Journal of Artificial Intelligence Research 1 (2011) 1-30 Submitted x/xxxx; published x/xxxx

Depth-First Heuristic Search for the Job Shop Scheduling Problem

Carlos Mencía MENCIACARLOS@UNIOVI.ES
María R. Sierra SIERRAMARIA@UNIOVI.ES
Ramiro Varela RAMIRO@UNIOVI.ES
Department of Computing, University of Oviedo. Campus de Viesques, 33271 Gijón, Spain

Abstract

We evaluate two variants of depth-first search algorithms and consider the classical job shop scheduling problem as test bed. The starting point of our work is the well-known branch-and-bound algorithm proposed by P. Brucker et al. This is a very efficient algorithm which is able to obtain and certify optimal solutions for many hard instances. It exploits a chronological backtracking strategy that produces an almost negligible memory consumption. As an alternative, we propose to use a partially informed depth-first search strategy that is slightly more memory consuming but at the same time it is able to exploit much better the heuristic knowledge available. 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 of medium, large and very large sizes. The results show that the proposed partially informed depth-first search algorithm outperforms the original Brucker's algorithm. It reaches and certifies optimal solutions quickly for medium size instances and, for large and very large ones, it reaches much better lower and upper bounds when both algorithms are given the same amount of time. Moreover, the experiments make it clear that partially informed depth-first search is able to exploit the heuristic knowledge much better than backtracking. In particular, when the more informed and time consuming heuristic is used, the latter gets worse while the first is much more efficient in reaching optimal or suboptimal solutions.

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

A variety of constraint optimization problems, such as the JSSP, have been successfully solved by means of constraint-based reasoning methods, which interleave the execution of *backtracking search* and *constraint propagation* algorithms that reduce and simplify the problem represented by each node in the search space.

The backtracking search strategy is a popular variant of depth-first search in which only one of the successors of the current state is generated and then selected for continuing the search from in the next iteration. So, it is suitable for solving hard problems as it requires few memory resources. However, the information of the successor states is not fully available at the moment of deciding what path in the search space to follow as they are not yet

generated, so it can be expected that depth-first search algorithms are able to exploit the heuristic knowledge from the problem domain to a larger extent than backtracking search.

Backtracking search and constraint propagation are usually combined into a *branch and bound* algorithm.

One smart and successful algorithm of this kind, for solving the job shop scheduling problem with makespan minimization, is that proposed by P. Brucker et al. in (Brucker, Jurisch, & Sievers, 1994a). This algorithm was designed from some ideas and results due to other researchers such as J. Carlier and E. Pinson (Carlier, 1982), (Carlier & Pinson, 1989), (Carlier & Pinson, 1990), (Carlier & Pinson, 1994) or B. Giffler and G. L. Thomson (Giffler & Thompson, 1960). As it is usual in this kind of algorithms, Brucker's algorithm computes a heuristic estimation for each state n visited along the search. This estimation is a lower bound for the cost of the best solution reachable from n, denoted $f(n)$. When $f(n) \geq UB$, where UB is the cost of the incumbent solution, i.e. the best solution reached so far, the state n is pruned, being this pruning capability the only utility of the $f-$values.

Building on Brucker's branch and bound algorithm, the main contributions of this paper are the following:

- In order to exploit the heuristic estimation to a larger extent, we propose to use a depth-first search strategy instead of backtracking search. In our algorithm, the $f-$values are used to discriminate in favor of the most promising successors of a given state. Therefore, the heuristic estimation is not only used for pruning, but also to guide the search towards promising regions of the search space. As we will see, this partially informed depth-first search algorithm is able to exploit much better the heuristic knowledge available than it is the simple backtracking algorithm. At the same time, the memory requirement remains reasonably low, as it only has to store the unexplored successors along the branch from the initial state to the current one.

