for very large instances.

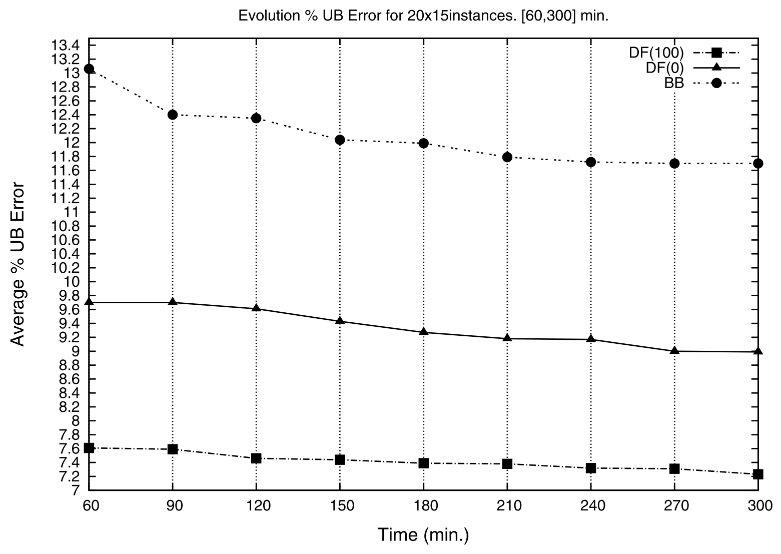
We have made some experiments, not reported here, combining BB with the improved heuristic and the results were not good. In this case BB takes much more time to reach the same solutions, or reaches worse solutions in a given time. The reason for this is that the capability for pruning states from the condition $f(n) \geq UB$ is quite similar with both heuristics as every state n undergoes immediate selection, while h_{IS} is more time consuming than h_{JPS}. However, when combined with DF, h_{IS} is able to guide the search so that better solutions are reached earlier and so the effective search space is reduced.

So, we can draw two main conclusions from this experimental study. The first one is that DF is better than BB when both of them use the same heuristic information given by h_{JPS} and the second one is that DF is able to improve when it is given more informed and time

Fig. 7 Evolution % UB Error for 20×15 instances. Time interval $[0, 300]$ minutes



(a)



(b)

consuming heuristic, such as h_{IS}, but in this case we have to be aware of the problem size and exploit this heuristic up to a level that depends on the problem size. In any case, for sufficiently large instances it is worth to exploit this heuristic along the whole search.

6.3 Detailed results

In this section we report detailed results from BB, DF using h_{JPS} (DF(0)) and DF using h_{IS} (DF(100)). Table 7 shows the results obtained for the instances with size 10×10 and Table 8 reports the results for the largest instances, those in the Selected and Taillard's sets. Table 7 shows the number of expanded nodes (#Expanded) and the time taken (Time (s)) to certify the optimal solution for all 10×10 instances which are summarized in Table 3. All three algorithms can solve these instances in a short time and there are only some differences in the number of expanded states, in favor of DF(100), and in the time taken, in favor of BB and DF(0). So, we can consider that the three algorithms are actually similar in solving instances of this size.

The situation is quite different for the instances in the Selected and Taillard's sets which are much harder to solve and most of them can be solved by none of the algorithms by

Table 7 Summary of results from 10 × 10 instances

Instance	BB		DF(0)		DF(100)	
	#Expanded	Time (s)	#Expanded	Time (s)	#Expanded	Time (s)
ORB01	25281	20	25474	25	19604	34
ORB02	2315	2	2005	2	1640	3
ORB03	59234	48	24382	26	35570	61
ORB04	7830	6	7665	7	9751	17
ORB05	3131	2	2433	2	1908	3
ORB06	12288	10	6511	7	4484	8
ORB07	1953	1	808	1	1121	2
ORB08	5418	4	4109	3	2211	3
ORB09	1085	1	987	1	470	1
ORB10	843	1	2163	2	976	2
LA16	252	0	335	1	169	0
LA17	63	0	85	0	58	0
LA18	271	0	547	0	353	1
LA19	1456	1	2727	3	2441	4
LA20	1381	1	2045	2	1315	3
ABZ5	2146	2	1035	1	993	2
ABZ6	135	0	205	0	183	0
FT10	4242	4	6301	6	5225	9

5 hours. Table 8 shows the errors in percentage w.r.t. the best known lower and upper bounds (E_{LB} and E_{UB}) for these large instances. From these results we can appreciate differences between the three methods showing that DF outperforms BB. In this case to enhance the conclusions we have done some statistical tests to analyze the differences between the algorithms. Following Garcia et al. (2010), as we have multiple-problem analysis, we have used non-parametric statistical tests, in fact we have used paired Wilcoxon tests which show statistically significant differences in favor of DF.

Table 9 reports the results from E_{UB} analysis. For the experiments at 1 hour, both versions of DF perform better than BB, being the difference more significant for DF(100) as it is indicated by p-value = 0.0033 when it is compared to BB and p-value = 0.006659 when it is compared to DF(0). The statistical tests draw similar results for the comparison of the algorithms at 5 hours, with only one exception for DF(0) and BB for which a significatively difference cannot be established (p-value = 0.06355). Regarding the comparison of the same algorithm at 1 and 5 hours, there are significant differences in all three cases. Here it is remarkable that the largest differences are in both versions of DF (p-values 0.0008281 and 0.0005454 respectively, versus p-value 0.001929 for BB) showing that it is DF the algorithm that has more capability to improve when it is given more time. Also, it is remarkable the difference between DF(100) at 1 hour and BB and DF(0) at 5 hours with p-values 0.03056 and 0.03065 respectively, showing that DF(100) in 1 hour outperforms both BB and DF(0) in 5 hours.

Regarding the results for E_{LB} reported in Table 8, we can observe that BB and DF(0) at 1 and 5 hours are practically equivalent. In this case the Wilcoxon paired tests draw the results given in Table 10, showing statistical differences between BB and DF(100) at 1 and 5 hours with p-values 0.0001605 and 0.0003633 respectively.

Table 8 Summary of results from Selected and Taillard's instances

Instance	Size		BK	1 hour (*Errors*)			5 hour (*Errors*)		
				BB	DF(0)	DF(100)	BB	DF(0)	DF(100)
FT20	20 × 5	UB	1165	1.2	1.9	0.0	1.2	1.9	0.0
		LB	1165	0.1	0.1	0.0	0.1	0.1	0.0
LA21	15 × 10	UB	1046	0.8	1.1	1.3	0.0	0.0	0.0
		LB	1046	4.9	4.9	1.2	0.0	0.0	0.0
LA24	15 × 10	UB	935	0.0	0.0	0.0	0.0	0.0	0.0
		LB	935	0.0	0.0	0.0	0.0	0.0	0.0
LA25	15 × 10	UB	977	0.0	0.0	0.0	0.0	0.0	0.0
		LB	977	0.0	0.0	0.0	0.0	0.0	0.0
LA27	20 × 10	UB	1235	2.8	5.9	2.6	2.8	5.7	2.4
		LB	1235	0.0	0.0	0.0	0.0	0.0	0.0
LA29	20 × 10	UB	1152	4.3	8.1	7.5	4.3	8.1	7.1
		LB	1152	3.3	3.3	2.9	3.3	3.3	2.9
LA38	15 × 15	UB	1196	2.7	2.3	1.1	0.0	0.0	0.0
		LB	1196	9.9	9.9	7.4	0.0	0.0	0.0
LA40	15 × 15	UB	1222	0.3	1.5	2.1	0.0	0.7	0.3
		LB	1222	4.3	4.3	2.5	0.0	4.3	2.5
ABZ7	20 × 15	UB	656	10.7	6.6	5.0	7.9	5.6	4.6
		LB	656	0.9	0.9	0.8	0.9	0.9	0.8
ABZ8	20 × 15	UB	665	15.3	9.2	5.7	15.3	8.3	5.6
		LB	645	7.4	7.4	5.7	7.4	7.4	5.7
ABZ9	20 × 15	UB	678	20.9	11.5	5.6	20.6	10.9	5.5
		LB	661	6.8	6.8	4.2	6.8	6.8	4.2
tai11	20 × 15	UB	1357	9.3	7.6	13.1	8.8	7.1	13.1
		LB	1323	5.2	5.2	4.1	5.2	5.2	4.1
tai12	20 × 15	UB	1367	13.7	10.4	5.9	12.0	7.6	5.4
		LB	1351	5.0	5.0	3.0	5.0	5.0	3.0
tai13	20 × 15	UB	1342	13.0	11.4	8.0	8.0	9.9	7.2
		LB	1282	3.0	3.0	3.0	3.0	3.0	3.0
tai14	20 × 15	UB	1345	8.0	2.8	3.2	7.6	2.8	3.2
		LB	1345	0.3	0.3	0.0	0.3	0.3	0.0
tai15	20 × 15	UB	1339	16.5	12.2	9.3	16.5	12.2	9.1
		LB	1304	5.6	5.6	4.4	5.6	5.6	4.4
tai16	20 × 15	UB	1360	13.4	11.0	9.0	11.8	11.0	8.1
		LB	1304	5.1	5.1	4.4	5.1	5.1	4.4
tai17	20 × 15	UB	1462	9.8	9.6	5.1	9.8	9.6	5.1
		LB	1462	2.0	2.0	1.4	2.0	2.0	1.4
tai18	20 × 15	UB	1396	9.2	10.0	10.5	9.2	9.2	10.5
		LB	1369	3.9	3.9	0.8	3.9	3.9	0.8
tai19	20 × 15	UB	1332	14.9	13.3	7.7	9.0	12.3	7.6
		LB	1304	6.7	6.7	5.1	6.7	6.7	5.1
tai20	20 × 15	UB	1348	15.0	10.6	10.8	15.0	10.2	9.2
		LB	1318	3.0	3.0	2.2	3.0	3.0	2.2

Table 9 Paired Wilcoxon tests to analyze E_{UB} from BB, DF(0) and DF(100) on the Selected and Taillard's instances. The alternative hypothesis is "The values from Algorithm 1 are greater than those from Algorithm 2". Times are given in hours

Algorithm 1	Time 1	Algorithm 2	Time 2	p-value
BB	1	DF(0)	1	0.01565
BB	1	DF(100)	1	0.0033
DF(0)	1	DF(100)	1	0.006659
BB	5	DF(0)	5	0.06355
BB	5	DF(100)	5	0.006055
DF(0)	5	DF(100)	5	0.007855
BB	1	BB	5	0.001929
DF(0)	1	DF(0)	5	0.0008281
DF(100)	1	DF(100)	5	0.0005454
BB	5	DF(100)	1	0.03056
DF(0)	5	DF(100)	1	0.03065

Table 10 Paired Wilcoxon tests to analyze E_{LB} from BB, DF(0) and DF(100) on the Selected and Taillard's instances

Algorithm 1	Time 1	Algorithm 2	Time 2	p-value
BB	1	DF(100)	1	0.0001605
BB	5	DF(100)	5	0.0003633

Therefore, the statistical tests confirm that DF(100) is the best of the three algorithms in reaching both lower and upper bounds for large instances.

7 Conclusions and future work

In this paper, we have built an exact partially-informed depth-first search algorithm for the job shop scheduling problem, which is a variant of the well-known Brucker's backtracking algorithm, and designed and formalized a new heuristic estimation for these algorithms. Then, the algorithms were compared by means of an experimental study across a set of conventional instances of medium, large and very large sizes. The results of this study show that the proposed algorithm combined with the new heuristic outperforms the original Brucker's, as it reaches much better solutions and lower bounds for the largest instances of the set when both algorithms are given the same time.

From this study we have seen that partially informed depth-first search may be more efficient than chronological backtracking in solving a complex problem such as the job shop scheduling problem. Even though the latter requires a negligible amount of memory, the former can take much more profit from the heuristic information while it keeps the use of memory at very low level. So, in spite of that chronological backtracking can expand a larger number of states than the partially informed depth-first counterpart in a given time, it is the last one that guides the search towards more promising areas of the search space, so that the algorithm is able to find out better and better solutions more quickly.

We have also observed that the improvement is in direct ratio with the information degree of the heuristic estimation, even though this degree is also in direct ratio with the time required for calculating the heuristic on the search states.

The results of our work suggest us that similar techniques could be used considering other search spaces and other problems, in particular if powerful constraint propagation rules and greedy algorithms were available. So, we propose to exploit other constraint propagation rules such as those proposed in Dorndorf et al. (2000) to devise new heuristics for the job shop scheduling problem. We also conjecture that any backtracking algorithm may be outperformed by a partially informed depth-first search algorithm in the same way as it was Brucker's. So, for example, it would be interesting to exploit the edge-guessing procedure, proposed in Dorndorf et al. (2002), in combination with a partially informed depth-first search.

As partially informed depth-first is just one of the techniques that allow search algorithms to maintain low memory requirement, we propose to experiment with other search strategies that were devised with this purpose, such as Iterative Deepening A* (IDA*) (Korf 1985), or Limited Discrepancy Search (Harvey and Ginsberg 1995). Furthermore, there are some recent techniques such as Beam Search (Pinedo 2008) or Nested Partitions (Wu et al. 2010, 2011) which can exploit knowledge better than single branch and bound to focus on the most promising branches of the search tree. These techniques have been applied to some scheduling problems, for example in Yau and Shi (2009) the authors applied nested partitions to a scheduling problem considering the bills of material and the working shifts. So, it would be interesting to compare these techniques with branch and bound and partially informed depth first search on the job shop scheduling problem.

Appendix A: Details of the expansion mechanism

Let us consider a feasible schedule S being compatible with the disjunctive arcs fixed in state n. Of course, S might be a schedule calculated by Algorithm 2. The solution graph G_S has a critical path with critical blocks $B_1, \dots, B_k$. For block $B_j = (u_1^j, \dots, u_{m_j}^j)$ the sets of operations

$$E_j^B = B_j \setminus \{u_1^j\} \quad \text{and} \quad E_j^A = B_j \setminus \{u_{m_j}^j\}$$

are called *before-candidates* and *after-candidates* respectively. For each before-candidate (after-candidate) a successor s of n is generated by moving the candidate before (after) its corresponding block. An operation $l \in E_j^B$ is moved before B_j by fixing the arcs $\{l \to i;\ i \in B_j \setminus \{l\}\}$. Similarly, $l \in E_j^A$ is moved after B_j by fixing the arcs $\{i \to l;\ i \in B_j \setminus \{l\}\}$.

This expansion strategy is complete as it guarantees that at least one optimal solution is contained in the search graph. However, it can be improved by fixing additional arcs in order to define a complete search tree. Let us consider a permutation $(E_1, \dots, E_{2k})$ of all the sets E_j^B and E_j^A. This permutation defines an ordering for successors generation. When a successor is created from a candidate E_t, we can assume that all the solutions reachable from n by fixing the arcs corresponding to the candidates $E_1, \dots, E_{t-1}$ will be explored from the successors associated to these candidates. So, for the successor state s generated from E_t the following sets of disjunctive arcs can be fixed: $F_j = \{u_1^j \to i;\ i = u_2^j, \dots, u_{m_j}^j\}$, for each $E_j^B < E_t$ and $L_j = \{i \to u_{m_j}^j;\ i = u_1^j, \dots, u_{m_j-1}^j\}$, for each $E_j^A < E_t$ in the permutation above. So the successors of a search tree node n generated from the permutation $(E_1, \dots, E_{2k})$ are defined as follows. For each operation $l \in E_j^B$, $j = 1 \dots k$, generate a candidate successor s by fixing the arcs $FD_s = FD_n \cup S_l^B$, where

$$S_l^B = \bigcup_{E_i^B < E_j^B} F_i \cup \bigcup_{E_i^A < E_j^B} L_i \cup \{l \to i : i \in B_j \backslash \{l\}\}. \tag{11}$$

And for each operation $l \in E_j^A$ generate a search tree node s by fixing the arcs $FD_s = FD_n \cup S_l^A$ with

$$S_l^A = \bigcup_{E_i^B < E_j^A} F_i \cup \bigcup_{E_i^A < E_j^A} L_i \cup \{i \to l : i \in B_j \backslash \{l\}\}. \tag{12}$$

After fixing one of these sets of arcs to build a new successor n' from a state n, heads and tails are calculated by the algorithm *Computing new Heads and Tails* as it is indicated in Sect. 4.1. To compute heads, the algorithm is applied from the *start* node following a topological ordering in $G_{n'}$ (analogously for tails starting from node *end* backwards). If heads and tails get fixed for all operations, then n' is a feasible search state, otherwise $G_{n'}$ contains some cycle and so n' is unfeasible.

For example, if we consider the search state of Fig. 2(a) the heuristic solution might be that of Fig. 1(b). Then, the first successor state would be that represented in Fig. 2(b).

Appendix B: Improving heads and tails

The algorithms to improve heads and tails proposed in Brucker et al. (1994a) are based on the following ideas. If (J, c) is primal pair, the operation c cannot start at a time lower than

$$r_J = \max_{J' \subseteq J} \left\{ \min_{j \in J'} r_j + \sum_{j \in J'} p_j \right\}. \tag{13}$$

So, if $r_c < r_J$, we can set $r_c = r_J$ and then the procedure *Select* fixes all the primal arcs $\{j \to c;\ j \in J\}$. Analogously, if (c, J) is dual pair, q_c cannot be lower than

$$q_J = \max_{J' \subseteq J} \left\{ \sum_{j \in J'} p_j + \min_{j \in J'} q_j \right\}. \tag{14}$$

So, if $q_c < q_J$, we can set $q_c = q_J$ and then the procedure *Select* fixes all the dual arcs $\{c \to j;\ j \in J\}$.

In Brucker et al. (1994a) and Brucker (2004) an algorithm is given that computes all primal and dual arcs in polynomial time $(O(|I|^2))$. This algorithm is based on other two algorithms for improving heads and tails from the current upper bound *UB* and then fixing additional arcs with the procedure *Select*.

To improve the head of an operation c, the idea is to compute a lower bound, s_c, of the completion time of c in any solution that improves *UB*, under the assumption that preemption is allowed. Clearly, s_c is also a lower bound for the completion of c in a non preemptive schedule, so r_c may be improved to $r_c' = s_c - p_c$. The algorithm for improving the head of an operation c is given in Algorithm 5. Analogously for improving tails.

Figure 8 shows the schedule built by the Algorithm 5 to improve the head of operation $c = 4$ with $UB = 52$ for the same instance as in Fig. 3. As we can observe, the lower bound obtained for the completion time of operation 4 is $s_4 = 39$, so $r_4' = s_4 - p_4 = 39 - 5 = 34$. Therefore, after improving heads, the arc $(5 \to 4)$ gets fixed by the procedure *Select*, due to

Algorithm 5 Improving Heads (upper bound *UB*)

```
1: Calculate $JPS$ up to $r_c$. Let $p_k^+$ the remaining time of operation $k$;
2: From $UB - 1$ backwards, schedule all operations with $p_k^+ > 0$, excluding $c$, sorted by
   non decreasing tails, at the largest starting time $t_k$ such that $t_k + p_k^+ + q_k \leq UB - 1$;
3: Schedule operation $c$ from $r_c$ forward using the earliest idle periods. Let $s_c$ the comple-
   tion time of $c$;
4: If $s_c - p_c > r_c$ then $r_c = s_c - p_c$;
```

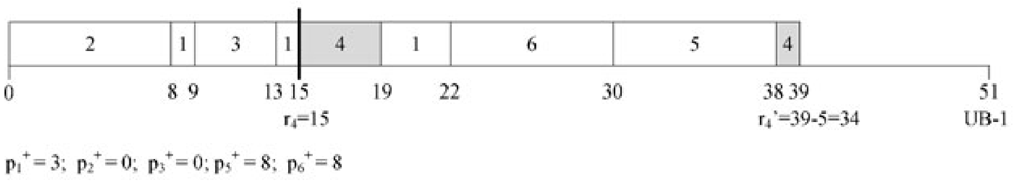


Fig. 8 Calculation of earliest completion times s_c to $c = 4$

$r'_4 + p_4 + p_5 + q_5 = 60 \geq 52$. However, before improving heads that arc would not be fixed as $r_4 + p_4 + p_5 + q_5 = 42 < 52$.

In (Brucker 2004) the following results on Algorithm *Improving Heads* are given.