- Starting from some ideas proposed in (Carlier & Pinson, 1990), we devised a new heuristic estimation. This estimation is obtained by combining the lower bound estimation used by Brucker's algorithm together with constraint propagation rules. We present formal and empirical studies of this heuristic. As we will see, it is consistent (or monotonic) and more informed than the one used in Brucker's algorithm. At the same time, it is more time consuming, although it can be efficiently calculated.

- We present theoretical results about the consequences of using monotonic heuristics in a depth-first search algorithm. Concretely, we prove that the sequence of global lower bounds computed by the partially informed depth-first search algorithm is non decreasing if it uses a monotonic heuristic. Besides, we show that the new heuristic can be computed more efficiently thanks to it being monotonic.

We have conducted an experimental study across conventional medium and large instances to compare our approach with the original Brucker's algorithm. The results show clearly that our approach is able to better exploit the available heuristic estimation. 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, our approach 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 Section 2 we give some background on heuristic search. In Section 3 a formulation of the JSSP is given and some of the solving methods are revised. Section 4 outlines the main characteristics of Brucker's algorithm. In Section 5 we describe the proposed partially informed depth-first search algorithm for the JSSP. In Section 6 we present the new heuristic. We include a formal study to justify its properties and efficiency. Section 7 reports the results of the experimental study. Finally, in Section 8 we summarize the main conclusions of the paper and propose some ideas for future research.

The present paper is an extended version of (Mencía, Sierra, & Varela, 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 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. This list represents the frontier between the explored and unexplored regions of the search space. 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 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 graph. 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 heuristic knowledge represented by f can be fully exploited selecting for expansion the most promising state in the whole list OPEN at each step of the search. 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 $g(n)$ 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 optimal

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

A^* algorithm proposed by Hart, Nilsson, and Raphael (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. Another interesting property is monotonicity, i.e. $h(n_1) \leq h(n_2) + c(n_1, n_2)$, for all states n_1, n_2. This property is equivalent to consistency defined as $h(n_1) \leq h(n_2) + k(n_1, n_2)$, 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* (DFS) 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: at each step the next node for expansion is selected among the successors of the last state expanded. Whenever a leaf is reached the search backtracks and continues from a shallower state.

The backtracking mechanism allows the search algorithm to recover from deadends and to keep on looking for solutions after reaching a goal state. So, it returns a sequence of improving solutions and eventually finds and proves an optimum if it is given enough time to exhaustively search the whole search space, i.e. it is an any-time algorithm.

A popular variant of depth-first search is *backtracking search* (BS). Unlike DFS, a backtracking search algorithm only generates one successor $n' \in SUC(n)$ at a time and selects it for the next iteration. As it was pointed in (Pearl, 1984), the main advantage of this search strategy is the low use of memory: it requires even less memory than DFS, 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 BS algorithm can be implemented very efficiently, as it does not have to duplicate the information of a state for all its successors. So, BS is able to traverse the search space faster than DFS.

One of the main drawbacks of these strategies is the big effort that has to be made 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 needs to traverse a whole subtree before proceeding the search from a better option. This problem becomes worse at shallow levels of the search tree. So, it is worth using the available knowledge from the problem domain in order to choose the most promising successor for continuing the search from. This way, in DFS, after expanding a state n, all its succesors $n' \in SUC(n)$ are generated and stored in the list OPEN sorted according to a given criterion, like a dispatching rule or a heuristic. In the case of BS, it stores and sorts the rules that may be used for generating the successors of n instead.

Hence, DFS can exploit the knowledge to a greater extent than BS for guiding the search, as it has all the information about each successor state to decide what path to follow. This may have important consequences on the performance of the algorithms.

As we have pointed in the introductory section of this paper, one successful approach for dealing with constraint satisfaction and optimization problems is the combination of search with constraint propagation methods. These methods make deductions about the problem represented by a state and reduce to a great extent the search space. So, a DFS algorithm can apply constraint propagation to each successor when it is generated and take profit from the deducted knowledge to guide the search.

Finding good solutions soon may boost the whole search process when these search strategies are used in a *branch and bound* algorithm ($B\&B$). Here, a lower bound (f-value) on the cost of the best solution reachable from each generated state is computed. States n such that $f(n) \geq UB$, where UB is the cost of the incumbent solution (i.e. the best solution found so far), can be pruned as they do not lead to any better solution. New upper bounds are obtained each time a state in Γ is reached.

In order to keep a low use of memory and at the same time to exploit the heuristic knowledge available as long as possible, we propose here to use a depth-first search strategy in which all successors of an expanded state n are sorted by non decreasing f-values and appended at the beginning of the list OPEN. This is usually termed partially informed depth-first search (Pearl, 1984).

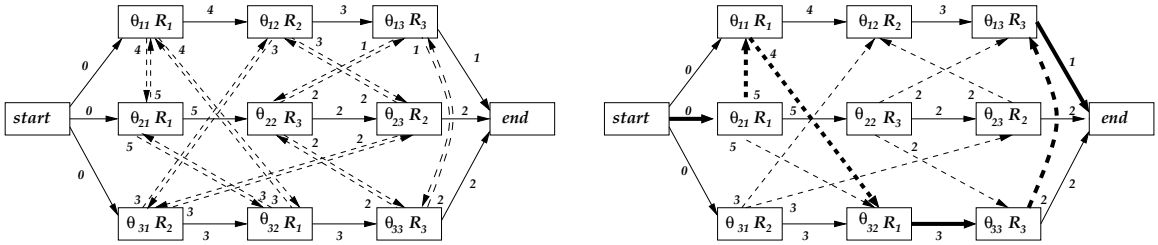
The partially informed depth-first search algorithm terminates when the list OPEN becomes empty or after a given time. In the first case an optimal solution with cost UB is returned. In the second case the algorithm returns the best solution reached, with cost UB, and the best lower bound obtained, i.e. the lowest f-value in the OPEN list. Here there is a difference w.r.t. backtracking algorithms that only can return as lower bound the lowest f-value from all evaluated states.

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 for this problem.

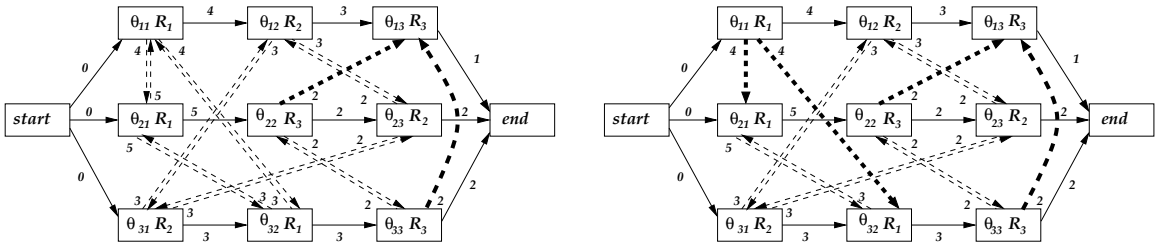
3.1 Problem Formulation

The job shop scheduling problem (JSSP) 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 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: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, 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 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).



(a) (b)

Figure 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.



(a) (b)

Figure 2: Two partial solution graphs for the problem instance in Figure 1(a)

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 & 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 $E = \cup_{i=1,...,M} E_i$. 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 $H = \cup_{i=1,...,M} H_i$, 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*. 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(b) shows a solution graph for the instance of Figure 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. PJ_v and SJ_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 Figure 1(a).

In a partial schedule, heads and tails can be estimated as

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

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

with $r_{start} = q_{end} = 0$ and where DP_v denotes the 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 Figure 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 Figure 2(b), being $\{\theta_{21}, \theta_{32}\}$ the subset of disjunctive successors giving the largest value in expression (2).

3.3 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 broad variety of solving methods have been proposed for the $J||Cmax$ problem up to date.