Theorem 7

(i) *The schedule S calculated by Algorithm Improving Heads is a feasible preemptive schedule.*

(ii) *Algorithm Improving Heads provides a lower bound for the completion time of operation c in a preemptive schedule under the assumption that preemption at integer times is allowed.*

Theorem 8 *Procedure Select calculates all primal and dual arcs associated with I if we replace the original r_j-values and q_j-values by the modified values obtained with the algorithms for improving heads and improving tails respectively.*

Appendix C: Proof of Lemma 1

Lemma 1 *Let n_1 and n_2 be two states fulfilling conditions* (6) *and* (7). *If the procedure Improving Heads is applied to n_1 and n_2 with tentative upper bounds P_1 and $P_2 \leq P_1$ respectively, and then new arcs are fixed by the procedure Select, previous conditions* (6) *and* (7) *hold.*

Proof Let $S(n_1, P_1, r_c(n_1))$ be the preemptive schedule built by the algorithm *Improving Heads* when applied to the set of operations I that require the same machine as operation c, considering heads and tails in state n_1 and tentative upper bound P_1 (analogously $S(n_2, P_2, r_c(n_2))$). And let s_c^{11} and s_c^{22} be the completion times of operation c in $S(n_1, P_1, r_c(n_1))$ and $S(n_2, P_2, r_c(n_2))$ respectively.

First of all, we have to clarify that if $S(n_2, P_2, r_c(n_2))$ is feasible, then $S(n_1, P_1, r_c(n_1))$ is feasible as well. This is true as condition (6) guarantees that any feasible preemptive schedule for the operations in I with heads and tails as they are in n_2 and tentative upper bound P_2 is a feasible preemptive schedule for the set I with heads and tails as they are in n_1 and tentative upper bound P_1. If $S(n_2, P_2, r_c(n_2))$ is not feasible, then $s_c^{22} = \infty$.

From now on, we consider that both $S(n_1, P_1, r_c(n_1))$ and $S(n_2, P_2, r_c(n_2))$ are feasible. In order to prove that $s_c^{11} \leq s_c^{22}$ we can do the following reasoning. Let us consider the preemptive schedule $S(n_2, P_1, r_c(n_1))$, i.e., a schedule built by Algorithm *Immediate Selection* with heads and tails as they are in n_2 and P_1, but considering the head of operation c as it is in n_1. Let s_c^{21} be the completion time of operation c in $S(n_2, P_1, r_c(n_1))$.

On one hand, due to $r_i(n_1) \leq r_i(n_2)$ and $q_i(n_1) \leq q_i(n_2)$ it is clear that $S(n_2, P_1, r_c(n_1))$ is a feasible preemptive schedule for the set I considering P_1 and the heads and tails as they are in n_1. So, from Theorem 7 it follows that $s_c^{11} \leq s_c^{21}$ due to the fact that s_c^{11} is a lower bound for the completion time of operation c in a preemptive schedule for I in that context.

At the same time, if we compare the preemptive schedules $S(n_2, P_1, r_c(n_1))$ and $S(n_2, P_2, r_c(n_2))$ we can observe that they are the same up to the time $r_c(n_1)$. So, the remaining time units for all the operations after $r_c(n_1)$ are the same in both schedules, even though each one of these unit intervals is scheduled at a different time in each preemptive schedule.

Now, in order to prove that $s_c^{21} \leq s_c^{22}$ we can use the following arguments: let x be a unit time interval of an operation u scheduled after $r_c(n_1)$, and tx_1 and tx_2 the times at which this unit interval is scheduled in $S(n_2, P_1, r_c(n_1))$ and $S(n_2, P_2, r_c(n_2))$ respectively. It is clear that $tx_2 \leq tx_1$ if u is an operation other than c, but $tx_2 \geq tx_1$ if u is c. In particular, this holds for the last unit interval of c, so $s_c^{21} \leq s_c^{22}$.

Then $s_c^{11} \leq s_c^{22}$ and $r'_c(n_1) = s_c^{11} - p_c \leq s_c^{22} - p_c = r'_c(n_2)$. So, after replacing all heads by the improved values we have $r_i(n_1) \leq r_i(n_2)$ for all i. As tails remain unchanged, condition (6) holds after the procedure *Improving Heads*. Condition (7) trivially holds as none of the graphs is modified by this procedure.

Now it is easy to see that both conditions hold after Procedure *Select*. Let $(i \rightarrow j)$ be a disjunctive arc fixed in n_1, then $r_j(n_1) + p_j + p_i + q_i(n_1) \geq P_1$ and trivially $r_j(n_2) + p_j + p_i + q_i(n_2) \geq P_2$ due to $r_j(n_1) \leq r_j(n_2)$, $q_i(n_1) \leq q_i(n_2)$ and $P_2 \leq P_1$, so the arc $(i \rightarrow j)$ is fixed in n_2 as well. □

References

Agnetis, A., Flamini, M., Nicosia, G., & Pacifici, A. (2011). A job-shop problem with one additional resource type. *Journal of Scheduling*, *14*(3), 225–237.

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, *3*, 149–156.

Artigues, C., & Feillet, D. (2008). A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, *159*(1), 135–159.

Balas, E., Simonetti, N., & Vazacopoulos, A. (2008). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, *11*, 253–262.

Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling*. Norwell: Kluwer Academic.

Beasley, J. E. (1990). Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, *41*(11), 1069–1072. http://www.jstor.org/stable/2582903.

Beck, J. C. (2007). Solution-guided multi-point constructive search for job shop scheduling. *The Journal of Artificial Intelligence Research*, *29*, 49–77.

Beck, J. C., & Fox, M. S. (2000). Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, *117*(1), 31–81.

Blazewicz, J., Domschke, W., & Pesch, E. (1996). The job shop scheduling problem: conventional and new solution techniques. *European Journal of Operational Research*, *93*(1), 1–33.

Blazewicz, J., Pesch, E., & Sterna, M. (2000). The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, *127*(2), 317–331.

Brinkkötter, W., & Brucker, P. (2001). Solving open benchmark instances for the job-shop problem by parallel heads and tails adjustments. *Journal of Scheduling*, *4*(1), 53–64.

Brucker, P. (2004). *Scheduling algorithms* (4th ed.). Berlin: Springer.

Brucker, P., & Thiele, O. (1996). A branch and bound method for the general-job shop problem with sequence-dependent setup times. *OR Spektrum*, *18*, 145–161.

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

Brucker, P., Jurisch, B., & Sievers, B. (1994b). Source code of a branch & bound algorithm for the job shop scheduling problem. ftp://www.mathematik.uni-kl.de/pub/Math/ORSEP/BRUCKER2.ZIP.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, *11*, 42–47.

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, *35*(2), 164–176.

Carlier, J., & Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, *26*, 269–287.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, *41*, 231–252.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000). Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, *122*, 189–240.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2002). Constraint propagation and problem decomposition: a preprocessing procedure for the job shop problem. *Annals of Operations Research*, *115*, 125–145.

Garcia, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. *Information Sciences*, *180*, 2044–2064.

Garey, M., & Johnson, D. (1979). *Computers and intractability*. New York: Freeman.

Gharbi, A., & Labidi, M. (2010). Extending the single machine-based relaxation scheme for the job shop scheduling problem. *Electronic Notes in Discrete Mathematics*, *36*, 1057–1064.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, *8*, 487–503.

González, M. A., Vela, C. R., González-Rodríguez, I., & Varela, R. (2012). Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-011-0622-5.

González-Rodríguez, I., Puente, J., Vela, C. R., & Varela, R. (2008). Semantics of schedules for the fuzzy job shop problem. *IEEE Transactions on Systems, Man and Cybernetics. Part A. Systems and Humans*, *38*(3), 655–666.

González-Rodríguez, I., Vela, C. R., Puente, J., & Hernández-Arauzo, A. (2009). Improved local search for job shop scheduling with uncertain durations. In *Proceedings of ICAPS 2009*.

Graham, R., Lawler, E., Lenstra, J., & Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Guyon, O., Lemaire, P., Pinson, E., & Rivreau, D. (2012). Solving an integrated job-shop problem with human resource constraints. *Annals of Operations Research*. doi:10.1007/s10479-012-1132-3. 1–25.

Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Harvey, W. D., & Ginsberg, M. L. (1995). Limited discrepancy search. In *Proceedings of IJCAI 1995* (Vol. 1, pp. 607–615).

Korf, R. E. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, *27*, 97–109.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artificial Intelligence*, *143*(2), 151–188.

Martin, P., & Shmoys, D. B. (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of IPCO 1996* (pp. 389–403).

Meeran, S., & Morshed, M. S. (2011). A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-011-0520-x.

Mencía, C., Sierra, M. R., & Varela, R. (2010). Partially informed depth-first search for the job shop problem. In *Proceedings of ICAPS 2010* (pp. 113–120).

Mencía, R., Sierra, M. R., Mencía, C., & Varela, R. (2011). Genetic algorithm for job-shop scheduling with operators. In *LNCS: Vol. 6687. New challenges on bioinspired applications* (pp. 305–314).

Meseguer, P. (2012). Towards 40 years of constraint reasoning. *Progress in Artificial Intelligence*, *1*(1), 25–43. doi:10.1007/s13748-011-0006-2. url:http://dx.doi.org/10.1007/s13748-011-0006-2.

Nilsson, N. (1980). *Principles of artificial intelligence*. Palo Alto: Tioga.

Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, *8*(2), 145–159.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Reading: Addison-Wesley.

Petrovic, S., Fayad, C., Petrovic, D., Burke, E., & Kendall, G. (2008). Fuzzy job shop scheduling with lot-sizing. *Annals of Operations Research*, *159*, 275–292.

Pinedo, M. (2008). *Scheduling: theory, algorithms and systems* (3rd ed.). Berlin: Springer.

Roy, B., & Sussman, B. (1964). *Les problemes d'ordonnancements avec contraintes disjonctives* (Tech. Rep.). Notes DS No. 9 bis, SEMA, Paris.

Sierra, M. R., Mencía, C., & Varela, R. (2011). Optimally scheduling a job-shop with operators and total flow time minimization. In *LNCS: Vol. 7023. Advances in artificial intelligence* (pp. 193–202).

Smith, S. F., & Cheng, C. C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of AAAI 1993* (pp. 139–144).

Streeter, M. J., & Smith, S. F. (2006). Exploiting the power of local search in a branch and bound algorithm for job shop scheduling. In *Proceedings of ICAPS 2006* (pp. 324–333).

Streeter, M. J., & Smith, S. F. (2007). Using decision procedures efficiently for optimization. In *Proceedings of ICAPS 2007* (pp. 312–319).

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285.

Van Laarhoven, P., Aarts, E., & Lenstra, K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, *40*, 113–125.

Vela, C. R., Varela, R., & González, M. A. (2010). Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, *16*(2), 139–165.

Watson, J. P., & Beck, J. C. (2008). A hybrid constraint programming/local search approach to the job-shop scheduling problem. In *Proceedings of CPAIOR 2008* (pp. 263–277).

Wu, T., Shi, L., & Duffie, N. (2010). An HNP-MP approach for the capacitated multi-item lot sizing problem with setup times. *IEEE Transactions on Automation Science and Engineering*, *7*(3), 500–511. doi:10.1109/TASE.2009.2039134.

Wu, T., Shi, L., Geunes, J., & AkartunalI, K. (2011). An optimization framework for solving capacitated multi-level lot-sizing problems with backlogging. *European Journal of Operational Research*, *214*(2), 428–441.

Yau, H., & Shi, L. (2009). Nested partitions for the large-scale extended job shop scheduling problem. *Annals of Operations Research*, *168*(1), 23–39.

Zhang, C. Y., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, *35*, 282–294.

5.3. Intensified iterative deepening A* with application to job shop scheduling

Se presenta la siguiente publicación:

- Carlos Mencía, María R. Sierra and Ramiro Varela. Intensified iterative deepening A* with application to job shop scheduling. *Journal of Intelligent Manufacturing*. 2013. DOI 10.1007/s10845-012-0726-6.
    - Estado: **Publicado online.**

J Intell Manuf
DOI 10.1007/s10845-012-0726-6

Intensified iterative deepening A* with application to job shop scheduling

Carlos Mencía · María R. Sierra · Ramiro Varela

Received: 1 August 2012 / Accepted: 19 December 2012

Abstract We propose a novel, exact any-time search strategy that combines iterative deepening A* (IDA*) with depth-first search and we consider the job shop scheduling problem with makespan minimization as a test bed. The combination of these search strategies is done so that limited depth-first searches are issued from some of the states distributed along the frontier reached by IDA* in each iteration. In this way, a proper equilibrium between intensification and diversification search effort is achieved while the algorithm keeps the capability of obtaining tight lower bounds. To evaluate the proposed strategy and to compare it with other methods, we have conducted an experimental study involving a number of conventional benchmarks with instances of various sizes. The results of these experiments show that the proposed algorithm takes less time than other methods in reaching optimal solutions for small and medium-size instances, and that it is quite competitive in reaching good solutions and good lower bounds for the instances that cannot be optimally solved.

Keywords Job shop scheduling · Heuristic search · Depth-first search · Iterative deepening A* · Hybrid algorithms

Electronic supplementary material The online version of this article (doi:10.1007/s10845-012-0726-6) contains supplementary material, which is available to authorized users.

C. Mencía · M. R. Sierra · R. Varela (✉)
Departamento de Informática, University of Oviedo,
33271 Gijón, Spain
e-mail: ramiro@uniovi.es

C. Mencía
e-mail: menciacarlos@uniovi.es

M. R. Sierra
e-mail: sierramaria@uniovi.es

Introduction

Depth-first search (DFS) is a commonly used technique for solving hard constraint satisfaction and optimization problems (CSOPs). Its main advantages are its low memory consumption and the fact that it usually returns a suboptimal solution quickly and then may obtain a sequence of improving solutions if it is given enough time. The quality of these solutions can be improved if the search is locally guided by means of heuristics, which is usually termed partially informed DFS (Pearl 1984), as shown for example in Mencía et al. (2010) for the job shop scheduling problem. However, regarding large problem instances it usually gets trapped in suboptimal regions of the search space thus being barely able to improve the last solution after some time. In order to avoid this inconvenience, a number of variants of heuristic DFS algorithms were proposed, for example limited discrepancy search (LDS) (Harvey and Ginsberg 1995) or Iterative-Deepening A* (IDA*) (Korf 1985). LDS searches first the branches of the search tree that have less heuristic discrepancies, while IDA* explores first the search tree up to a cost threshold given by the heuristic estimation of the initial state and then continues iteratively at deeper levels defined by the best heuristic estimation of the states exceeding the threshold in the previous iteration. So, LDS often reaches better solutions than DFS in the first iterations and IDA* is often able to establish better lower bounds even though it can hardly reach a solution for large problem instances. In both cases, the advantage of low memory requirement goes hand in hand with the drawback of requiring reexpanding states that have already been expanded in previous iterations.

In this paper, we propose combining IDA* with partially informed DFS with the objective of complementing their advantages and mitigating their drawbacks as far as possible. The idea behind our approach is to make use of the fact that the frontier reached by IDA* in each iteration represents the whole set of paths from the initial state towards all the instance's solutions. So, if we select a subset of these states containing a diversity of paths and spend some time searching for a number of solutions from each one of these states, we can achieve a strategy that obtains a proper balance between diversification and intensification. The intensification phases can be done by issuing limited DFS searches from each selected frontier state. This strategy is termed *intensified* IDA* or *i*IDA* for short.

In order to assess *i*IDA*, we applied it to solve the classic job shop scheduling problem (JSSP) with makespan minimization. This is an NP-hard problem and it is a paradigm of CSOP. We have conducted an experimental study involving conventional instances to compare *i*IDA* with other algorithms such as IDA*, DFS and LDS and the results show that it can solve small or medium-size instances taking less time and that for large instances it can reach better solutions and lower bounds when all algorithms are given the same time.

The remainder of the paper is structured as follows: In "Intensified iterative-deepening A*" section we describe the *i*IDA* strategy and relate it to other well-known search techniques. In "The job shop scheduling problem" section we formulate the JSSP and summarize some of the most effective methods proposed to solve it. In "An *i*IDA* algorithm for job shop scheduling" section we show how *i*IDA* may be adapted for the JSSP. "Experimental study" section reports the results of the experimental study. Finally, in "Conclusions" section we summarize the main conclusions and propose some ideas for future research.

Intensified iterative-deepening A*

We start this section with a description of IDA*. Then, we highlight some of its drawbacks for solving constraint optimization problems and introduce the extensions that give rise to *i*IDA*.

Iterative-deepening A*

IDA* is one of the most successful linear space search strategies for solving combinatorial optimization problems. It performs a series of depth-first searches bounded by a cost threshold (*CT*) that is iteratively increased until a goal state is found. Hereafter, we will consider a non-overestimating heuristic evaluation function f, such that $f(n)$ returns a lower bound on the cost of the best solution reachable from the state n. We will also denote the cost of the optimal solutions as C* and *GlobalLB* will denote a lower bound on C*.

Concretely, IDA* works as follows: given the initial state of the search space *ini*, IDA* starts setting $CT = GlobalLB = f(ini)$. Then, in each iteration, it searches depth-first expanding only states n having $f(n) \leq CT$ and discarding those having $f(n) > CT$. If a goal state is reached, the algorithm finishes returning a proven optimal solution (with cost *CT*). Otherwise, if all the states with $f(n) \leq CT$ were expanded without finding a goal, *GlobalLB* is updated to the minimum f value of the discarded states and the cost threshold is set as $CT = GlobalLB$ for the next iteration.

As it was pointed out in Korf (1985), IDA* was designed with the aim of reducing the memory requirements of the A* algorithm (Nilsson 1980). So, it is well suited for *shortest path* problems, such as puzzles or automated planning. These problems usually have search spaces with transpositions[1] and cycles where the optimal solutions are located at an initially unknown and unbounded depth.

This strategy is admissible whenever it uses a non-overestimating heuristic function, i.e. it finds an optimal solution if enough time is given. Besides, its maximum memory consumption is linear on the size of the problem, so it may be effective for facing large problem instances. However, its main drawback is that it may need to reexpand a lot of states, due to transpositions in the search space or to performing many iterations if $f(ini)$ is far from C*. Also, for very large instances it often happens that it is unable to visit the space under the first or second cost thresholds so it returns neither a solution nor a lower bound other than the trivial value $f(ini)$.

IDA* has been a subject of great interest for researchers. There have been proposals for improving its performance as, for instance, the algorithm *Enhanced* IDA* (Reinefeld and Marsland 1994). This algorithm adapts a number of techniques borrowed from two-player games to IDA*, and many other approaches intended to reduce the number of reexpansions of IDA* which are briefly commented on later. Also, in Zhou and Hansen (2006) it is shown that using breadth-first search instead of depth-first search within IDA* may lead to great improvements for a class of search spaces with cycles and transpositions. Moreover, there has been considerable research carried out in understanding and predicting the performance of IDA* algorithms (Korf et al. 2001; Zahavi et al. 2010). This search strategy has also been extended to more general search frameworks, such as multiobjective optimization, with notable examples, for instance IDMOA* (Harikumar and Kumar 1996) and PIDMOA* (Coego et al. 2009, 2012).

Extensions to cope with constraint optimization problems

We propose here three extensions of IDA* with the purpose of enhancing this algorithm to solve constraint optimization problems such that, as the job shop scheduling problem, generate a search tree. The first one is a way to

[1] In the context of graph search the notion of transposition refers to the existence of many paths connecting the same pair of states.

propagate constraints that allows IDA* to reduce the number of states explored in each iteration. This method is labeled as non conservative due to the fact that after an iteration the lowest discarded f value may not be a lower bound on C*. Then, we introduce a method to update the value of *CT* that allows the algorithm to reduce the number of iterations maintaining its admissibility. And, finally, we detail the proposed method to intensify the search from a subset of diverse frontier states, which gives the algorithm the capability of obtaining solutions from the beginning of the search. As we will see, this extension provides iIDA* with a proper diversification/intensification balance which makes this algorithm really effective at solving constraint optimization problems.

Non conservative constraint propagation

Constraint propagation is a powerful tool for solving problems and is one of the key components of constraint reasoning solvers (Meseguer 2012). Applied to a search state n, constraint propagation can deduce properties that must be satisfied by any solution reachable from n with cost not exceeding a bound used by the method (denoted here as *CP bound*). The lower the *CP bound* is the more effective the constraint propagation.[2] For example, edge finding algorithms for scheduling problems are able to detect some situations in which one operation must be processed the first or the last in a set of operations which share the same resource in any schedule whose makespan is not greater than a given value.

Combined with a search algorithm, such as DFS, constraint propagation may prune the search space to a great extent. In these cases, in order to guarantee admissibility, a conservative *CP bound* as *BestUB* -1 is taken, being *BestUB* the cost of the best solution reached so far. The original formulation of IDA* does not consider constraint propagation, as it does not compute suboptimal solutions throughout the search. Besides, its combination with these methods may make some of its theoretical properties not to hold anymore. Nevertheless, a conservative use of constraint propagation, with a known upper bound on the optimal cost, guarantees its admissibility and may increase its efficiency.