Designed from seminal ideas proposed by van Laarhoven, Aarts, and Lenstra (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. These algorithms are able to reach high quality solutions to extremely hard instances. A thorough analysis of the outstanding Nowicki and Smutnicki's algorithm is carried out by Watson et al. (2006).

Also, effective exact methods to cope with the JSSP have been developed within the constraint-based scheduling community. 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 of successful constraint propagation methods for this problem are timetabling, edge-finding and energetic reasoning (Dorndorf et al., 2000; Laborie, 2003; Baptiste et al., 2001). Regarding search strategies, backtracking search 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 & Fox, 2000) and slack-based heuristics (Smith & Cheng, 1993).

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 for solving the JSSP (Streeter & Smith, 2007). 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 & 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 in 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 for guiding the search.

Finally, hybrid algorithms combining exact and local search approaches have been proposed recently. Streeter and Smith (2006a), combine an iterated local search procedure with Brucker's $B\&B$ algorithm. Watson and Beck (2008) propose combining SGMPS with tabu search. In both cases performance improvements are achieved over the two building methods.

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 P. Brucker et al. (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 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 by Carlier (1982). Upper bounds are obtained by means of the greedy $G\&T$ algorithm (Giffler & Thompson, 1960).

Finally, Brucker's algorithm exploits a constraint propagation method termed *immediate selection* (Carlier & 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. In fact, it

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. Exact algorithms that use other search strategies, as backtracking search with restarts and randomized heuristics, or non-chronological backtracking such as limited discrepancy search usually obtain better solutions than Brucker's algorithm to large problem instances that cannot be solved to optimality in a reasonable time, but they are not better for proving quickly the optimality of a solution unless the optimal cost is known in advance because they need to reexpand a lot of states.

is able to solve a number of the famous set of instances selected by Applegate and Cook (1991) that are considered very hard to solve.

5. Partially informed depth-first search algorithm for the JSSP

In this section, we present the key components of the proposed partially informed depth-first search algorithm. We start describing the search space, the heuristic estimation and the algorithm used to obtain upper bounds. All these three elements are borrowed from Brucker's algorithm. The search space is described by the states representation model, the expansion mechanism and also by a powerful constraint propagation method that reduces the effective search space. As we will see, these elements are interdependent, so we have chosen an appropriate order to facilitate their comprehension.

Then we consider the issue of the control strategy. As we have pointed, Brucker's algorithm exploits a backtracking search strategy. This allows to reduce the memory requirements to a minimum, but at the same time it can hardly take advantage of the knowledge provided by the heuristic estimations. As an alternative, we propose to use a partially informed depth-first search that takes more profit from the heuristic knowledge while the memory requirements are still low.

5.1 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 $FD_n = \oslash$. The partial solution graphs of Figure 1 represents two possible search states. In the state of Figure 2(a), only the arcs $(\theta_{33} \rightarrow \theta_{13})$ and $(\theta_{22} \rightarrow \theta_{13})$ are fixed, while in the state of Figure 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 Figure 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.

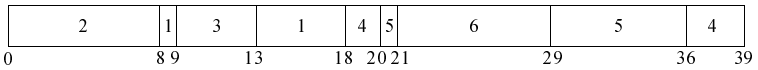
Heads and tails in n are calculated by expressions (1) and (2). In (Brucker, Jurisch, & Sievers, 1994b), algorithms for computing these values in polynomial time are given. For instance, the algorithm for computing the head of an operation v in a state n is the following: the disjunctive predecessors of v, $DP_v(n)$, are sorted by non decreasing heads as $(v_1, ..., v_k)$, 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.*

5.2 Heuristic estimation

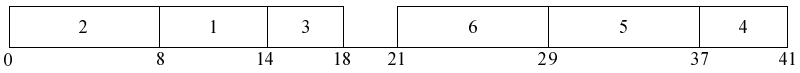
For each generated state n, a heuristic estimation $f(n)$ of the cost of the best solution reachable from n (i.e. the cost of the best path from the initial state to a goal constrained to

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

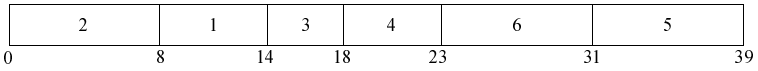
a) An OMS problem instance



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)

Figure 3: The Jackson's Preemptive Schedule for an OMS problem instance

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).