We can note that IDA* solves a series of decision problems of the form: *Does a solution with cost less than or equal to CT exist?* If the answer is *yes*, an optimal solution is returned. Otherwise, a tighter lower bound is proven and a new iteration is started off.

A variety of methods have been proposed to deal with optimization problems by solving a number of decision problems. They are usually very effective for computing accurate lower bounds, as for instance the algorithm given in Brinkkötter and Brucker (2001) for the JSSP. Another example is the query strategy proposed in Streeter and Smith (2007) for the same problem, that defines an order and different time limits for facing such decision problems with the aim of reaching good lower and upper bounds by a given running time. We propose here to exploit constraint propagation in IDA* in a similar way as it is used in these methods. So, when solving each decision problem, constraint propagation is exploited by setting *CP bound* $=$ *CT*. Therefore, more states can be pruned and then the time taken to perform a given iteration of IDA* may be reduced.

If *CT* $<$ C*, this way of using constraint propagation is not conservative in the sense that the minimum f value of the discarded states might not be a lower bound on C*. The reason for this is that a pruned state could lead to some discarded state having a lower f value. So, the minimum f value of the discarded states cannot be used for updating *GlobalLB* after completing an intermediate iteration of IDA*. In any case, we can establish that a solution with cost less than or equal to *CT* does not exist and so we can update *GlobalLB* to *CT* $+1$. So, we can state the next result:

Proposition 1 *IDA* using constraint propagation in a non conservative way is an admissible search strategy if GlobalLB is updated to CT + 1 after completing an intermediate iteration.*

In principle, to force updating the threshold one-by-one upon IDA* may lead the algorithm to do more iterations. However, our experience demonstrates that for hard instances of some problems, such as the JSSP, it is often the case that the original IDA* is not able to increase the threshold in more than one unit in each iteration. For this reason, updating $C\ Pbound = CT$ does not actually present any inconvenience. In other problems, whether or not constraint propagation is worth being exploited in this non conservative way depends on the tradeoff between its effectiveness in pruning the search space in each iteration and the overhead due to performing more iterations.

In any case, we can use the technique described in the next section, which is able to reduce to a great extent the number of iterations of IDA*, no matter how constraint propagation is used.

A strategy for updating the cost threshold

IDA* is conservative in the way it updates the cost threshold: *CT* is always set to a lower bound on C*. This policy guarantees that no states with f value greater than C* are ever expanded and so the first solution reached is optimal. However, if $f(ini) <$ C* and *GlobalLB* increases slowly, IDA* may have to perform many iterations, so reexpanding a large number of states.

Over the last years, different approaches have been proposed with the aim of reducing the number of reexpansions of IDA*. On the one hand, strategies such as MA*

[2] We consider minimization problems w.l.o.g.

(Chakrabarti et al. 1989) or MREC (Sen and Bagchi 1989) extend IDA* storing some states whose f value exceedes *CT* in a given iteration. Then, in succesive stages, the paths leading to those states are not reexpanded. Unfortunately, this technique requires additional memory resources and pays an overhead due to managing the storage of the states. Besides, it cannot be used if constraint propagation is performed with *CP bound* $<$ C*, as some states leading to optimal solutions could be pruned.

On the other hand, strategies that increase the threshold more liberally have been proposed as well. Some examples of these techniques are IDA*CR (Sarkar et al. 1991), RIDA* (Wah and Shang 1994) or DFS* (Vempaty et al. 1991).

In this section we describe DFS*. This method is a hybrid between IDA* and depth-first search, and has been shown to be well suited for some constraint optimization problems, like the Traveling Salesman Problem. As IDA*, it begins with $CT = f(ini)$, but if an iteration is completed without finding a solution, *CT* is doubled. So, the threshold could be set larger than C*. If a solution is found with cost $UB \leq CT$, IDA* automatically shifts to DFS and continues the search until finding and proving an optimal solution, this being the last iteration of the method.

In our opinion, this strategy has some drawbacks that become plain in hard problems such as the JSSP. If accurate lower bounds can be computed, doubling *CT* after an iteration of IDA* could set the threshold much larger than C*. So, although avoiding most of the reexpansions, the method may have to cope with a huge search tree in its last iteration. Moreover, IDA* could become DFS prematurely, before good lower bounds are proved. This is the case of scheduling problems, where it is usual that C* is much less than $2 \times f(ini)$.

In order to avoid this drawback, we propose using DFS*, but with a different strategy for updating the cost threshold in each iteration. Let CT^k denote the cost threshold at iteration k, and let CT^0 denote the initial cost threshold. We propose to define CT^k as

$$CT^k = \begin{cases} f(ini), & \text{if } k = 0 \\ \min\{BestUB, f(ini) + 2^{k-1}\}, & \text{if } k \geq 1 \end{cases} \quad (1)$$

where $BestUB$ is the cost of the best upper bound on C* reached so far (or ∞ if none is available). From Eq. (1) we can compute the maximum number of iterations produced by the new strategy for a problem instance, which is given by the following result.

Proposition 2 *The maximum number of iterations ($MAXIT$) performed by an IDA* algorithm using the strategy for updating the cost threshold given in Eq. (1) is*

$$MAXIT = \begin{cases} 1, & \text{if } C* = f(ini) \\ \lceil \log_2(C* - f(ini)) \rceil + 2, & \text{if } C* > f(ini) \end{cases} \quad (2)$$

This strategy is expected to eventually set *CT* to a value not much larger than C* in a few number of iterations. So, it may reach a tradeoff between the overhead due to reexpanding states and the overhead of expanding states with $f(n) >$ C* in the last iteration.

In this way, if the problem instance is not too hard to solve, it is possible to reach an optimal solution without expanding too many states beyond C*. At the same time, if the problem instance is hard so an optimal solution cannot be obtained in a reasonable time, this strategy gives IDA* the chance to compute good lower bounds, as *CT* is not increased one-by-one, and at the same time to reach suboptimal solutions, as after a small number of iterations $CT >$ C*. Furthermore, for extremely hard instances where it is not possible to explore the search space up to $f(ini)$, it is still possible to reach suboptimal solutions thanks to the method described in the next section.

Issuing limited depth-first searches beyond the frontier states

IDA* does not compute a solution until it reaches an iteration where *CT* = C*. Although the technique presented in the previous section makes the algorithm able to find a solution faster, it may not be effective if the search space is too large and there is no time available to reach the last iteration. In this case, the algorithm would not provide any solution at all.

This fact may be a serious inconvenience in a number of domains, such as manufacturing, where the time available for computing a solution is usually scarce. In order to mitigate this drawback, we introduce a technique that makes IDA* an any-time algorithm, i.e., it will be able to find a sequence of improving solutions from the beginning of the search.

At each iteration of IDA*, the cost threshold splits the search tree in two parts: Firstly, the states with f value not greater than *CT* (which are all expanded) and the second containing the states with f value beyond *CT* (a number of them generated and discarded). We term the nodes in the frontier of the first part as *frontier states*.

Definition 1 (*frontier state*) A search state n is a frontier state iff $f(n) \leq CT$ and it has at least one successor s with $f(s) > CT$.

It is also interesting to distinguish a subset of the frontier states: the *frontier leaves*.

Definition 2 (*frontier leaf*) A frontier state n is a frontier leaf iff it does not have any successor s with $f(s) \leq CT$.

We propose an extension of IDA* that yields depth-first searches limited by a number of expansions from some of the frontier leaves. This way, the method samples different regions in the search space that are similarly promising, as most of the frontier leaves will have a f value close to CT and none of them has any successor s with $f(s) \leq CT$. Diversifying the search is a commonly used technique in constraint satisfaction and optimization problems. Strategies such as limited discrepancy search or depth-first search with restarts and randomization are some well-known examples, which have been used successfully in many situations.

Limited depth-first searches issued from the frontier leaves only expand nodes n having $f(n) \leq BestUB - 1$, as the states beyond this bound do not lead to a better solution than the best one found so far. For the same reason, constraint propagation is used with $CP\ bound = BestUB - 1$.

In order to reduce the overhead of this technique and to keep a proper balance between intensification and diversification, we only issue a depth-first search from a frontier leaf when the number of states expanded in IDA* mode (they all having $f(n) \leq CT$) is larger than the number of states expanded in the limited depth-first searches issued so far [having $f(n) > CT$]. At the same time, each depth-first search is limited by a number of expansions defined as twice the number of nodes expanded until reaching the first fail (backtrack); in this way, it can make a real intensification from n as it can reach a number of solutions from n and this number is expected to be in direct ratio with the time taken by each intensification search.

So, the search can be diversified along the frontier defined by IDA* and, although the time taken to perform an intermediate iteration could be doubled, reaching good solutions throughout the search can save a lot of time in the last iteration, which is usually much more costly than the others.

Combining best-first and depth-first search has given promising results for solving shortest-path problems. Stern et al. (2010) propose a combination in which a depth-first search limited on a maximum depth is yielded from each node to be expanded by an A* algorithm. This approach lets the algorithm save a lot of memory thanks to the depth-first search component. However, limiting the depth of the depth-first searches is not appropriate for a search tree where the solutions are located at a known depth. This is the case of many scheduling problems, including the one we face in this paper.

Discussion of the proposed extensions

Thanks to the combination of two algorithms as IDA*, which is good at obtaining lower bounds, and DFS, which can quickly obtain upper bounds, we expect the proposed algorithm to be effective at obtaining good lower and upper bounds at the same time.

An important observation here is that depth-first searches are more intensively issued from promising frontier leaves. This can be understood from the fact that all frontier leaves have similar f values and so the most promising ones are those at the deepest levels of the search tree. So, due to the form used to keep a balance between the number of expanded nodes by IDA* and DFS, the number of discarded frontier leaves between two consecutive DFSs is in inverse ratio to the mean depth of these nodes in a region of the search tree. Also, the promising frontier leaves are reached first due to the fact that in each iteration of IDA* the search is guided locally by the heuristic. This helps to get good solutions early, which in its turn, helps to reduce the search space.

The job shop scheduling problem

The JSSP is a paradigm of scheduling and constraint optimization problems. It is NP-hard in the strong sense (Garey and Johnson 1979) and is considered as one of the most intractable problems known so far. Over the last years, the JSSP has interested researchers due to its difficulty and the fact that it models many real life problems. For example in Meeran and Morshed (2011) a steel mill problem is modeled as a pure JSSP with makespan minimization. However, in other cases some characteristics have to be added such as setup times (González et al. 2012), uncertain durations (Temiz and Erol 2004; González Rodríguez et al. 2010) or other objective functions (Sierra and Varela 2010). Also, the JSSP with makespan minimization is often used as a test bed to evaluate and compare algorithms; for example in Pérez et al. (2012) the authors use this problem to analyze a number of niching techniques in the framework of multi-modal genetic algorithms.

In the remainder of this section we give a formulation of the JSSP and present briefly a number of important solving methods proposed so far.

Problem formulation

The job shop scheduling problem requires scheduling a set of N jobs $\{J_1, \ldots, J_N\}$ on a set of M resources or machines $\{R_1, \ldots, R_M\}$. Each job J_i consists of a set of tasks or operations $\{\theta_{i1}, \ldots, \theta_{iM}\}$ to be sequentially scheduled. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, a fixed duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. The JSSP has three constraints: precedence, capacity and non-preemption. Precedence constraints translate into linear inequalities of the type: $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$. Capacity constraints translate into disjunctive constraints of the form: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, if $R_v = R_w$. Non-preemption requires assigning machines to operations without interruption during their whole processing time.

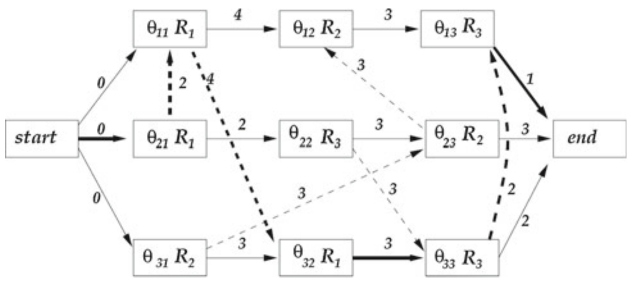


Fig. 1 A feasible schedule to a problem with 3 jobs and 3 machines. *Bold face arcs* show a critical path whose length (makespan) is 12

The objective is to come up with a feasible schedule so that the completion time, i.e. the *makespan*, is minimized. This problem is denoted as $J||C_{max}$ in the $\alpha|\beta|\gamma$ notation (Graham et al. 1979).

The disjunctive graph model

A problem instance can be represented by a disjunctive graph $G = (V, A \cup E)$ (Roy and Sussman 1964). Each node in the set V represents an actual operation, with the exception of the dummy nodes *start* and *end*, which represent operations with processing time 0. The arcs of A are called *conjunctive arcs* and represent precedence constraints and the arcs of E are called *disjunctive arcs* and represent capacity constraints. E is partitioned into subsets E_i with $E = \cup_{\{i=1,\ldots,M\}} E_i$. E_i includes an *arc* (v, w) for each pair of operations requiring R_i. The arcs are weighted with the processing time of the operation at the source node. Node *start* is connected to the first operation of each job and the last operation of each job is connected to node *end*.

To calculate a solution, each disjunctive arc (v, w) must be fixed so that v is processed before w or conversely w is processed before v. Thus, calculating a solution can be viewed as a process in which the disjunctive arcs are fixed in some way. Like this, a partial schedule is given by a subgraph of G where some of the disjunctive arcs are not fixed yet.

A feasible schedule S is represented by an acyclic subgraph G_S of $G, G_S = (V, A \cup H)$, where $H = \cup_{i=1,\ldots,M} H_i$, H_i being a processing ordering for the operations requiring R_i. The makespan is the cost of a *critical path*. A critical path is a longest path from node *start* to node *end*. A critical path contains a set of *critical blocks*. A critical block is a maximal sequence, of consecutive operations in a critical path requiring the same machine. Figure 1 shows a solution graph for an instance with 3 jobs and 3 machines.

Solving methods

The notorious difficulty of the JSSP along with the simplicity of its formulation and its excellent practical applications make this problem very interesting for both Artificial Intelligence and Operations Research. So, a great number of techniques have been applied to the $J||C_{max}$ problem over the last decades.

Many approximate algorithms have been proposed to solve the JSSP. Most of them are based on one or several meta-heuristics. For example, in Mattfeld (1995), some genetic algorithms are combined with local searchers to obtain quite competitive results. Local search methods were usually designed from smart neighborhood structures devised previously in Van Laarhoven et al. (1992), Dell' Amico and Trubian (1993), Taillard (1994). All of them rely on exchanging the processing order of operations in a critical path of the current solution. Some of these structures were also considered in Balas and Vazacopoulos (1998), together with a quite different neighborhood derived from the well-known shifting bottleneck heuristic; these structures are then embedded in a variable depth search procedure termed GLS (guided local search). Regarding more recent proposals, the tabu search algorithm termed *i*-TSAB (Nowicki and Smutnicki 2005) and the hybrid tabu search/simmulated annealing method proposed in Zhang et al. (2008) deserve special mention. These algorithms are able to compute state-of-the-art solutions to extremely hard instances. A thorough study of the outstanding *i*-TSAB algorithm was carried out in Watson et al. (2006).

Meta-heuristic algorithms usually reach high-quality solutions, and their effectiveness depends on their ability to reach a proper tradeoff between diversification across the whole search space and intensification in promising areas. This idea has motivated the design of the proposed *i*IDA* algorithm, that intensify the search in promising regions by yielding limited depth-first searches from some of the frontier states of IDA*.

Also, effective exact methods to cope with the JSSP have been proposed. One advantage of exact techniques is that they may certify that a solution is optimal or at least provide a good lower bound on the optimal value. However, as they have to do a systematic search, they are not usually able to come up with solutions as good as those obtained by non-exact methods when solving very large instances. These algorithms usually combine search with powerful constraint propagation techniques that are able to reduce the search space to a great extent by reasoning and deducing properties that must be hold in any improving schedule w.r.t. the best one found so far. Some examples of successful constraint propagation methods for this problem are timetabling, edge-finding or energetic reasoning (Dorndorf et al. 2000; Laborie 2003; Baptiste et al. 2001). Regarding search strategies, branch and bound has been widely used in this domain, and a number of variable and value ordering heuristics have been proposed for guiding the search, such as texture-based (Beck and Fox 2000) and slack-based heuristics (Smith and Cheng 1993).

In this context, Brucker's branch and bound algorithm (Brucker et al. 1994) is one of the most effective methods at finding and proving optimal solutions to medium-size instances. Although it was proposed several years ago, it is still often chosen as a building method for developing new ideas on. For example, it was used to test an efficient query strategy for solving the JSSP (Streeter and Smith 2007) and also in Gharbi and Labidi (2010) to obtain lower bounds of JSSP instances by solving non-polynomial relaxations. It was also used to perform an analysis on the relationship between the jobs:machines ratio and the landscape's shape of the solution space (Streeter and Smith 2006b).

For larger and more difficult instances that cannot be solved in a reasonable time, solution guided multi-point constructive search (SGMPS) (Beck 2007) stands out as the state-of-the-art constructive method regarding the quality of the solutions returned. SGMPS performs backtracking search with restarts and uses the best solutions found so far and randomized texture-based heuristics to guide the search.

Finally, hybrid algorithms combining exact and local search methods have been proposed as well. For example, in Streeter and Smith (2006a), the authors propose combining an iterated local search method with Brucker's $B\&B$ algorithm. Also, in Watson and Beck (2008) SGMPS is combined with tabu search. In both cases performance improvements are achieved over the two building techniques.

An *i*IDA* algorithm for job shop scheduling

We have implemented an *i*IDA* algorithm that uses the informed depth-first search algorithm proposed in Mencía et al. (2010) as a building block. We first present a brief description of the main components of this algorithm. Then, in the second subsection, we include the particular details of the proposed *i*IDA* algorithm for the $J||C_{max}$ problem.

Base DFS algorithm

Based on the well-known Brucker's $B\&B$ algorithm (Brucker et al. 1994), DFS searches depth-first an n-ary tree in which each state corresponds to a disjunctive graph with a number of arcs fixed (representing a partial schedule).

The successors of a state n are generated from a feasible schedule S that includes all disjunctive arcs fixed in n. The schedule S is computed by a variant of the greedy $G\&T$ algorithm proposed in Giffler and Thompson (1960). The expansion mechanism is as follows: an operation in a critical path in S is moved to the beginning or to the end of its critical block, and a number of disjunctive arcs are fixed accordingly in the new state. Furthermore, a powerful edge-finding technique (constraint propagation), termed *immediate selection* (Carlier and Pinson 1990), which fixes additional arcs in each successor state is exploited. In order to fix the disjunctive arc (u, v), it tries to prove that scheduling v before u does not lead to a schedule improving the best solution found so far (with makespan *BestUB*).

Finally, for each generated state n a heuristic estimation $f(n)$ is computed. $f(n)$ is a lower bound on the cost of the best solution reachable from n and it is used for pruning states under the condition $f(n) \geq BestUB$ and also to guide the search towards promising solutions. Two different lower bounds are considered: f_{JPS} and f_{IS}. f_{JPS} is based on preemptive one machine sequencing problem relaxations and requires computing the so-called Jackson's preemptive schedule (JPS). f_{IS} is a destructive lower bound that uses the *immediate selection* procedure to prove infeasibility. It tries different values in the interval $[f_{JPS}(n), BestUB]$ and returns the lowest value that does not produce an inconsistent state after executing the constraint propagation method. f_{IS} is more informed and computationally more expensive than f_{JPS} and, in accordance with their experiments, the authors propose using f_{IS} for medium-size instances or at shallow levels of the search tree for large instances and f_{JPS} at deeper levels of large instances.

For further details of the DFS algorithm we refer the interested reader to Mencía et al. (2010).

Some particular details

The *i*IDA* algorithm proposed here uses the DFS algorithm presented above as a building block. So, this algorithm is used for both working on IDA* mode, limiting the search by a cost threshold, and performing search intensifications from some of the frontier states. In both cases, the constraint propagation method used is the *immediate selection* procedure, setting *CP bound* as indicated in "Non conservative constraint propagation" section and "Issuing limited depth-first searches beyond the frontier states" section, respectively. Finally, the heuristic estimations (f values) are those used by the DFS algorithm.

Note that the above DFS algorithm needs to compute a solution from each search state to be expanded in order to generate its successors. This is done by means of a simple greedy algorithm. Therefore, every algorithm built on DFS will compute suboptimal solutions along the search, even the classical IDA*, in the same way as an any-time algorithm. Obviously, if we consider other problems or other search spaces for $J||C_{max}$ problem, this may no longer be so and then IDA* could not return a solution unless it completes the last iteration.

Experimental study

In this section we report the results from the experimental study. In this study, we started evaluating the extensions of

Table 1 Evaluation of the components of *i*IDA* on 10 × 10 instances

IDA*			IDA* + CP			IDA* + CP + ST			*i*IDA*		
T_p	T_f	# Iter.	T_p	T_f	# Iter.	T_p	T_f	# Iter.	T_p	T_f	# Iter.
209.75	208.96	58.00	32.62	32.62	58.56	4.59	2.71	7.94	4.29	1.65	7.89

The values reported are the times taken to find and prove optimal solutions (in seconds), as well as the number of iterations of the IDA* component

Table 2 Evaluation of the *i*IDA* components on Applegate and Cook's selected instances

IDA*			IDA* + CP			IDA* + CP + ST			*i*IDA*		
$\bar{E}_{UB}$	$\bar{E}_{LB}$	# Sol.	$\bar{E}_{UB}$	$\bar{E}_{LB}$	# Sol.	$\bar{E}_{UB}$	$\bar{E}_{LB}$	# Sol.	$\bar{E}_{UB}$	$\bar{E}_{LB}$	# Sol.
4.17	1.29	2/12	3.28	0.87	3/12	2.53	1.02	4/12	1.37	1.05	5/12

The values reported are the errors on the lower and upper bound reached by 3,600 s

IDA* proposed in this paper for solving the $J||C_{max}$ problem. Then, we compared our approach with other search strategies. All the experiments were performed on an Intel Xeon (2.26 GHz), 4 GB RAM. and the algorithms were coded in C programming language.

Evaluation of the proposed extensions

In order to assess each component of *i*IDA* we conducted two series of experiments across two sets of well-known instances taken from the OR-Library (Beasley 1990). The first one includes eighteen instances of size 10 × 10 (10 jobs and 10 machines): FT10, ABZ5–6, LA16–20 and ORB01–10, which can all be solved to optimality within a reasonable time. Then we considered a set of twelve heterogeneous instances selected in Applegate and Cook (1991) as very hard to solve: FT10 (10 × 10), FT20 (10 × 10), LA21, LA24, LA25 (15 × 10), LA27, LA29 (20 × 10), LA38, LA40 (15 × 15) and ABZ7–9 (20 × 15).

We made this evaluation incrementally, beginning with the classical IDA* and then introducing one by one each extension proposed in this paper. This way, we have four different algorithms: IDA*, which uses constraint propagation in a conservative way ($CP\ bound = BestUB - 1$), IDA* + CP, which uses non conservative constraint propagation, IDA* + CP + ST, which introduces the new strategy for updating the cost threshold between two iterations of IDA*, and finally *i*IDA*, which extends the previous algorithm by yielding depth-first searches from some of the frontier leaves.

Table 1 shows the time (in seconds) taken to find (T_f) and to prove (T_p) an optimal solution, as well as the number of iterations performed by each algorithm, averaged for all the instances of size 10 × 10.

As we can see, using non conservative constraint propagation allows IDA* + CP to reduce the time taken to find and prove optimal solutions in about one order of magnitude w.r.t. that of IDA*. At the same time the average number of iterations is slightly increased, which indicates that in most cases IDA* increments the cost threshold one-by-one along the search. Also, IDA* + CP + ST reduces exponentially the number of iterations w.r.t IDA* + CP, in accordance with the maximum number of iterations established in "A strategy for updating the cost threshold" section. Finally, the intensification via DFS allows *i*IDA* to further reduce the time taken to find and prove optimal solutions and also to slightly reduce the number of iterations w.r.t. IDA* + CP + ST.

Regarding Applegate and Cook's selected instances, the majority of them cannot be solved to optimality within a reasonable time, so we have given the algorithms a time limit of 3,600 s. For each instance, the algorithms returned a lower and an upper bound on the optimal cost (LB and UB resp.). Then, from these values, we computed the error in percentage terms (E_{LB} and E_{UB}) w.r.t. the best known lower and upper bounds (BK_{LB} and BK_{UB} resp.) using the following expressions:

$$E_{LB} = 100 \times \frac{LB - BK_{LB}}{BK_{LB}} \qquad (3)$$

$$E_{UB} = 100 \times \frac{BK_{UB} - UB}{BK_{UB}} \qquad (4)$$

Table 2 shows the average error in percentage terms for both upper and lower bounds ($\bar{E}_{UB}$ and $\bar{E}_{LB}$ resp.) and the number of instances solved to optimality by the time limit (# Sol.) as well.

IDA* yields the greatest error in both upper and lower bounds, and it is only able to solve to optimality the smallest instances (FT10 and FT20) by the time limit. IDA* + CP completes each iteration faster than IDA*, so it obtains better solutions and lower bounds and it is able to prove the optimality of one instance more than IDA* (LA24, of size 15 × 10). IDA* + CP + ST solves one instance more than IDA* + CP (LA25, of size 15 × 10) and reduces the average error in the upper bounds w.r.t. IDA* + CP. However, the lower bounds get worse. Finally, *i*IDA* is the algorithm that reaches the best upper bounds, while the lower bounds get slightly worse w.r.t. IDA* + CP + ST, but in any case they are

Table 3 Results from DFS, IDA*, LDS and iIDA* accross instances of size 10×10

	T_p	T_f
DFS	5.27	3.35
LDS	59.15	9.13
IDA*	209.75	208.96
iIDA*	**4.29**	**1.65**

Bold values indicate the best ones of all methods
The values reported are the times taken to find and prove optimal solutions (in seconds)

much better than those obtained by IDA*. It is remarkable that this algorithm solves to optimality a very large instance (LA38, of size 15×15). So, we consider iIDA* as the best option to reach a good tradeoff between lower and upper bounds for large instances that cannot be optimally solved.

As we have seen in these experiments, each extension of IDA* is worth using in the iIDA* algorithm and so we considered this algorithm in the remainder of the experimental study.

Comparison with other search strategies

The second part of the experimental study is intended for comparing iIDA* with a number of well-known search strategies, namely its two components DFS and IDA*, and also with LDS. DFS is the algorithm presented in Mencía et al. (2010), and the other algorithms were built on it. We consider LDS as it is one paradigmatic example of linear space strategy that diversifies the search and has an important track record of success in solving constraint optimization problems.

We made this comparison across several instances with $n \times m$ ranging from 10×10 to 20×20. Besides the instances of size 10×10 and the selected instances used to evaluate the components of iIDA* in the previous section, we additionally considered five sets of instances. Three sets taken from the OR-Library with five instances each: LA21–25 (15×10), LA26–30 (20×10) and LA36–40 (15×15). And two more sets, proposed in Taillard (1993), containing extremely hard instances, each one with 10 instances: Tai11–20 (20×15) and Tai21–30 (20×20).

Table 3 shows the results from the instances of size 10×10. These instances are all solved to optimality by all the methods. For each algorithm, we report the time taken (in seconds) to find (T_f) and to prove (T_p) optimal solutions averaged out for all the instances. Also, in these experiments, all the algorithms used the light heuristic f_{JPS}. As we can see, both DFS and iIDA* are clearly the best algorithms for solving this kind of instances. But iIDA* outperforms DFS as it reduces the time taken in finding and proving optimal solutions by about 50.71 and 18.62 % respectively.

Table 4 Results from Wilcoxon's paired tests for 10×10 instances

Hypothesis	p value (T_p)	p value (T_f)
iIDA* vs. DFS	3.49E−03	3.85E−03
iIDA* vs. LDS	3.82E−05	1.68E−03
iIDA* vs. IDA*	3.82E−06	3.82E−06

Following García et al. (2010) we have also done Wilcoxon's paired tests aimed at identifying significant differences among the results from iIDA* those reached by the other search strategies. Table 4 shows the p values obtained when comparing each algorithm with iIDA* w.r.t. the time taken to both find and prove an optimal solution. The alternative hypothesis is always that the time taken by iIDA* is smaller than the time taken by the other method. As we can see, all the tests gave very low p values, showing that there is strong statistical evidence that iIDA* outperforms IDA*, DFS and LDS.

The remaining six sets of instances are very hard to solve to optimality, so we have given the algorithms a time limit of 3,600 s. Table 5 reports the error in percentage terms in upper ($\bar{E}_{UB}$) and lower bounds ($\bar{E}_{LB}$) w.r.t. the best known bounds. These values were computed using the expressions (3) and (4). The results are averaged out for each set of instances. Here, the algorithms used the heavy heuristic f_{IS}.

Clearly, iIDA* and LDS are better than DFS and IDA* for computing upper bounds in all benchmark sets. It is remarkable that iIDA* achieves the best results in 5 out of 6 sets. At the same time, although LDS reaches the same error as iIDA* for the 15×15 instances and is slightly better than iIDA* for the instances of 20×10, the difference in favor of iIDA* is greater for most cases, especially for the largest and hardest instances with size 20×20. Regarding lower bounds, the best methods overall are IDA* and iIDA* as could have been expected. Here, iIDA* reaches the best results in 4 out of the 6 sets of instances, and in the two sets where IDA* is better than iIDA* the differences between these algorithms are really small.

To enhance these conclusions we have done a series of Wilcoxon's paired tests aimed at establishing significant differences among iIDA* and the other search strategies w.r.t. the lower and upper bounds computed by the time limit across all the instances in the six sets. Table 6 reports the p values from these tests, which show very significant statistical evidence that the solutions computed by iIDA* have less error than those returned by DFS, LDS and LDS. Also, the tests clearly support the hypotheses that iIDA* computes more accurate lower bounds than both DFS and LDS. At the same time, there is no statistical evidence that iIDA* reaches better lower bounds than IDA*. Nevertheless, we have also made a test under the alternative hypothesis that IDA* returns better

Table 5 Results from DFS, IDA*, LDS and *i*IDA* across the large instances (3,600 s)

	Selected		15 × 10		20 × 10		15 × 15		20 × 15		20 × 20	
	$\bar{E}_{UB}$	$\bar{E}_{LB}$	$\bar{E}_{UB}$	$\bar{E}_{LB}$	$\bar{E}_{UB}$	$\bar{E}_{LB}$	$\bar{E}_{UB}$	$\bar{E}_{LB}$	$\bar{E}_{UB}$	$\bar{E}_{LB}$	$\bar{E}_{UB}$	$\bar{E}_{LB}$
DFS	2.58	2.06	0.27	0.25	2.63	**0.57**	0.58	1.98	8.27	2.84	9.70	2.99
IDA*	4.17	1.29	1.77	0.44	2.30	**0.57**	1.64	0.60	8.31	**2.14**	8.66	**1.79**
LDS	1.60	2.28	0.21	0.76	**0.60**	**0.57**	**0.10**	1.03	4.21	2.84	6.58	2.99
*i*IDA*	**1.37**	**1.05**	**0.04**	**0.08**	0.88	**0.57**	**0.10**	**0.21**	**3.99**	2.17	**5.16**	1.83

Bold values indicate the best ones of all methods
The values reported are the errors in percentage terms in the lower and upper bounds obtained by each algorithm averaged out for each set of instances with the same size

Table 6 Results from Wilcoxon's paired tests for large instances

Hypothesis	p value (E_{UB})	p value (E_{LB})
*i*IDA* vs. DFS	2.00E−06	3.20E−05
*i*IDA* vs. LDS	2.76E−02	9.72E−05
*i*IDA* vs. IDA*	9.13E−07	1.47E−01

lower bounds than *i*IDA* obtaining a much greater p value of 8.59E−01.

So, these results support our claim that *i*IDA* can be quite competitive with a strategy such as IDA*, which is very efficient at obtaining lower bounds in tree search problems, and that it can outperform other strategies such as DFS or LDS, which usually can obtain efficiently good upper bounds thanks to the use of heuristic guidance or search diversification via restarts.

Conclusions

We have seen that by combining two different search strategies such as IDA* and informed depth-first search it is possible to get an efficient search algorithm for solving constraint optimization problems, which keeps the strongest points and avoids some of the inconveniences of both algorithms. The key for the success of the resulting algorithm, termed *i*IDA*, is a proper balance between diversification and intensification efforts of the search, as well as a strategy that allows the IDA* component to carry out each iteration quickly and at the same time to update the cost threshold rationally. As a test bed we have considered the job shop scheduling problem with makespan minimization. The results of the experimental study show that *i*IDA* is able to certify optimal solutions for small and medium-size instances taking much less time than IDA*, LDS or depth-first search alone. Also, for large instances, which in most cases cannot be solved to optimality, *i*IDA* reaches better lower and upper bounds than the other algorithms when all of them are given the same time. *i*IDA* is particularly good when dealing with extremely large instances where it is only possible to visit a very small fraction of the search space.

As future work we plan to apply *i*IDA* to other constraint optimization problems, in particular to scheduling problems with other constraints or different objective functions.

Acknowledgments We wish to thank the anonymous referees for their useful comments that helped improving this work. We are also grateful to Mr. Neil Beckingham for his thorough work at revising the English language in this paper. This research has been supported by the Spanish Government under research project MICINN-FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP09105.

References

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, *3*, 149–156.
Balas, E., & Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, *44*(2), 262–275.
Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling*. Norwell, MA: Kluwer.
Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society 41*(11), 1069–1072. http://www.jstor.org/stable/2582903.
Beck, J. C. (2007). Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research (JAIR)*, *29*, 49–77.
Beck, J. C., & Fox, M. S. (2000). Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, *117*(1), 31–81.
Brinkkötter, W., & Brucker, P. (2001). Solving open benchmark instances for the job-shop problem by parallel heads and tails adjustments. *Journal of Scheduling*, *4*(1), 53–64.
Brucker, P., Jurisch, B., & Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, *49*, 107–127.
Carlier, J., & Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, *26*, 269–287.
Chakrabarti, P. P., Ghose, S., Acharya, A., & Sarkar, S. C. D. (1989). Heuristic search in restricted memory. *Artificial Intellignce*, *41*(2), 197–221.
Coego, J., Mandow, L., & Pérez de la Cruz, J. (2009). A new approach to iterative deepening multiobjective A. In *AI*IA*, pp 264–273.
Coego, J., Mandow, L., & Pérez de la Cruz, J. (2012). A comparison of multiobjective depth-first algorithms. *Journal of Intelligent Manufacturing*, 1–9. doi:10.1007/s10845-012-0632-y.
Dell' Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, *41*, 231–252.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000). Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, *122*, 189–240.

García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, *180*(10), 2044–2064.

Garey, M., & Johnson, D. (1979). *Computers and intractability*. San Francisco, CA: Freeman.

Gharbi, A., & Labidi, M. (2010). Extending the single machine-based relaxation scheme for the job shop scheduling problem. *Electronic Notes in Discrete Mathematics*, *36*, 1057–1064.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, *8*, 487–503.

González, M. A., Vela, C. R., González-Rodríguez, I., & Varela, R. (2012). Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. *Jornal of Intelligent Manufacturing*. doi:10.1007/s10845-011-0622-5.

González Rodríguez, I., Vela, C. R., & Puente, J. (2010). A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty. *Journal of Intelligent Manufacturing*, *21*, 65–73.

Graham, R., Lawler, E., Lenstra, J., & Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Harikumar, S., & Kumar, S. (1996). Iterative deepening multiobjective A. *Information Processing Letters*, *58*(1), 11–15.

Harvey, W. D., & Ginsberg, M. L. (1995). Limited discrepancy search. In *Proceedings of IJCAI 1995 (1)*, pp. 607–615.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*, 97–109.

Korf, R. E., Reid, M., & Edelkamp, S. (2001). Time complexity of iterative-deepening-A*. *Artificial Intelligence*, *129*(1–2), 199–218.

Laborie, P. (2003). Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intellignece*, *143*(2), 151–188.

Mattfeld, D. (1995). *Evolutionary search and the job shop investigations on genetic algorithms for production scheduling*. Berlin: Springer.

Meeran, S., & Morshed, M. (2011). A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study. *Journal of Intelligent Manufacturing*, *23*(4), 1063–1078.

Mencía, C., Sierra, M. R., & Varela, R. (2010). Partially informed depth-first search for the job shop problem. In *Proceedings of ICAPS 2010*, pp. 113–120.

Meseguer, P. (2012). Towards 40 years of constraint reasoning. *Progress in Artificial Intelligence*, 1–19. doi:10.1007/s13748-011-0006-2.

Nilsson, N. (1980). *Principles of artificial intelligence*. Palo Alto, CA: Tioga.

Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, *8*(2), 145–159.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Reading, MA: Addison-Wesley.

Pérez, E., Posada, M., & Herrera, F. (2012). Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. *Journal of Intelligent Manufacturing*, *23*(3), 341–356.

Reinefeld, A., & Marsland, T. A. (1994). Enhanced iterative-deepening search. *IEEE Pattern Analysis and Machine Intelligence*, *16*(7), 701–710.

Roy, B., & Sussman, B. (1964). *Les problemes d'ordonnancements avec contraintes disjonctives*. Technical report, notes DS no. 9 bis, SEMA, Paris.

Sarkar, U. K., Chakrabarti, P. P., Ghose, S., & Sarkar, S. C. D. (1991). Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence*, *50*(2), 207–221.

Sen, A. K., & Bagchi, A. (1989). Fast recursive formulations for best-first search that allow controlled use of memory. In *IJCAI*, pp. 297–302.

Sierra, M. R., & Varela, R. (2010). Pruning by dominance in best-first search for the job shop scheduling problem with total flow time. *Journal of Intelligent Manufacturing*, *21*(1), 111–119.

Smith, S. F., & Cheng, C. C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of AAAI 1993*, pp. 139–144.

Stern, R., Kulberis, T., Felner, A., & Holte, R. (2010). Using lookaheads with optimal best-first search. In *AAAI*.

Streeter, M. J., & Smith, S. F. (2006a). Exploiting the power of local search in a branch and bound algorithm for job shop scheduling. In *Proceedings of ICAPS 2006*, pp. 324–333.

Streeter, M. J., & Smith, S. F. (2006b). How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research (JAIR)*, *26*, 247–287.

Streeter, M. J., & Smith, S. F. (2007). Using decision procedures efficiently for optimization. In *Proceedings of ICAPS 2007*, pp. 312–319.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285.

Taillard, E. (1994). Parallel taboo search techniques for the job shop scheduling problem. *INFORMS Journal on Computing*, *6*(2), 108–117.

Temiz, I., & Erol, S. (2004). Fuzzy branch-and-bound for flow shop scheduling. *Journal of Intelligent Manufacturing*, *15*, 449–454.

Van Laarhoven, P., Aarts, E., & Lenstra, K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, *40*, 113–125.

Vempaty, N. R., Kumar, V., & Korf, R. E. (1991). Depth-first versus best-first search. In *AAAI*, pp. 434–440.

Wah, B. W., & Shang, Y. (1994). Comparison and evaluation of a class of IDA* algorithms. *International Journal of Artificial Intelligence Tools*, *3*(4), 493–523.

Watson, J. P., Beck, J. C. (2008). A hybrid constraint programming/local search approach to the job-shop scheduling problem. In *Proceedings of CPAIOR 2008*, pp. 263–277.

Watson, J. P., Howe, A. E., & Whitley, L. D. (2006). Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem. *Computers & OR*, *33*, 2623–2644.

Zahavi, U., Felner, A., Burch, N., & Holte, R. C. (2010). Predicting the performance of IDA* using conditional distributions. *Journal of Artificial Intelligence Research (JAIR)*, *37*, 41–83.

Zhang, C. Y., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & OR*, *35*, 282–294.

Zhou, R., & Hansen, E. A. (2006). Breadth-first heuristic search. *Artificial Intelligence*, *170*(4–5), 385–408.

5.4. An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators

Se presenta la siguiente publicación:

- Carlos Mencía, María R. Sierra and Ramiro Varela. An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators. *International Journal of Production Research*. 2013. DOI 10.1080/00207543.2013.802389
    - Estado: **Aceptado (en prensa).**

Nota: Al principio del artículo se adjunta el correo electrónico de aceptación y la información que aparece sobre el mismo en el sistema de gestión de publicaciones de la editorial.

Ramiro Varela

De:	onbehalfof+dolgui+emse.fr@manuscriptcentral.com en nombre de dolgui@emse.fr
Enviado el:	lunes, 22 de abril de 2013 23:46
Para:	ramiro@uniovi.es
Asunto:	Acceptance of your revised paper ID TPRS-2012-IJPR-0920.R1
Datos adjuntos:	revised.pdf

22-Apr-2013

Dear Prof. Varela:

Ref: TPRS-2012-IJPR-0920.R1, "An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators" by Mencía, Carlos; Sierra, María; Varela, Ramiro

Our referees have considered the revised version of your paper and have recommended publication in International Journal of Production Research. We are pleased to accept your paper in its current form, which will now be forwarded to the publisher for copy editing and typsetting.
You will receive proofs for checking, and instructions for transfer of copyright and off-print copying in due course.

The publisher requests that proofs are checked and returned within 48 hours.

Please note, if you have provided a PDF file for the peer-review process, you will need to promptly supply your original source files when contacted by the publisher. These source files will prevent any delay in the copy editing and typesetting process and the eventual publication of your manuscript.

Thank you for your contribution to International Journal of Production Research and we look forward to receiving further submissions from you.
Will you please ensure that your details on the IJPR ScholarOne Manuscripts on-line submission and review site are maintained up-to date (URL http://mc.manuscriptcentral.com/tprs).

Have you read the latest up-to-date research articles published in IJPR? A voucher is available that entitles you to free access to journals within your subject area for 7 days. Please visit: http://www.tandf.co.uk/journals/authoraccess/pst.asp. Please note we have a free table of contents alerting service for this Journal. To register for this free service visit http://www.tandfonline.com/loi/tprs20 and select ‘Alert Me’ under the Journal cover.

Sincerely

Alexandre Dolgui
Editor-in-Chief, International Journal of Production Research dolgui@emse.fr

You are logged in as: Ramiro Varela | Sign out | Change Password

Central Article Tracking System

- **Articles in Production**
- Published Articles
- Frequently Asked Questions
- Author Services

Article Requirements | **Additional Actions**

This page lists key information and actions that are required for publication. Other options you may need for this article are listed in the "Additional Actions" area (top right tab).

An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators
(ID: 802389 DOI:10.1080/00207543.2013.802389)

Journal: International Journal of Production Research (Download Current Citation: RIS BibTex)
Estimated Publication date - 19 Jun 2013 (Online)
Authors: Carlos Mencía, María Sierra & Ramiro Varela

A copyright agreement licensing us to publish this article has been received and approved.

If you have not yet sent in the signed form, this must also be posted to your Production Editor.

This article is not yet ready for Proofreading. Please check back 23 May 2013

Copyright Supplied

The Production Editor for this journal is James Baldock. (Contact Production Editor)

International Journal of Production Research
Vol. 00, No. 00, 00 Month 2013, 1–30

RESEARCH ARTICLE

An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators

Carlos Mencía, María R. Sierra and Ramiro Varela*

University of Oviedo. Campus de Viesques, 33271, Spain
(*Received 00 Month 200x; final version received 00 Month 200x*)

We confront the job shop scheduling problem with human operators and total flow time minimization. To solve this problem we propose a hybrid search algorithm that interleaves best-first and depth-first search. The efficiency of this algorithm relies on a proper combination of diversification and intensification capabilities of best-first and depth-first searches respectively, as well as two admissible heuristics and some global pruning rules designed from problem domain knowledge. We conducted an experimental study on a benchmark set with small, medium-size and large instances. The results of this study show that the algorithm is efficient thanks to the combination of all its components and that it compares favorably with an outstanding constraint programming solver specialized in scheduling problems.

Keywords: job shop scheduling with operators, total flow time, best-first search, depth-first search, hybrid search algorithms, anytime algorithms, global pruning rules.

1. Introduction

The job shop scheduling problem (JSP) with an additional resource type and with the objective of minimizing the makespan has been recently proposed by Agnetis *et al.* (2011) where it is denoted $JSO(n,p)$. This problem is motivated by manufacturing processes in which part of the work is done by human operators sharing the same set of tools. The problem is formalized as a classic JSP in which the processing of an operation on a given machine requires the assistance of one of p available operators.

Agnetis *et al.* (2011) made a thorough study of this problem and established the minimal NP-hard cases. Also, they proposed a number of exact and approximate algorithms

We are grateful to the anonymous reviewers for their useful comments and suggestions. This research has been supported by the Spanish Government under project FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP09105.
*Corresponding author. Email: ramiro@uniovi.es

ISSN: 1478-6451 print/ISSN 1478-646X online
© 2013 Taylor & Francis
DOI: 10.1080/1478645YYxxxxxxx
http://www.informaworld.com

to cope with this problem which are evaluated across a set of instances generated from that minimal relevant cases. The results of their experimental study show that instances with 3 jobs, 3 machines, 2 operators and a number of 30 operations per job may be hard to solve to optimality.

The classic JSP has interested researchers over the last decades because of its difficulty and its proximity to real life problems. In some cases the pure JSP is able to model exactly a real problem, for example Meeran and Morshed (2012) model a work shop at a steel mill as a classic JSP. However, in most cases, some additional characteristics are required in order to model a particular problem. So, inspired by different problems, a number of variants of the JSP has been considered in the literature. To name just a few recent ones, we can consider the following. Driebel and Monch (2012) consider a JSP in which an operation can be processed in one from a set of identical parallel machines. Liu *et al.* (2012) introduce a job value function that deteriorates over time for weighting the total flow time of sub-jobs. Niu *et al.* (2012) consider different operation modes so that the time taken and the cost spent by one operation depend on the mode in which it is performed. Also, Rabiee *et al.* (2012) consider a JSP where each operation may be processed by any from a subset of the machines with the objective of minimizing the makespan and the total operation cost.

Zouba *et al.* (2009) remark that research in classical scheduling theory has mainly concentrated on machine and job characteristics such as machine availability or job processing sequences; and that the models have ignored in most of the cases considerations linked with operator interventions on the machines by assuming their number infinite. However, human resources are getting more and more critical in production systems, and it is not enough to concentrate only on the physical resources in order to expect a reasonable solution. So, they are being considered in some recent models. For instance, Artigues *et al.* (2009) and Guyon *et al.* (2012) consider integrated employee-timetable and job shop scheduling problems, in which human operators with different skills have to be assigned to machines and shifts in a job shop production line. In these problems, the objective is minimizing the total cost of the assignments. Ruiz-Torres *et al.* (2012) model a real JSP in pharmaceutical industry; the resources are technicians with different skills and only a subset of them can be working in parallel.

In this paper we deal with the $JSO(n, p)$ with the objective of minimizing the total flow time. This objective function is often more interesting than the makespan in real environments (Brucker and Knust 2006) and at the same time it makes scheduling problems harder (González *et al.* 2010). As it is pointed out by Framinan and Leisten (2003), total flow time is one of the most important performance measures, as it can lead to a stable utilization of resources, rapid completion of some jobs and so minimizing the work-in-progress management cost. Therefore it is a very important objective in many environments, for instance, electronics, chemicals, textile, food and service industries (Lin *et al.* 2012).

To solve the $JSO(n, p)$ problem we propose a hybrid algorithm that interleaves best-first and depth-first searches. The objective is exploiting the advantages of these heuristic search algorithms and avoiding some of their inconveniences at the same time. Both search strategies are exhaustive so that they may eventually come up with a proven optimal solution. Best-first diversifies the search as it keeps open many paths towards candidate solutions and so it requires an exponential amount of memory. For this reason, it often fails to reach a solution due to the memory getting exhausted; nevertheless it may return a tight lower bound when a solution is not reached. On the other hand, depth-first intensifies the search on a particular region of the search space and so it can take a very long time to reach an optimal or near optimal solution.

We propose to combine best-first and depth-first search so that the second is issued from some of the search states selected by the first to be expanded next. In this way, we expect to achieve a proper balance between diversification and intensification so that the combined algorithm comes eventually with near optimal solutions and tight lower bounds at the same time. The equilibrium between search intensification and diversification effort is the key for success in other search algorithms such as hybrid metaheuristics (Talbi 2009, Glover and Laguna 1993), where a global searcher such as an evolutionary algorithm is often combined with some local searcher such as tabu search or path relinking to obtain outstanding algorithms (Vela *et al.* 2010, González *et al.* 2012). One advantage of the proposed combination is that it may reach near optimal solutions and tight lower bounds, and therefore it may certify the optimality of solutions at difference of hybrid metaheuristics.

The algorithm is guided by two heuristic estimations, which rely on two problem relaxations, and it is enhanced with some global pruning rules. The use of these rules require bookkeeping expanded states in order for each new expanded state to be compared with some of the states expanded previously for some dominance relation. As we will see in the experimental study, the pruning method is a key component of the proposed algorithm. In spite of the fact that it consumes memory, it allows the algorithm to reduce drastically the effective search space.

To assess the performance of the algorithm and to compare it with other method, we have conducted an experimental study in which we analyzed the contribution of each component to the algorithm performance and compared the algorithm with an implementation on IBM CPLEX CP Optimizer (CP). This is a commercial solver embedding constraint propagation rules and local search methods dedicated to scheduling (Laborie 2009, IBM 2009) and it is often used to compare with other approaches to scheduling problems (Gacias *et al.* 2010). The results of this study show that the hybrid algorithm is efficient thanks to the proper combination of its components and that it compares favorably with CP.

The remaining of the paper is organized as follows. In Section 2 we define the problem and propose a disjunctive model for it. Then, in Section 3, we give some background on heuristic search and describe the proposed hybrid algorithm termed A*DFS. In Section 4 we show how the proposed hybrid algorithm is adapted for the $JSO(n, p)$. The design and results of the experimental study are reported in Section 5. Finally, the main conclusions of the paper and some guidelines for further research are given in Section 6.

2. The job shop scheduling problem with operators

Formally the JSP with operators can be defined as follows. We are given a set of n jobs $\{J_1, \ldots, J_n\}$, a set of m resources or machines $\{R_1, \ldots, R_m\}$ and a set of p operators $\{O_1, \ldots, O_p\}$. Each job J_i consists of a sequence of v_i operations or tasks $(\theta_{i1}, \ldots, \theta_{iv_i})$. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, an integer duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ and an assisting operator $O_{\theta_{il}}$ to be determined. A feasible schedule is a complete assignment of starting times and operators to operations that satisfies the following constraints: (i) the operations of each job are sequentially scheduled, (ii) each machine can process at most one operation at any time, (iii) no preemption is allowed and (iv) each operation is assisted by one operator and one operator cannot assist more than one operation at the same time. The objective is to find a feasible schedule that minimizes the sum of the completion times of all jobs, i.e. the total flow time. This problem was first defined by Agnetis *et al.* (2011) for makespan minimization and is denoted as $JSO(n, p)$.

The significant cases of this problem are those with $p < min(n, m)$, otherwise the problem is a standard job-shop problem denoted as $J||\Sigma C_i$.

Scheduling problems are usually represented by means of a disjunctive model. We propose here to use a model for the $JSO(n, p)$ that is similar to that used by Agnetis *et al.* (2011). Figure 1 shows a solution graph for an instance with 3 jobs, 3 machines and 2 operators. In addition to the nodes that represent operations and the dummy nodes *start* and end_i, for each job J_i, we introduce nodes $O_j^{start}, 1 \leq j \leq p$, to represent operators in this model. So, we have three main types of arcs: job, machine and operator arcs. Each type defines the sequence of operations in the same job, machine or operator respectively. Operator nodes are linked to the first operation assisted by the operator. Also, there are arcs from the *start* node to each operator node and from the last operation of each job J_i to the corresponding end_i node. A critical path for job J_i is a longest weighted path from node *start* to node end_i. The total flow time of the solution is obtained by adding the cost of a critical path for each job J_i.

In Figure 1, discontinuous arrows represent operator arcs. So, the sequences of operations assisted by operators O_1 and O_2 are $(\theta_{21}, \theta_{11}, \theta_{32}, \theta_{12}, \theta_{13})$ and $(\theta_{31}, \theta_{22}, \theta_{23}, \theta_{33})$, respectively. In order to simplify the graph drawing, only the operator arc is drawn when there are two arcs between the same pair of nodes. Continuous arrows represent job arcs and dotted arrows represent machine arcs. In this example, the costs of the critical paths for jobs J_1, J_2 and J_3 are 13, 10 and 14 respectively, so the total flow time of the schedule is 37.

Figure 2 shows a Gantt chart for the schedule represented by Figure 1. As we can see, all the constraints are satisfied and, as there are only two operators available, no more than two operations are ever processed in parallel.

Figure 1 should appear here

Figure 2 should appear here

3. The hybrid heuristic search algorithm A*DFS

In this section we start with a review of the basic notions about state space heuristic search and then introduce the hybrid algorithm proposed in this paper.

3.1. *Heuristic search*

A search problem is given by a quintuple $\langle \mathbf{S}, \Gamma, s, SUC, c \rangle$, where $\mathbf{S}$ is the set of states or nodes, $s \in \mathbf{S}$ is the initial state, $\Gamma \subseteq \mathbf{S}$ is the set of goals, SUC is the transition operator such that for a state $n \in \mathbf{S}$, $SUC(n)$ returns the set of successor states of n, and each transition from n to $n' \in SUC(n)$ has a non negative cost $c(n, n')$. The states and the transitions between states conform a graph, which is called *search space*. We consider here the simple case where the search space is a tree, as this structure of the search space naturally arises in a variety of combinatorial optimization problems, especially scheduling and manufacturing problems. $P_{s\text{-}n}$ denotes the path from s to n and $g(n)$ the cost of this path.

Starting from s and applying systematically the operator SUC, a search algorithm looks for an optimal solution[1], i.e. a path $P_{s\text{-}o}$ with $o = \arg\min\{g_{P_{s\text{-}o'}} | o' \in \Gamma\}$, the cost of this solution is denoted C^*. For this purpose, a tree search algorithm maintains the list OPEN to store those states that have been generated but not yet expanded. This list represents the frontier between the explored and unexplored regions of the search space. At each step, the search algorithm selects for expansion the first state from the list OPEN. So, the criterion used to sort the nodes in OPEN, namely the *search strategy*, has implications in the completeness, admissibility and efficiency of the algorithm.

The performance of a search algorithm can be dramatically improved by exploiting the knowledge from the problem domain. This knowledge may be used for guiding the search towards promising regions of the search space, so reaching a goal state without the need of exploring every path in the search tree. This can be done by means of a heuristic evaluation function f, such that for each state n, $f(n)$ gives a measure of its promise. The properties of this function and the way it is used determine the properties of the search algorithm.

3.1.1. Best-first search

The heuristic knowledge represented by f can be fully exploited if the nodes in OPEN are sorted by non-decreasing f-values. This search strategy is called *best-first search.* The function f is usually defined such that $f(n)$ is an estimation of $f^*(n)$, which is the cost of the best solution reachable from n, i.e. $f^*(n) = g(n) + h^*(n)$, where $h^*(n)$ is the optimal cost from n to its nearest goal. This way, the evaluation function can be defined as $f(n) = g(n) + h(n)$ for all states n, where $h(n)$ is a heuristic estimation for $h^*(n)$. In this case, we have the A* algorithm proposed by Hart *et al.* (1968) and very well described by Nilsson (1980) and Pearl (1984).

The A* algorithm has some interesting properties that depend on the function h. When $h(n) \leq h^*(n)$ for all states n, the algorithm is admissible and then it guarantees that the first goal reached represents an optimal solution. Moreover, if we have two admissible heuristics h_1 and h_2 such that $h_1(n) < h_2(n)$ for all non goal states n, h_2 is more informed than h_1 and then the number of nodes expanded by A* to reach a solution with h_2 is not greater than it is with h_1.

3.1.2. Depth-first search

The main inconvenience of best-first search algorithms is the large amount of space they require for the list OPEN what, in practice, may get the memory of the target machine exhausted before reaching a solution even for small or medium-size instances. For this reason a common alternative is to use *depth-first search* (DFS). In this case the OPEN list is managed as a LIFO structure, i.e. the successors of the expanded state n are inserted at the beginning of OPEN. If these nodes are locally sorted by their f-values, then the search strategy is termed partially-informed depth-first search (Pearl 1984, Mencía *et al.* 2010, 2012b). This strategy only requires a linear consumption of memory. It usually finds a suboptimal solution quickly and then a sequence of improving solutions, i.e., it is an any-time search strategy. At the same time, nodes n such that $f(n) \geq UB$, UB being the cost of the best solution found so far, i.e. the incumbent, can be pruned as they do not lead to an improving solution. This pruning is a key component of *branch and bound* algorithms. In the long term DFS can come up with an optimal solution after the whole search space is explored. However, it often gets stuck in low

[1]Without loss of generality, we consider minimization problems in this section.

promising areas of the search space due to the local heuristic guidance.

3.2. *The hybrid algorithm: A*DFS*

Our proposal here is to combine both strategies into a hybrid algorithm, termed A*DFS, to exploit the exploration capability of best-first search and the intensification capability of depth-first search. Algorithm **1** provides a pseudo-code for A*DFS, and Algorithm **2** shows the DFS component. The hybrid algorithm consists of two consecutive phases: The first one starts in the beginning and completes when a memory limit is reached. In this phase, the algorithm alternates between best-first (A*) and depth-first searches. Concretely, it starts in A* mode; then, after every N expanded nodes, a limited depth-first search is issued from the next selected node n just before being expanded by A* (Algorithm **1**, lines 15-22). The depth-first search from n finishes after expanding E states and the algorithm turns again into A* mode expanding n. When the memory limit is reached, A*DFS shifts to a second phase, in which an exhaustive depth-first search (without limiting the number of expansions) is carried out from each state that was generated but not expanded by A*. In both phases, the search is guided by f-values and when a state n is generated such that $f(n) \geq UB$, n is pruned.

Combining A* and depth-first search has given promising results for solving shortest-path problems[1], in which the search space is a graph with cycles and the solutions are located at unknown and unbounded depths. Stern *et al.* (2010) propose a combination in which an depth-first search limited on a maximum depth is yielded from each node to be expanded by an A* algorithm. This approach lets the algorithm save a lot of memory thanks to the depth-first search component and so it can solve more problem instances. However, limiting the depth of the depth-first searches is not appropriate for a search tree where the solutions are located at a known depth. This is the case of many scheduling problems, including the one we face in this paper.

3.2.1. Phase 1. Interleaving A with limited depth-first searches*

In the first phase, the integration of A* with DFS is done in the following way. When running in A* mode, the algorithm keeps an OPEN-A* list with nodes sorted by f-values. However, each time the algorithm turns into DFS mode, a local OPEN list is created. This list is initiated with n and it is finally erased after DFS completes E expansions. At this time, the algorithm turns into A* mode for the next N expansions, being n the first node expanded in this step as it is the first one in OPEN-A*. This search strategy produces some node reexpansions, but at the same time it helps to prevent the algorithm from running out of memory too early, and therefore it can issue more intensification steps in different regions of the search space.

The parameters N and E are relevant in order to reach a proper balance between exploration and exploitation. A very small value of N may lead the algorithm to sample quite similar regions of the search space, so achieving little diversification. On the contrary, a very large N, could lead to a very small number of intensification steps, hence reducing the ability of the algorithm to sample different regions. Clearly, the value of E must be at least the distance from the node n to a leaf node, but it should be larger than this to explore more than one solution in the region under the node n. However, a very large E may require the algorithm to spend too much time in each intensification.

[1]Some examples are puzzles or planning problems.

Finally, as it is common in metaheuristics, we try to adapt the intensification effort

Algorithm 1 A*DFS

```
Require: $start$, $N$, $E$, $memoryLimit$, $timeLimit$.
Ensure: Lower ($LB$) and upper ($UB$) bounds on the optimal solution. A solution
    ($incumbentSolution$) with cost $UB$ (if exists and is reached by $timeLimit$).
    Phase 1: Interleaving A* with limited Depth First Search
 1: $LB \leftarrow f(start)$;
 2: $UB \leftarrow \infty$;
 3: $incumbentSolution \leftarrow null$;
 4: $OPEN\_A^* \leftarrow \{start\}$;
 5: $\#expandedNodes \leftarrow 0$;
 6: while not ($empty(OPEN\_A^*)$) and ($LB < UB$) and not ($memoryLimit$ or
    $timeLimit$ reached) do
 7:   $n \leftarrow pop(OPEN\_A^*)$;
 8:   $LB \leftarrow f(n)$;
 9:   if $isGoal(n)$ then
10:     $UB \leftarrow f(n)$;
11:     Update $incumbentSolution$ from $n$;
12:     return $incumbentSolution$, $LB$, $UB$; {optimal}
13:   end if
14:   {Every $N$ node expansions a limited DFS is issued}
15:   if ($\#expandedNodes$ mod $N$) = 0 then
16:     $(newSolution, LB_{DFS}, UB_{DFS}) \leftarrow DFS(n, E, UB, timeLimit)$;
17:     $LB \leftarrow min(LB_{DFS}, min\{f(q) \mid q \in OPEN\_A^*\})$;
18:     if $UB_{DFS} < UB$ then
19:       $incumbentSolution \leftarrow newSolution$;
20:       $UB \leftarrow UB_{DFS}$;
21:     end if
22:   end if
23:   if $f(n) < UB$ then
24:     $S \leftarrow SUC(n)$;
25:     $OPEN\_A^* \leftarrow merge(S, OPEN\_A^*)$; {According to $f$-values}
26:     $\#expandedNodes \leftarrow \#expandedNodes + 1$;
27:   end if
28: end while
    Phase 2: Finishing with exhaustive depth-first search
29: if $memoryLimit$ reached then
30:   while not ($empty(OPEN\_A^*)$) and ($LB < UB$) and not ($timeLimit$ reached)
      do
31:     $n \leftarrow pop(OPEN\_A^*)$;
32:     $(newSolution, LB_{DFS}, UB_{DFS}) \leftarrow DFS(n, \infty, UB, timeLimit)$;
33:     $LB \leftarrow min(LB_{DFS}, min\{f(q) \mid q \in OPEN\_A^*\})$;
34:     if $UB_{DFS} < UB$ then
35:       $incumbentSolution \leftarrow newSolution$;
36:       $UB \leftarrow UB_{DFS}$;
37:     end if
38:   end while
39: end if
40: return $incumbentSolution$, $LB$, $UB$;
```

of the depth-first searches to the promising degree of the search tree under node n. To do that, we restart the counter of expansions at 0 each time the algorithm reaches an improving solution when running in DFS mode, so keeping the search in this mode for at least E more expansions (Algorithm **2**, lines 9-13).

3.2.2. Phase 2. Finishing with exhaustive depth-first search

Once the memory limit is reached, A*DFS turns to the second phase. Here, a depth-first search is performed from each node stored in the OPEN-A* list. As this list is sorted by non-decreasing f-values, DFS traverses first the subtrees under the most promising nodes generated in the previous phase in A* mode, so it is more likely to find good solutions and to improve the lower bounds soon. So, DFS operates on the OPEN-A* list and in this way, A*DFS runs always safe from memory getting exhausted.

The algorithm completes when either OPEN-A* gets empty or the minimum f-value in OPEN-A* is not lower than the cost of incumbent solution.

3.2.3. Discussion

The proposed combination of best-first and depth-first search is expected to take profit from the benefits of both strategies and overcome some of their drawbacks.

Firstly, the use of A* in the first phase of A*DFS provides global guidance towards

Algorithm 2 DFS

```
Require: $state$, $E$, $UB$, $timeLimit$.
Ensure: Lower ($LB_{DFS}$) and upper ($UB_{DFS}$) bounds in the subtree rooted with $state$.
    A solution ($newSolution$) of the subtree if $UB_{DFS} < UB$.
 1: $LB_{DFS} \leftarrow f(state)$;
 2: $UB_{DFS} \leftarrow \infty$;
 3: $newSolution \leftarrow null$;
 4: $OPEN \leftarrow \{state\}$;
 5: $\#expDFS \leftarrow 0$;
 6: while ($not\_empty(OPEN\_A^*)$) and ($\#expDFS < E$) and (not $timeLimit$ reached) do
 7:   $n \leftarrow pop(OPEN)$;
 8:   if $f(n) < min(UB, UB_{DFS})$ then
 9:     if $isGoal(n)$ then
10:       $UB_{DFS} \leftarrow f(n)$;
11:       Update $newSolution$ from $n$;
12:       $\#expDFS \leftarrow 0$;
13:     end if
14:     $S \leftarrow SUC(n)$ {$S$ sorted by non-decreasing $f$-values};
15:     $OPEN \leftarrow append(S, OPEN)$ ;
16:     $\#expDFS \leftarrow \#expDFS + 1$;
17:   end if
18: end while
19: if $empty(OPEN)$ then
20:   $LB_{DFS} \leftarrow min(UB, UB_{DFS})$;
21: else
22:   $LB_{DFS} \leftarrow min\{f(q) \mid q \in OPEN\}$;
23: end if
24: return $newSolution$, $LB_{DFS}$, $UB_{DFS}$;
```

promising regions in the search space and allows the algorithm to improve the lower bounds over time. Then DFS intensifies the search on some of these regions without getting stuck in any of them due to the limited number of expansions; hopefully, in these searches new solutions improving the incumbent may be found. Also, developing exhaustive DFS from the nodes in OPEN-A* in the second phase of the algorithm makes A*DFS able to run for a long time without increasing the memory consumption and to explore the most promising subtrees first, so improving both lower and upper bounds.

Regarding the formal properties of A*DFS, it is clear that when a new solution is reached in DFS mode, this solution may not be optimal. However, if the heuristic function h is admissible, a solution is optimal if it is reached when running in A* mode (Algorithm **1**, lines 9-13). This is a consequence of OPEN-A* being sorted by non-decreasing f-values.

Also, at any time, the lowest f-value in OPEN-A* and OPEN lists is a lower bound on the optimal solution. Here, it is worth doing some remarks regarding the computation of lower bounds. As we can observe in Algorithm **2**, DFS searches for solutions with cost less than UB in the subtree under the node *state*, and returns a lower bound on the cost of the best solution in that subtree. So, if DFS is able to traverse the whole subtree without finding any improving solution, UB is a lower bound as it is proven that no solution with less cost exists from *state*. On the other hand, if the subtree is completely explored finding an improving solution, its cost $UB_{DFS} < UB$ will be a lower bound, as it is the best solution reachable from *state*. In other cases, the lower bound is defined by the least f-value in OPEN. These cases are shown in lines 19-23. A*DFS takes profit from the lower bounds returned by DFS, as depicted in lines 17 and 33 of Algorithm **1**. In these cases, the least f-value in OPEN-A* will be given by the first state, so it can be computed very efficiently.

The expected behavior of A*DFS is to return a sequence of improving solutions, as it DFS does, but this sequence having a larger rate of improvement than that of the sequence returned by single DFS. Also, it is expected to return lower bounds as accurate as A*, or even better, and to certify more optimal solutions as it can run indefinitely.

4. A*DFS for $JSO(n,p)$ with total flow time minimization

In this section we set out the main components of the proposed algorithm, namely the search space where the algorithm looks for solutions, the heuristic estimation used to guide the search and also some global pruning rules which help to reduce the effective search space.

4.1. *The search space: OG&T schedule generation scheme*

We define a search space which is derived from the $OG\&T$ algorithm proposed by Sierra *et al.* (2011b). This is a schedule generation scheme for the $JSO(n,p)$ which is an extension of the well-known $G\&T$ algorithm proposed by Giffler and Thompson (1960) for the classic job shop scheduling problem. The operations are scheduled one at a time following a sequence of non-deterministic choices. When an operation u is scheduled, its preceding operation in the job sequence, denoted PJ_u, was already scheduled if this operation exists. At this time, u is assigned a starting time st_u and an operator O_i, $1 \leq i \leq p$.

In order to select the candidate operations to be scheduled next, we start considering a naive strategy to establish an initial set of candidates which is subsequently restricted by means of some local pruning rules. Let SC be the set of scheduled operations at an arbitrary time. Then, the next non-deterministic choice may be any operation of the set A defined as

$$A = \{v \notin SC, \nexists PJ_v \vee (PJ_v \in SC)\} \tag{1}$$

i.e., the set that includes the first unscheduled operation of each job that has at least one unscheduled operation. If the operation u in A is selected, the starting time of u is given by its head r_u which is calculated as

$$r_u = \max\{r_{PJ_u} + p_{PJ_u}, r_v + p_v, \min_{1 \le i \le p} t_i\} \tag{2}$$

where t_i, $1 \le i \le p$, is the time at which the operator O_i is available and v denotes the last operation scheduled having $R_v = R_u$. At the same time, the operator O_i that is available at the latest time before r_u, i.e.

$$i = \arg\max\{t_j; t_j \le r_u; 1 \le j \le p\} \tag{3}$$

is assigned to assist the operation u. Let v^* be the operation in A having the earliest completion time if it were scheduled next, i.e.

$$v^* = \arg\min\{r_u + p_u; u \in A\}. \tag{4}$$

The set of non-deterministic choices may be reduced to the subset $A' \subseteq A$

$$A' = \{u \in A; r_u < r_{v^*} + p_{v^*}\} \tag{5}$$

Moreover, the set of choices can be further restricted in the following way. Let $\tau_0 < \cdots < \tau_k$ be the sequence of all times along the interval $[\min\{r_u; u \in A'\}, r_{v^*} + p_{v^*})$, where each τ_i is given by the head of some operation in A' or the time at which some operator becomes available. Let p'_i be the number of operators available in the subinterval $[\tau_i, \tau_{i+1})$ and let m'_i be the number of different machines that are required by the operations in A' which may be processed along this subinterval. Then, A' may be reduced as long as the following operations are maintained:

(i) The operations requiring the same machine as v^*.
(ii) For each interval $[\tau_i, \tau_{i+1})$ with $m'_i > p'_i$, the operations required by at least $m'_i - p'_i$ machines.

The set of operations obtained in this way is termed B and it is clear that $|B| \le |A'| \le |A|$. An important property of this schedule generation scheme is that if the number of operators is large enough, in particular if $p \ge \min(n, m)$ so that $JSO(n, p)$ becomes $J||\Sigma C_i$, it is equivalent to the $G\&T$ algorithm. Sierra *et al.* (2011b) give a full description of the $OG\&T$ algorithm together with a formal proof of its dominance property, i.e. the search space contains at least one optimal solution.

So, in accordance with the above, a state is a partial schedule. In the initial state all the operations are unscheduled and the remaining states correspond to each one of the situations that may be generated by the algorithm *OG&T* after one of the operations of the set B is scheduled. Figure 3 shows a partial schedule that represents a feasible state to a problem with 5 jobs, 5 machines and 3 operators. In accordance with the *OG&T* algorithm, in this situation $A = \{\theta_{12}, \theta_{22}, \theta_{32}, \theta_{42}, \theta_{51}\}$, $A' = \{\theta_{22}, \theta_{32}, \theta_{42}, \theta_{51}\}$ and $B = \{\theta_{32}, \theta_{42}, \theta_{51}\}$ (if R_2 and R_1 are chosen from the interval $[\tau_1, C)$). Therefore, this state has 3 successors as a result of scheduling the operations θ_{32}, θ_{42} and θ_{51} respectively. For a state n, $g(n)$ is the total flow time of the partial schedule, i.e. the summation of the completion times of the last operation scheduled in each job, and consequently the cost $c(n, n')$ from n to a successor n' is the variation of this value from n to n'. So, for the state n in Figure 3, $g(n) = 19$ and if n' is the result of scheduling θ_{32} from n, then $c(n, n') = 7$. The goals are those states having all the operations scheduled.

Figure 3 should appear here

4.2. *Heuristic functions*

The evaluation function is $f(s) = g(s) + h(s)$, where $g(s)$ denotes the total flow time accumulated in the state s, and $h(s)$ is a heuristic function that estimates the additional cost required to reach a solution from s. We consider two admissible heuristics derived from problem relaxations, termed h_{PS} and h_{OP} respectively and finally we take $h(s) = \max(h_{PS}(s), h_{OP}(s))$.

4.2.1. Heuristic h_{PS}

The first one, termed h_{PS}, is borrowed from Sierra and Varela (2010) where the problem $J||\Sigma C_i$ is considered: it relies on relaxing non-preemption and operator constraints, and the capacity constraints for all but one of the machines. The optimal solution to this relaxed problem, denoted as $f_{PS}(s)$, is a lower bound on $f^*(s)$. So, we compute $h_{PS}(s) = f_{PS}(s) - g(s)$.

4.2.2. Heuristic h_{OP}

The second heuristic is obtained from relaxing constraints other than operator's. We start relaxing the heads and the capacity constraints of the machines for the unscheduled operations, with the only exception of the head of the first unscheduled operation in each job. So, in the relaxed problem the operators play the role of identical parallel machines available at different times and the unscheduled operations of each job join into one only operation whose release time is the head of the first unscheduled operation of the job. This problem is denoted $(P, NC_{inc}|r_i|\sum C_i)$. A simplification of this problem where all release times are equal, denoted $(P, NC_{inc}||\sum C_i)$, can be optimally solved applying the SPT (Shortest Processing Time) rule (Adiri *et al.* 1989, Schmidt 2000). At the same time, the one machine sequencing problem with release times (heads) and total flow time minimization, denoted $(1|r_i|\sum C_i)$ is NP-hard (Graham *et al.* 1979) and its preemptive version $(1|r_i, pmtn|\sum C_i)$ can be solved in polynomial time by an extension of the SPT rule. However, the problem $(P, |r_i, pmtn|\sum C_i)$ is NP-complete as proved by Baptiste *et al.* (2007). Therefore, the problem $(P, NC_{inc}|r_i, pmtn|\sum C_i)$ is NP-complete as well.

From the results above, we decided relaxing the release times as well to obtain a polynomially solvable problem, $(P, NC_{inc}||\sum C_i)$. Algorithm **1** shows the calculation of heuristic h_{OP} for a state s.

Algorithm 1 Calculating the heuristic h_{OP} for a state s

```
Require: A state $s$.
Ensure: The heuristic estimation $h_{OP}(s)$.
  Build a $(P, NC_{inc}||\sum C_i)$ instance $\mathbf{P}$ relaxing the problem represented by the state $s$
  as it is indicated in the text;
  while not all operations in $\mathbf{P}$ are scheduled do
    Let $m$ be the first machine (operator) available in the current partial schedule;
    Let $t$ be the time at which $m$ gets available;
    Select the unscheduled operation $\theta$ in $\mathbf{P}$ with the shortest processing time;
    Schedule the operation $\theta$ at time $t$ on the machine $m$;
  end while
  return The total flow time of the built schedule for $\mathbf{P}$- $g(s)$;
```

4.3. *Dominance relations*

The effective search tree may be reduced by means of dominance relations among states similar to that exploited by Sierra and Varela (2010) for the classic job shop scheduling problem. This relation is defined in accordance with the formal definition given in Ibaraki (1977) for dominance relations that guarantees a single optimal solution. Given two search states s_1 and s_2, s_1 dominates s_2 iff $f^*(s_1) \leq f^*(s_2)$. In general, dominance relations cannot be easily established, but in some particular cases an effective condition for dominance can be defined. For the above search space, an efficient and effective dominance rule is defined as follows. If s_1 and s_2 are states having the same operations scheduled, SC, then s_1 dominates s_2 if the following three conditions hold:

(1) $r_v(s_1) \leq r_v(s_2)$, for all $v \notin SC$.
(2) $\sum_{\theta_{iv_i} \in SC} r_{\theta_{iv_i}}(s_1) \leq \sum_{\theta_{iv_i} \in SC} r_{\theta_{iv_i}}(s_2)$.
(3) $av(s_1) \geq av(s_2)$.

where $r_v(s)$ and $av(s)$ denote the head of v and the availability of operators in state s, respectively. It must be taken into account that θ_{iv_i} is the last operation of job J_i.

From conditions (1) and (3) it follows that the subproblem represented by the unscheduled operations SC is less costly for state s_1 than it is for s_2 and condition (2) means that the accumulated flow time due to the jobs with all their operations scheduled is not greater in s_1 than it is in s_2. The availability of operators in a state can be evaluated as follows. Let $t_1 \leq \cdots \leq t_p$ be the times at which the operators get idle in the state s (here it is worth noting that the operator available at time t_i is any O_j, $1 \leq j \leq p$). If u^* is the unscheduled operation with the lowest head in s, then none of the operators can get busy again before r_{u^*}, so we can consider that the operators are actually available for the unscheduled operations at times $t'_1 \leq \cdots \leq t'_p$, where $t'_i = max(r_{u^*}, t_i)$. So, the availability of operators in state s is defined as the ordered vector $av(s) = (t'_1, \ldots, t'_p)$. At the same time, if x is the number of jobs and y is the number of machines with unscheduled operations in SC, then the maximum number of operators required to schedule the remaining operations in these states is limited by $p' = min(p, x, y)$, so $av(s_1) \geq av(s_2)$ iff $t'_{1i} \leq t'_{2i}, 1 \leq i \leq p'$.

The implementation of the dominance rules can be done as follows. When a state s is considered for expansion, s is compared to all the expanded states having the same operations scheduled. This can be done efficiently if the expanded states are stored in a CLOSED list implemented as a hash table where the key values are bit-vectors representing the scheduled operations. Moreover, this rule may be improved from the following result that establishes a sufficient condition for the conditions (1), (2) and (3) not to hold simultaneously.

Proposition 4.1: *If the heuristic estimation is obtained from a problem relaxation, i.e. $f(s)$ is the cost of an optimal solution to the relaxed problem obtained from s in accordance with that problem relaxation, and the states s_1 and s_2 fulfill all conditions (1), (2) and (3) then $f(s_1) \leq f(s_2)$.*

Proof: *It is trivial, as any solution to the relaxed instance obtained from s_2 is a solution to the relaxed instance obtained from s_1.* □

So, when a state s is expanded, it has only to be compared with states s' in CLOSED having the same operations scheduled and $f(s') \leq f(s)$.

As both heuristics h_{PS} and h_{OP} are obtained from problem relaxations, the evaluation functions defined as $f_{PS}(s) = g(s) + h_{PS}(s)$ and $f_{OP}(s) = g(s) + h_{OP}(s)$ fulfill the condition of Proposition 4.1. As $f(s) = \max(f_{PS}(s), f_{OP}(s))$, the condition may be evaluated on f, due to the fact that $f(s_1) > f(s_2)$ implies that at least one of the conditions $f_{PS}(s_1) > f_{PS}(s_2)$ or $f_{OP}(s_1) > f_{OP}(s_2)$ holds.

To demonstrate the application of this rule we can consider a search state similar to that of Figure 3 with the same operations scheduled, but exchanging the order of operations θ_{31} and θ_{41} on the machine R_1. These states dominate each other, so one of them can be discarded. In this example the heads of the unscheduled operations and the operators availability are the same in both states. However, other situations might appear where these values are not the same in two states while one of them dominates the other.

In order to save memory, the proposed A*DFS algorithm only stores nodes in the CLOSED list when it is running in A* mode. In DFS mode it only checks if the state to be expanded is dominated by a state in CLOSED.

5. Computational results

In this section we report the results from an experimental study carried out to evaluate the components of A*DFS and to compare it with other methods. We have experimented across a number of instances adapted from the classic JSP considering different numbers of operators. The target machine was Intel Xeon (2,26 GHz), 24 GB RAM. and all the algorithms were implemented in C++. In these experiments, we have set A*DFS parameters $N = 100$ and E at twice the distance from n to the leaves. The algorithms were evaluated at two time limits, 300s and 3600s. Also, A*, DFS and A*DFS have been given a memory limit of 4 GB.

5.1. *Evaluating the heuristics and the pruning method*

We start with some experiments to analyze the behavior of the heuristic estimations h_{PS} and h_{OP} separately and in combination and the efficiency of the pruning method by dominance rules. To do that, we chose a small instance such as $FT06$, of size $n \times m = 6 \times 6$, which can be solved to optimality for almost any combination of number of operators, heuristic and pruning option. Figure 4 shows the number of expanded states (in logarithmic scale) required to certify the optimal solution in each case. We established a time limit of 3600s. Even though most of the instances were solved in much less time, two of them were not solved: the instances with 1 and 2 operators using h_{PS} and no pruning. Moreover, these instances were not solved by 24 hours after expanding about 350 million states. It is remarkable that these instances get solved by 36s and 487s, respectively, with the same heuristic and pruning. In these two cases, the number of expanded states after 3600s was considered.

We can observe that h_{OP} is the best option for small values of p while h_{PS} is the best one for large values, independently of the pruning option. Clearly, the combined heuristic is the best option overall. At the same time, it is clear that the pruning method is quite effective. In almost all the cases, the number of expanded states is much lower with this option. In particular, for the instances with 3 and 4 operators, which seem to be the most difficult to solve, and the combined heuristic, the number of expanded states decreases in one order of magnitude thanks to the pruning method. From these results, we considered the combined heuristic and the pruning method in the remaining experiments.

Figure 4 should appear here

5.2. *Comparison with other algorithms*

As, to our knowledge, there is no other approach to the $JSO(n, p)$ problem with total flow time minimization, we opted to compare A*DFS with its two component algorithms A* and DFS[1], and also with an implementation on a current solver specialized in scheduling as it is IBM ILOG CPLEX CP Optimizer (IBM 2009). When DFS is used alone, we exploit the pruning rule in the same way as in A*, i.e. storing the expanded nodes in CLOSED as long as the memory limit is not reached. In the CP implementation, $JSO(n, p)$ is modeled like a classic job shop scheduling problem where the p operators are naturally modeled as a nonrenewable cumulative resource of capacity p. The solver was set to exploit constraint propagation on no overlap (*NoOverlap*) and cumulative function (*CumulFunction*) constraints to extended level. The search strategy used was depth-first search with restarts (default configuration).

In the experiments, we considered a large benchmark defined from the well-known LA01-40 instances (Beasley 1990) with $n \times m \in \{10 \times 5, 15 \times 5, 20 \times 5, 10 \times 10, 15 \times 10, 20 \times 10, 30 \times 10, 15 \times 15\}$; each subset having 5 instances. For each one of these instances, the number of operators p varies from 1 to $min(n, m)$, so we have 350 instances in all. Every instance was solved with each method taking two different time limits: 300s and 3600s respectively. The lower bound and the solution reached (upper bound) was recorded in each run.

5.2.1. *Upper bounds*

The errors in percentage terms (%Error) of each solution w.r.t. the best lower bound obtained in our experiments are summarized in Figures 5 and 6 and Tables 1 and 2. Figures 5 and 6 show the results from the 4 algorithms for each group of instances with the same size at 300s and 3600s, respectively, averaged for all instances with the same number of operators. The first remarkable result of these experiments is that for all combinations of n, m and p, A*DFS is always better than both A* and DFS alone independently of the time limit. Also, A*DFS is better than CP in most of the cases.

Figure 5 should appear here

[1]Some preliminary results from A* and DFS have been discussed in Sierra *et al.* (2011a) and Mencía *et al.* (2012a) respectively.

Figure 6 should appear here

Tables 1 and 2 summarize the results for each group of instances with the same size and time limits 300s and 3600s, respectively, and show the number of instances for which the algorithms are able to certify the optimality of the solution (#Sol.). As we can observe, the results of A* are the same at both time limits as it consumes the whole memory before 300s, while the results from DFS are slightly better at 3600s than they are at 300s. Nevertheless, A*DFS is able to improve substantially the quality of the solutions and also to certify the optimality of more of them when it is given more time. CP is the algorithm that layouts the largest difference in the quality of solutions from 300s to 3600s, but even at 3600s CP has larger average error than A*DFS at 300s. Moreover, even at 3600s CP is unable to certify the optimality of any solution.

In order to enhance the conclusions of the experimental comparison between A*DFS and CP we have conducted some statistical analysis following García *et al.* (2010). Particularly, we have used paired Wilcoxon tests samples. We have used as alternative hypothesis that the errors from a method M1 at a time T1 are smaller than those from a method M2 at time T2. So, we have analyzed several combinations of methods (A*DFS and CP) and time limits (300s and 3600s). The results of these tests are summarized in Table 3 where the p-values are indicated when each group of instances with the same size is taken as population sample. As we can observe, the null hypothesis is rejected at a high level of significance in all cases. So, these analysis confirm that A*DFS outperforms CP in this benchmark at both time limits. Furthermore, A*DFS at 300s is better than CP at 3600s.

Table 1 should appear here

Table 2 should appear here

Table 3 should appear here

5.2.2. Lower bounds

Table 4 reports the results of the lower bounds reached by A*, DFS and A*DFS at both time limits, 300s and 3600s. The values are errors in percentage terms , w.r.t. the best lower bounds reached in all the experiments, averaged for each group of instances with the same size. As we can observe, at 300s it is A* the algorithm that computes the best lower bounds in average. However, A*DFS at 3600s is the best one as it returns the best lower bound for every instance. So, we can conclude that not only A*DFS outperforms its two components A* and DFS in reaching upper bounds, but it also outperforms both A* and DFS in reaching lower bounds in the long term[1].

Table 4 should appear here

6. Conclusions

We have seen that by combining different search strategies as best-first search and depth-first search, it is possible to obtain a hybrid search algorithm that outperforms both

[1]Detailed results from all these experiments, including the best lower and upper bounds for each instance are provided in the supplementary material (www.di.uniovi.es/iscop, link Repository/Detailed Results from Papers).

strategies when each of them works separately. More concretely, we have proposed a new hybrid search strategy, termed A*DFS, that combines best-first and depth-first search in an original way. Firstly, it yields depth-first searches limited by a number of expansions from some of the states to be expanded by A*, hence diversifying the search and sampling different promising regions of the search space. Then, when a memory limit is reached, A*DFS yields exhaustive depth-first searches from the nodes generated but not yet expanded in the previous phase, which allows the algorithm to improve both lower and upper bounds without requiring additional memory resources.

We have applied this algorithm to solve the job shop scheduling problem with operators and total flow time minimization, which is very hard to solve and has interesting practical applications. To do so, we have exploited the schedule generation scheme proposed in (Sierra *et al.* 2011b), termed *OG&T*, defining a reduced search space. This algorithm also uses effectively some global pruning rules which rely on dominance relations among states and two heuristic estimations for search guidance and pruning.

The results from a comprehensive experimental study over a large benchmark set defined from well-known instances of the job shop scheduling problem shows that the hybrid algorithm is able to compute high quality solutions and lower bounds, and it clearly outperforms its two components: best-first and depth-first search. Furthermore, a comparison with IBM ILOG CPLEX CP Optimizer, a commercial solver specialized in scheduling problems, shows that the proposed algorithm is able to compute better solutions in 300s than CP Optimizer is in 3600s.

As future work we plan to improve A*DFS in various ways, for example by introducing restarts and randomization (Gomes *et al.* 1998) or other variants of depth-first such a limited discrepancy search (LDS) (Harvey and Ginsberg 1995) or the recent intensified iterative deepening A* (Mencía *et al.* 2013), in order to get better upper bounds.

Finally, we plan to refine the heuristics and the global pruning rules for the same problem and to apply a similar approach to other scheduling problems; in particular to scheduling problems with operators having different skills, as the one faced in Guyon *et al.* (2012).

References

Adiri, I., *et al.*, 1989. Single machine Flow-time scheduling with a single breakdown. *Acta Informatica*, 26, 679–696.

Agnetis, A., *et al.*, 2011. A job-shop problem with one additional resource type. *Journal of Scheduling*, 14 (3), 225–237.

Artigues, C., *et al.*, 2009. Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound. *Computers & Operations Research*, 36(8), 2330–2340.

Baptiste, P., *et al.*, 2007. The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, 10, 139146.

Beasley, J.E., 1990. OR-Library: Distributing Test Problems by Electronic Mail. *J Oper Res Soc*, 41 (11), 1069–1072.

Brucker, P. and Knust, S., 2006. *Complex Scheduling.* Springer.

Driebel, R. and Monch, L., 2012. An integrated scheduling and material-handling approach for complex job shops: a computational study. *International Journal of Production Research. DOI 10.1080/00207543.2011.639099.*

Framinan, J. and Leisten, R., 2003. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *Omega, International Journal of Management Science*, 31 (4), 311 – 317.

Gacias, B., Artigues, C., and Lopeza, P., 2010. Parallel machine scheduling with precedence constraints and setup times.. *Computers and Operations Research*, 37, 2141–2151.

García, S., *et al.*, 2010. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental Analysis of Power.. *Information Sciences*, 180, 2044–2064.

Giffler, B. and Thompson, G.L., 1960. Algorithms for Solving Production Scheduling Problems. *Operations Research*, 8, 487–503.

Glover, F. and Laguna, M., 1993. Tabu search. *In*: *Modern Heuristic Techniques for Combinatorial Problems.*, 70–150 Oxford: Blackwell Scientific Publishing.

Gomes, C.P., Selman, B., and Kautz, H.A., 1998. Boosting Combinatorial Search Through Randomization. *In*: *AAAI/IAAI*, 431–437.

González, M.A., *et al.*, 2012. An Efficient Hybrid Evolutionary Algorithm for Scheduling with Setup times and Weighted Tardiness Minimization. *Soft Computing*, DOI 10.1007/s00500-012-0880-y.

González, M.A., *et al.*, 2010. Tabu Search and Genetic Algorithm for Scheduling with Total Flow Time Minimization. *In*: *COPLAS 2010*, 33–41.

Graham, R., *et al.*, 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5, 287326.

Guyon, O., *et al.*, 2012. Solving an integrated job-shop problem with human resource constraints. *Annals of Operations Research*, On line, DOI 10.1007/s10479-012-1132-3.

Hart, P., Nilsson, N., and Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Science and Cybernetics*, 4(2), 100–107.

Harvey, W.D. and Ginsberg, M.L., 1995. Limited Discrepancy Search. *In*: *Proceedings of IJCAI 1995 (1)*, 607–615.

Ibaraki, T., 1977. The Power of Dominance Relations in Branch-and-Bound Algorithms. *Journal of the Association for Computing Machinery*, 24(2), 264–279.

IBM, 2009. Modeling with IBM ILOG CP Optimizer - Practical Scheduling Examples. *ftp://public.dhe.ibm.com/common/ssi/ecm/en/wsw14076usen/WSW14076USEN.PDF*.

Laborie, P., 2009. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. *In*: *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR09)*, 148–162.

Lin, S.W., *et al.*, 2012. Minimizig total flow time in permutation flowshop environment. *International Journal of Innovative Computing, Information and Control*, 8 (10(A)), 6599 – 6612.

Liu, C.H., Chen, L.S., and Lin, P.S., 2012. Lot streaming multiple jobs with values exponentially deteriorating over time in a job-shop environment. *International Journal of Production Research. DOI 10.1080/00207543.2012.657255.*

Meeran, S. and Morshed, M., 2012. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of Intelligent Manufacturing*, 1–16.

Mencía, C., *et al.*, 2012a. Combining Global Pruning Rules with Depth-First Search for the Job Shop Scheduling Problem with Operators. *In*: *19th RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with com-*

binatorial explosion.

Mencía, C., Sierra, M.R., and Varela, R., 2010. Partially Informed Depth First Search for the Job Shop Problem. *In*: *Proceedings of ICAPS'2010* AAAI Press, 113–120.

Mencía, C., Sierra, M.R., and Varela, R., 2012b. Depth-First Heuristic Search for the Job Shop Scheduling Problem. *Annals of Operations Research* Doi: 10.1007/s10479-012-1296-x.

Mencía, C., Sierra, M.R., and Varela, R., 2013. Intensified iterative deepening A* with application to job shop scheduling. *Journal of Intelligent Manufacturing* Doi: 10.1007/s10845-012-0726-6.

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

Niu, G., *et al.*, 2012. Two decompositions for the bicriteria job-shop scheduling problem with discretely controllable processing times. *International Journal of Production Research. DOI 10.1080/00207543.2011.651169.*

Pearl, J., 1984. *Heuristics: Intelligent Search strategies for Computer Problem Solving.* Addison-Wesley.

Rabiee, M., Zandieh, M., and Ramezani, P., 2012. Bi-objective partial flexible job shop scheduling problem: NSGA-II, NRGA, MOGA and PAES approaches. *International Journal of Production Research. DOI 10.1080/00207543.2011.648280.*

Ruiz-Torres, A.J., Ablanedo-Rosas, J.H., and Otero, L.D., 2012. Scheduling with multiple tasks per job - the case of quality control laboratories in the pharmaceutical industry. *International Journal of Production Research*, 50 (3), 691–705.

Schmidt, G., 2000. Scheduling with limited machine availability. *European Journal of Operational Research*, 5, 1–15.

Sierra, M.R., Mencía, C., and Varela, R., 2011a. Optimally Scheduling a Job-Shop with Operators and Total Flow Time Minimization. *CAEPIA 2011, Advances in Artificial Intelligence, Lecture Notes in Computer Science*, 7023, 193–202.

Sierra, M.R., Mencía, C., and Varela, R., 2011b. Searching for optimal schedules to the Job-Shop problem with operators. *Technical report. Computing Technologies Group. University of Oviedo.*

Sierra, M.R. and Varela, R., 2010. Pruning by Dominance in Best-First Search for the Job Shop Scheduling Problem with Total Flow Time. *Journal of Intelligent Manufacturing*, 21(1), 111–119.

Stern, R., *et al.*, 2010. Using Lookaheads with Optimal Best-First Search. *In*: *AAAI.*

Talbi, E., 2009. *Metaheuristics: From Design to Implementation.* Wiley Series on Parallel and Distributed Computing John Wiley & Sons.

Vela, C.R., Varela, R., and González, M.A., 2010. Local Search and Genetic Algorithm for the Job Shop Scheduling Problem with Sequence Dependent Setup Times. *Journal of Heuristics*, 16, 139 – 165.

Zouba, M., Baptiste, P., and Rebaine, D., 2009. Scheduling identical parallel machines and operators within a period based changing mode. *Comput. Oper. Res.*, 36 (12), 3231–3239.

Tables of the paper

Table 1. Errors in percentage of the solutions reached by the four algorithms averaged for each group of instances with the same size ($n \times m$). The time limit is 300s.

	CP		A*DFS		DFS		A*	
Size	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.
10×5	0	2,08	16	0,42	15	1,14	17	1,08
15×5	0	6,07	10	3,42	6	5,42	10	6,49
20×5	0	7,51	5	5,37	5	6,71	5	8,60
10×10	0	4,88	10	2,85	10	5,11	10	6,40
15×10	0	11,04	5	8,92	5	11,70	7	15,76
20×10	0	12,69	5	10,38	5	12,45	5	18,68
30×10	0	17,38	5	10,21	5	11,58	5	18,68
15×15	0	12,57	6	9,46	5	11,95	7	18,53
Avg. %Err.		9,28		6,62		8,26		11,78
Sum. #Sol.	0		62		56		66	

Table 2. Errors in percentage of the solutions reached by the four algorithms averaged for each group of instances with the same size ($n \times m$). The time limit is 3600s.

	CP		A*DFS		DFS		A*	
Size	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.
10×5	0	1,58	21	0,33	16	1,03	17	1,08
15×5	0	5,12	10	3,40	10	5,23	10	6,49
20×5	0	6,40	6	5,32	6	6,57	5	8,60
10×10	0	4,12	18	2,72	11	4,85	10	6,40
15×10	0	9,96	10	8,63	5	11,60	7	15,76
20×10	0	10,86	5	10,12	5	12,41	5	18,68
30×10	0	13,53	5	10,05	5	11,57	5	18,68
15×15	0	10,59	6	9,14	5	11,75	7	18,53
Avg. %Err.		7,77		6,22		8,13		11,78
Sum. #Sol.	0		81		63		66	

Table 3. p-values from Wilcoxon paired tests of the results from A*DFS and CP by 300s (5m) and 3600s (1h) when each group of instances with the same size ($n \times m$) is taken as population sample. The alternative hypothesis is that the errors from M1 at a time T1 are smaller than those from M2 at time T2.

	M1(T1)	A*DFS(5m)	A*DFS(1h)	A*DFS(5m)
Size	M2(T2)	CP(5m)	CP(1h)	CP(1h)
10×5		1,07E-04	1,61E-04	1,60E-04
15×5		9,84E-07	1,87E-04	2,09E-04
20×5		5,14E-04	1,47E-02	1,79E-02
10×10		1,21E-08	4,94E-07	2,11E-06
15×10		2,85E-06	1,87E-04	4,20E-03
20×10		2,06E-07	1,05E-02	3,95E-02
30×10		7,13E-10	1,77E-08	1,41E-07
15×15		1,34E-12	1,47E-07	1,13E-05

Table 4. Errors in percentage of the lower bounds reached by A*, DFS and A*DFS by 300s and 3600s. The values are averaged for each group of instances with the same size ($n \times m$). A* always finishes before 300s with either an optimal solution or the memory getting exhausted.

		300s		3600s	
Size	A*	A*DFS	DFS	A*DFS	DFS
10×5	0,15	0,17	2,43	0,00	2,28
15×5	0,07	0,07	1,89	0,00	1,79
20×5	0,02	0,06	0,95	0,00	0,93
10×10	0,48	0,83	4,72	0,00	4,62
15×10	0,06	0,22	1,66	0,00	1,66
20×10	0,06	0,11	0,71	0,00	0,71
30×10	0,01	0,04	0,22	0,00	0,22
15×15	0,03	0,25	1,19	0,00	1,19
Avg. Err.	0,11	0,22	1,72	0,00	1,68

List of figure captions

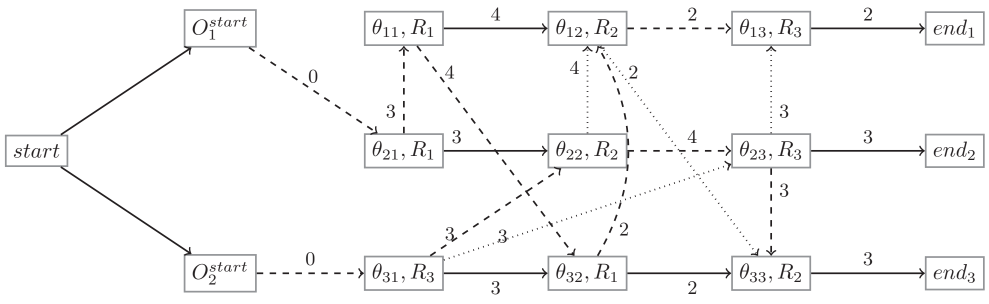


Figure 1. A feasible schedule to a problem with 3 jobs, 3 machines and 2 operators.

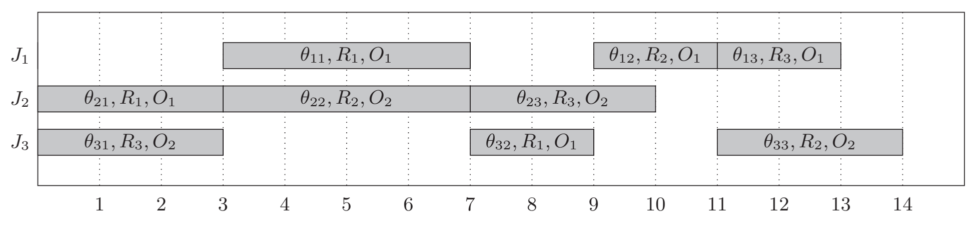


Figure 2. Gantt chart of the schedule represented in Figure 1.

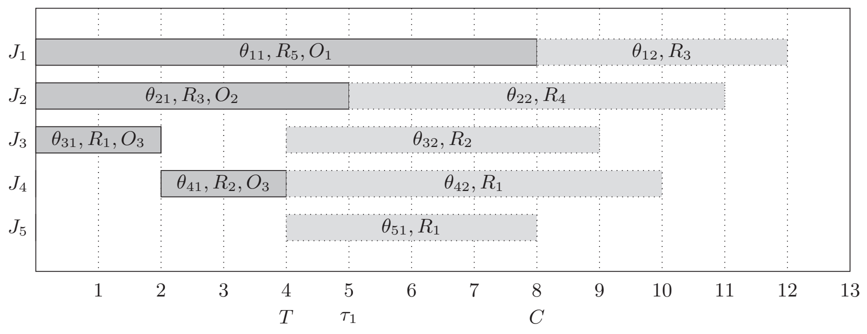
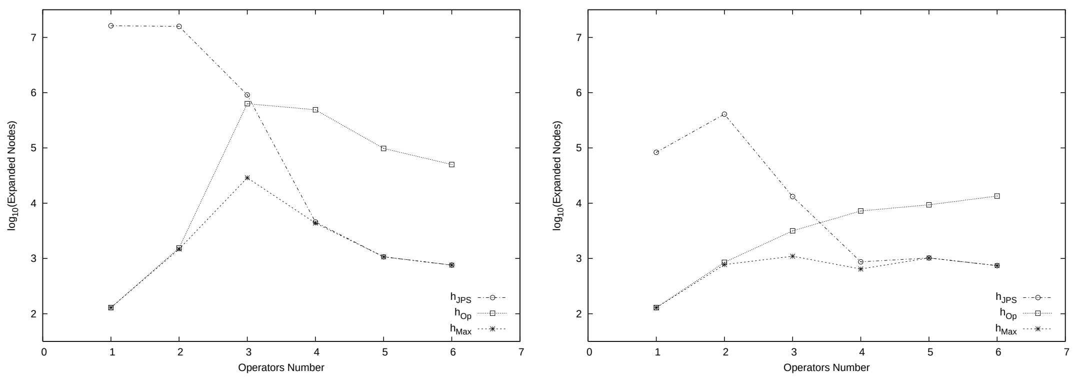


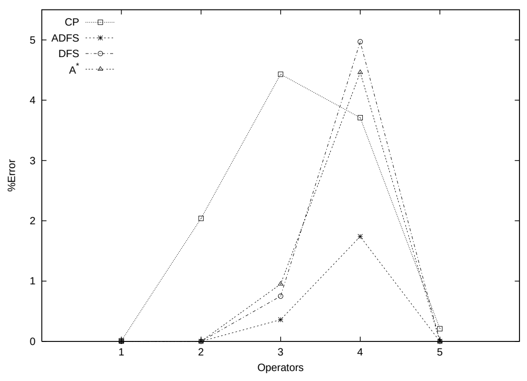
Figure 3. A partial schedule to a problem with 5 jobs, 5 machines and 3 operators. The scheduled operations are $\theta_{11}, \theta_{21}, \theta_{31}, \theta_{41}$



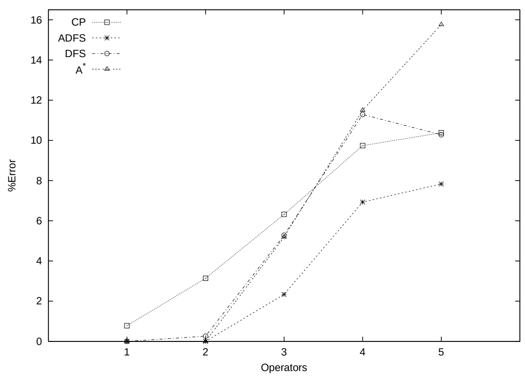
(a) A*DFS without pruning

(b) A*DFS with pruning

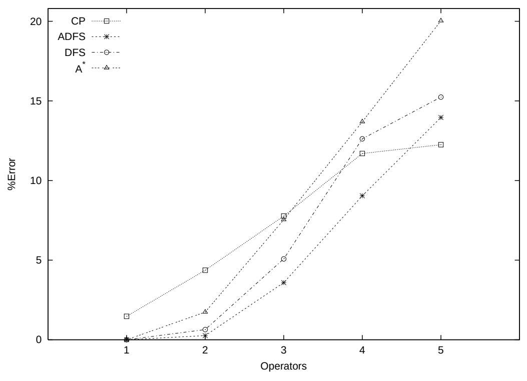
Figure 4. Summary of results from A*DFS on FT06 (6×6) instance for different combinations of heuristic, number of operators and pruning option. The time limit is 3600s.



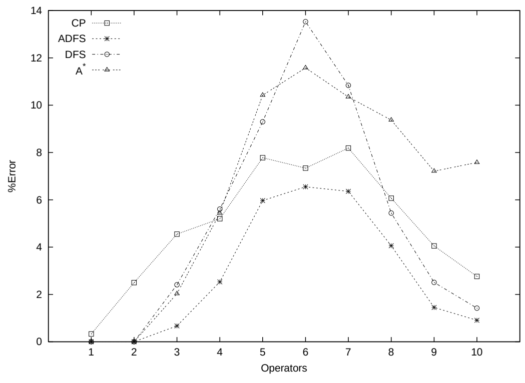
(a) 10 × 5



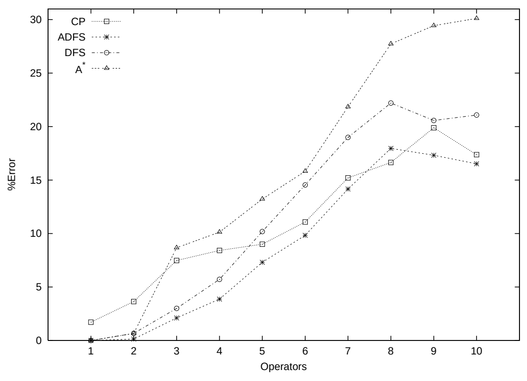
(b) 15 × 5



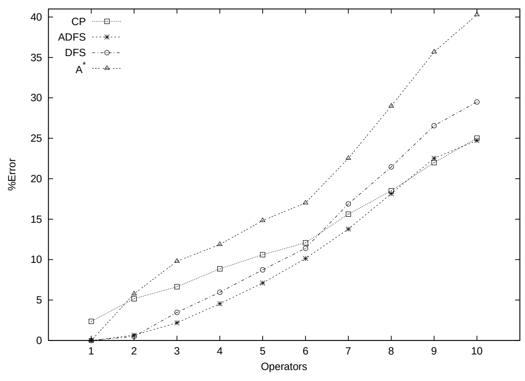
(c) 20 × 5



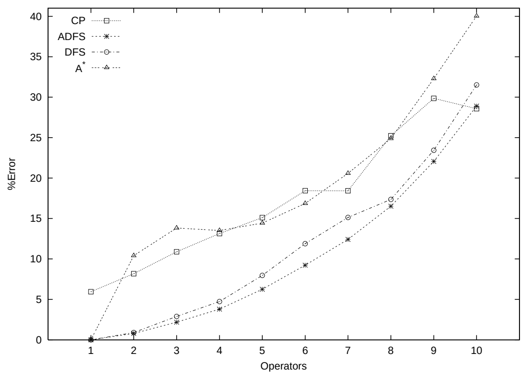
(d) 10 × 10



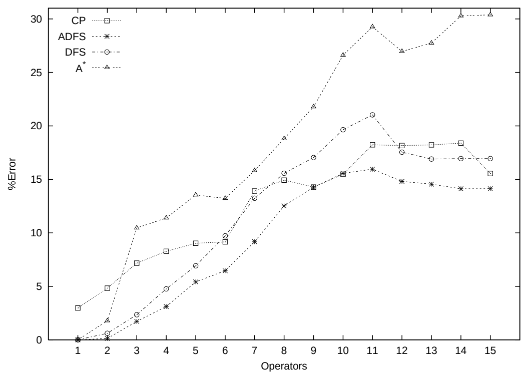
(e) 15 × 10



(f) 20 × 10

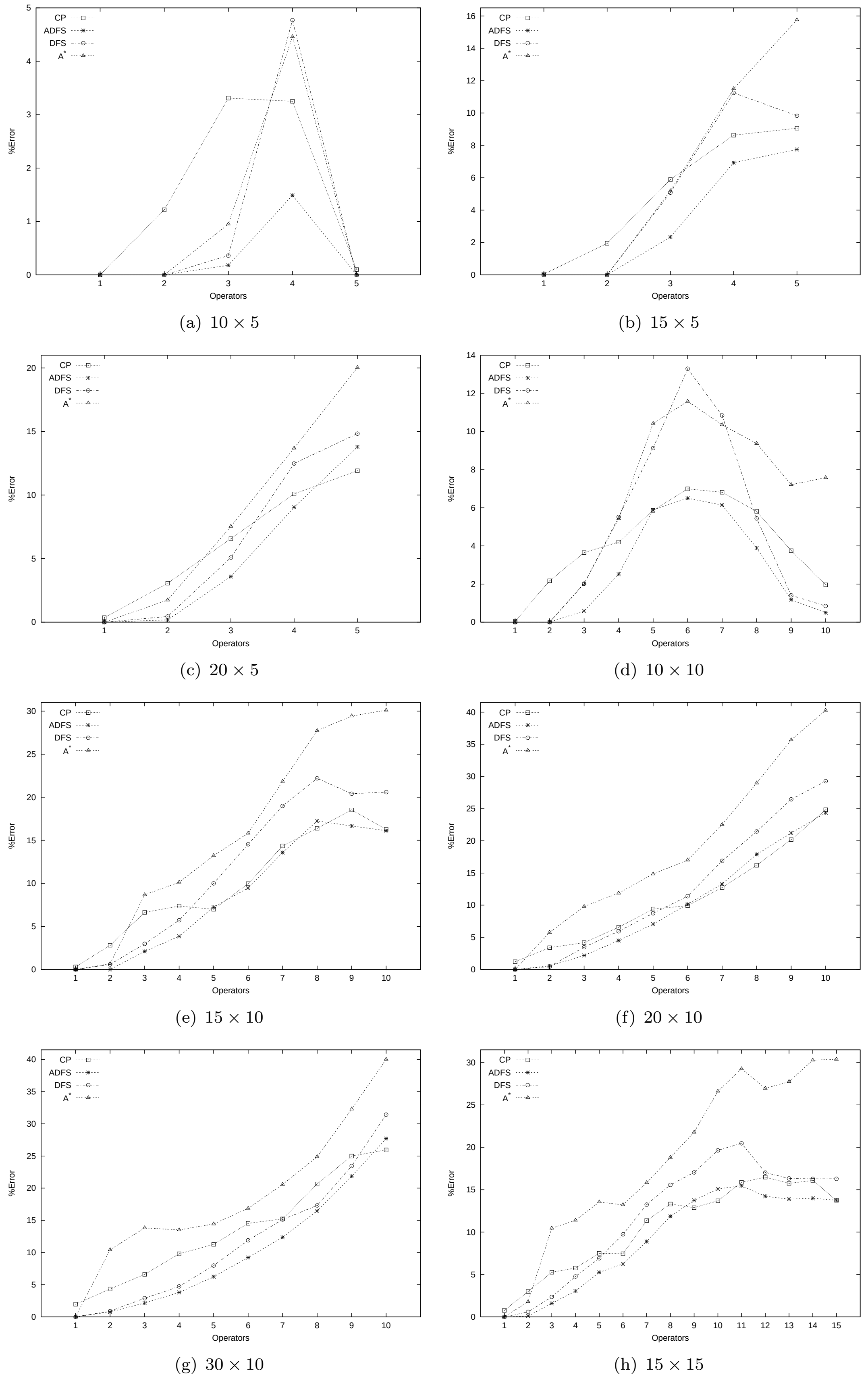


(g) 30 × 10



(h) 15 × 15

Figure 5. Errors in percentage from CP, A*DFS, DFS and A* across the LA set averaged for instances with the same size and number of operators. The time limit is 300s.



(a) 10×5 (b) 15×5

(c) 20×5 (d) 10×10

(e) 15×10 (f) 20×10

(g) 30×10 (h) 15×15

Figure 6. Errors in percentage from CP, A*DFS, DFS and A* across the LA set averaged for instances with the same size and number of operators. The time limit is 3600s.

Capítulo 6

INFORME SOBRE LA CALIDAD DE LAS PUBLICACIONES

En este capítulo se incluye información sobre la calidad y el factor de impacto de las publicaciones presentadas en el capítulo anterior.

Artículo 1

- Carlos Mencía, María R. Sierra and Ramiro Varela. Partially Informed Depth-First Search for the Job Shop Problem. *In Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS '10).* Toronto, Canada. AAAI Press. pp: 113-120. 2010
    - Estado: **Publicado.**
    - ERA 2010: A
    - CORE 2008: A+
    - CITESEER 2010: 0.1 (57/581)
    - CS Conference Ranking: 0.53(65/620)

Artículo 2

- Carlos Mencía, María R. Sierra and Ramiro Varela. Depth-first heuristic search for the job shop scheduling problem. *Annals of Operations Research* 2012. DOI 10.1007/s10479-012-1296-x
    - Estado: **Publicado online.**
    - Factor de impacto (JCR 2011): 0.840
    - Factor de impacto (5 años): 1.101
    - Categorías:
        - OPERATIONS RESEARCH & MANAGEMENT SCIENCE: 41/77(Q3), **T2**

Artículo 3

- Carlos Mencía, María R. Sierra and Ramiro Varela. Intensified iterative deepening A* with application to job shop scheduling. *Journal of Intelligent Manufacturing* 2013. DOI 10.1007/s10845-012-0726-6
    - Estado: **Publicado online.**
    - Factor de impacto (JCR 2011): 0.859
    - Factor de impacto (5 años): 1.414
    - Categorías:
        - COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE: 71/111(Q3), **T2**
        - ENGINEERING, MANUFACTURING: 19/37 (Q3), **T2**

Artículo 4

- Carlos Mencía, María R. Sierra and Ramiro Varela. An Efficient Hybrid Search Algorithm for Job Shop Scheduling with Operators. *International Journal of Production Research* 2013. DOI 10.1080/00207543.2013.802389
    - Estado: **Aceptado (en prensa).**
    - Factor de impacto (JCR 2011): 1.115
    - Factor de impacto (5 años): 1.367
    - Categorías:
        - OPERATIONS RESEARCH & MANAGEMENT SCIENCE: 25/77(Q2), **T1**
        - ENGINEERING, MANUFACTURING: 11/37 (Q2), **T1**
        - ENGINEERING, INDUSTRIAL: 18/43 (Q2), **T2**

Bibliografía

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Managament Science*, 34:391–401, 1988.

[2] A. Agnetis, M. Flamini, G. Nicosia, and A. Pacifici. A job-shop problem with one additional resource type. *J. Scheduling*, 14(3):225–237, 2011.

[3] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 3:149–156, 1991.

[4] C. Artigues, P. Lopez, and P. Ayache. Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research*, 138:21–52, 2005.

[5] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[6] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[7] J. C. Beck. Solution-guided multi-point constructive search for job shop scheduling. *J. Artif. Intell. Res. (JAIR)*, 29:49–77, 2007.

[8] J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artif. Intell.*, 117(1):31–81, 2000.

[9] C. Bierwirth. A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spectrum*, 17:87–92, 1995.

[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] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[15] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[16] J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[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] A. Cesta, A. Oddi, and S. F. Smith. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In H. A. Kautz and B. W. Porter, editors, *AAAI/IAAI*, pages 742–747. AAAI Press / The MIT Press, 2000.

[19] L. Climent, M. A. Salido, and F. Barber. Robustness in dynamic constraint satisfaction problems. *Int. Journal of Innovative Computing Information and Control*, 8(4):2513–2532, 2012.

[20] M. Dell' Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[21] U. Dorndorf, E. Pesch, and T. Phan-Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artif. Intell.*, 122:189–240, 2000.

[22] J. Escamilla, M. Rodríguez-Molins, M. A. Salido, M. R. Sierra, C. Mencía, and F. Barber. Robust solutions to job-shop scheduling problems with operators. In *ICTAI*, pages 299–306. IEEE, 2012.

[23] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2001.

[24] J. Framinan, J. Gupta, and R. Leisten. A review and classification heuristics for permutation flow shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55:1243–1255, 2004.

[25] M. Garey and D. Johnson. *Computers and Intractability*. Freeman, 1979.

[26] A. Gharbi and M. Labidi. Extending the single machine-based relaxation scheme for the job shop scheduling problem. *Electronic Notes in Discrete Mathematics*, 36:1057–1064, 2010.

[27] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[28] C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[29] M. A. González, C. R. Vela, I. González-Rodríguez, and R. Varela. Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. *Jornal of Intelligent Manufacturing*, Online First(DOI 10.1007/s10845-011-0622-5), 2012.

[30] M. A. González, C. R. Vela, M. R. Sierra, and R. Varela. Tabu search and genetic algorithm for scheduling with total flow time minimization. In *COPLAS 2010*, pages 33–41, 2010.

[31] 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.

[32] R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[33] O. Guyon, P. Lemaire, E. Pinson, and D. Rivreau. Solving an integrated job-shop problem with human resource constraints. *Annals of Operations Research*, On line, DOI 10.1007/s10479-012-1132-3, 2012.

[34] S. Harikumar and S. Kumar. Iterative deepening multiobjective A*. *Inf. Process. Lett.*, 58(1):11–15, 1996.

[35] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Science and Cybernetics*, 4(2):100–107, 1968.

[36] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI 1995 (1)*, pages 607–615, 1995.

[37] T. Ibaraki. The power of dominance relations in branch-and-bound algorithms. *Journal of the Association for Computing Machinery*, 24(2):264–279, 1977.

[38] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.*, 27:97–109, 1985.

[39] R. E. Korf. Improved limited discrepancy search. In W. J. Clancey and D. S. Weld, editors, *AAAI/IAAI, Vol. 1*, pages 286–291. AAAI Press / The MIT Press, 1996.

[40] A. Kovács and J. Beck. A global constraint for total weighted completion time for unary resources. *Constraints*, 16:100–123, 2011. 10.1007/s10601-009-9088-x.

[41] P. Laborie. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artif. Intell.*, 143(2):151–188, 2003.

[42] P. Laborie. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR09)*, pages 148–162, 2009.

[43] S.-W. Lin, C.-Y. Huang, C.-C. Lu, and K.-C. Ying. Minimizig total flow time in permutation flowshop environment. *International Journal of Innovative Computing, Information and Control*, 8(10(A)):6599 – 6612, 2012.

[44] L. Mandow and J.-L. P. de-la Cruz. Multiobjective A* search with consistent heuristics. *J. ACM*, 57(5), 2010.

[45] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedingos of IPCO 1996*, pages 389–403, 1996.

[46] S. Meeran and M. Morshed. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of Intelligent Manufacturing*, 23(4):1063–1078, 2011.

[47] C. Mencía, M. R. Sierra, M. A. A. Salido, J. Escamilla, and R. Varela. Combining global pruning rules with depth-first search for the job shop scheduling problem with operators. In *19th RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with combinatorial explosion*, 2012.

[48] C. Mencía, M. R. Sierra, and R. Varela. Partially informed depth-first search for the job shop problem. In *Proceedings of ICAPS 2010*, pages 113–120, 2010.

[49] C. Mencía, M. R. Sierra, and R. Varela. Depth-first heuristic search for the job shop scheduling problem. *Annals of Operations Research*, 2012. doi: 10.1007/s10479-012-1296-x.

[50] C. Mencía, M. R. Sierra, and R. Varela. An efficient hybrid search algorithm for job shop scheduling with operators. *International Journal of Production Research*, 2013. doi: 10.1080/00207543.2013.802389.

[51] C. Mencía, M. R. Sierra, and R. Varela. Intensified iterative deepening A* with application to job shop scheduling. *Journal of Intelligent Manufacturing*, 2013. doi: 10.1007/s10845-012-0726-6.

[52] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Genetic algorithm for job-shop scheduling with operators. In *New Challenges on Bioinspired Applications, LNCS 6687/2011*, pages 305–314, 2011.

[53] P. Meseguer. Interleaved depth-first search. In *IJCAI*, pages 1382–1387. Morgan Kaufmann, 1997.

[54] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *J. Scheduling*, 8(2):145–159, 2005.

[55] A. Oddi, R. Rasconi, A. Cesta, and S. F. Smith. Iterative improvement algorithms for the blocking job shop. In *ICAPS*, 2012.

[56] J. Pearl. *Heuristics: Intelligent Search strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[57] M. L. Pinedo. *Scheduling. Theory, Algorithms, and Systems*. Springer, third edition, 2008.

[58] N. Policella, A. Cesta, A. Oddi, and S. F. Smith. Solve-and-robustify. *J. Scheduling*, 12(3):299–314, 2009.

[59] J. Puente, C. R. Vela, and I. G. Rodríguez. Fast local search for fuzzy job shop scheduling. In *Proceedings of ECAI 2010*, pages 739–744, 2010.

[60] B. Roy and B. Sussman. Les problemes d'ordonnancements avec contraintes disjonctives. Technical report, Notes DS no. 9 bis, SEMA, Paris, 1964.

[61] A. J. Ruiz-Torres, J. H. Ablanedo-Rosas, and L. D. Otero. Scheduling with multiple tasks per job - the case of quality control laboratories in the pharmaceutical industry. *International Journal of Production Research*, 50(3):691–705, 2012.

[62] 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.

[63] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. J. Maher and J.-F. Puget, editors, *CP*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1998.

[64] M. R. Sierra. *Improving Heuristic Search Algorithms by means of Pruning by Dominance. Application to Scheduling Problems*. PhD thesis, University of Oviedo, 2009.

[65] M. R. Sierra, C. Mencía, and R. Varela. Optimally scheduling a job-shop with operators and total flow time minimization. In *Advances in Artificial Intelligence, LNCS 7023/2011*, pages 193–202, 2011.

[66] M. R. Sierra, C. Mencía, and R. Varela. Searching for optimal schedules to the job-shop problem with operators. *Technical report. Computing Technologies Group. University of Oviedo*, 2011.

[67] M. R. 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*, 21(1):111–119, 2010.

[68] S. F. Smith and C.-C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of AAAI 1993*, pages 139–144, 1993.

[69] M. J. Streeter and S. F. Smith. Exploiting the power of local search in a branch and bound algorithm for job shop scheduling. In *Proceedings of ICAPS 2006*, pages 324–333, 2006.

[70] M. J. Streeter and S. F. Smith. How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *J. Artif. Intell. Res. (JAIR)*, 26:247–287, 2006.

[71] M. J. Streeter and S. F. Smith. Using decision procedures efficiently for optimization. In *Proceedings of ICAPS 2007*, pages 312–319, 2007.

[72] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, January 1993.

[73] E. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2009.

[74] P. Van Laarhoven, E. Aarts, and K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[75] C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *J. Heuristics*, 16(2):139–165, 2010.

[76] N. R. Vempaty, V. Kumar, and R. E. Korf. Depth-first versus best-first search. In *AAAI*, pages 434–440, 1991.

[77] P. Vilím. *Global Constraints in Scheduling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic, August 2007.

[78] T. Walsh. Depth-bounded discrepancy search. In *In Proceedings of IJCAI-97*, pages 1388–1393, 1997.

[79] J.-P. Watson and J. C. Beck. A hybrid constraint programming / local search approach to the job-shop scheduling problem. In *Proceedings of CPAIOR 2008*, pages 263–277, 2008.

[80] J.-Z. Wu, X.-C. Hao, C.-F. Chien, and M. Gen. A novel bi-vector encoding genetic algorithm for the simultaneous multiple resources scheduling problem. *Journal of Intelligent Manufacturing*, 23:2255–2270, 2012.

[81] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & OR*, 35:282–294, 2008.

[82] M. Zouba, P. , and D. Rebaine. Scheduling identical parallel machines and operators within a period based changing mode. *Comput. Oper. Res.*, 36(12):3231–3239, Dec. 2009.