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 for n.

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.

5.3 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 & 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 1 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 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. 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 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. From these values the incumbent solution and its cost UB are updated.

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

$SC = \{start\}$;
while $(SC \neq O)$ **do**
  $A = \{v \in O \setminus SC;\ DP_v \cup \{PJ_v\} \subset SC\}$;
  $v^* = \arg\min\{r_u + p_u;\ u \in A\}$;
  $B = \{v \in A;\ R_v = R_{v^*} \text{ and } r_v < r_{v^*} + p_{v^*}\}$;
  $C = \{v \in O \setminus SC;\ R_v = R_{v^*}\}$;
  $w^* = \arg\min\{\text{makespan of JPS}(C \setminus \{w\}) \text{ after scheduling } w;\ w \in B\}$;
  Schedule w^* in S at time r_{w^*};
  Add w^* to SC and update heads of operations not included in SC;
end while
return S and its makespan;

5.4 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 1 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 1.
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.

So, 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 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 *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 $\{i \to l; i \in B_j \setminus \{l\}\}$.

For example, if we consider the search state of Figure 2(a) the heuristic solution might be that of Figure 1(b). In this case there are two critical blocks $B_1 = (\theta_{21}, \theta_{11}, \theta_{32})$ and $B_2 = (\theta_{33}, \theta_{13})$. The before-candidates are then $E_1^B = (\theta_{11}, \theta_{32})$ and $E_2^B = (\theta_{13})$, and the after-candidates $E_2^A = (\theta_{21}, \theta_{11})$ and $E_2^A = (\theta_{33})$. So, there are six candidate successors which would be obtained by fixing the sets of arcs: $\{(\theta_{11} \to \theta_{21}), (\theta_{11} \to \theta_{32})\}$, $\{(\theta_{32} \to \theta_{21}), (\theta_{32} \to \theta_{11})\}$ and $\{(\theta_{13} \to \theta_{33})\}$ from the before candidates and $\{(\theta_{11} \to \theta_{21}), (\theta_{32} \to \theta_{21})\}, \{(\theta_{21} \to \theta_{11}), (\theta_{32} \to \theta_{11})\}$ and $\{(\theta_{13} \to \theta_{33})\}$ from the after-candidates. Some of these candidate successors could be discarded, for example the two successors generated from block B_2 are the same due to this block having only two operations. Moreover, as the arc $(\theta_{33} \to \theta_{13})$ is fixed in the state of Figure 2(a), this candidate state will be definitively discarded due to a cycle in the partial solution graph after fixing $(\theta_{13} \to \theta_{33})$.

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 $(E_1, \ldots, E_{2k})$ 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 $E_1, \ldots, E_{t-1}$ 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 $l \in E_j^B$, $j = 1 \ldots k$, generate a candidate successor s by fixing the arcs $FD_s = FD_n \cup S_l^B$, where

$$S_l^B = \bigcup_{E_i^B < E_j^B} F_i \cup \bigcup_{E_i^A < E_j^B} L_i \cup \{l \to i : i \in B_j \setminus \{l\}\}. \quad (3)$$

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

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

In our running example, if we take the order $(E_1^B, E_1^A, E_2^B, E_2^A)$, then the first two successors would be generated from E_1^B by fixing the set of arcs $\{(\theta_{11} \to \theta_{21}), (\theta_{11} \to \theta_{32})\}$ and $\{(\theta_{32} \to \theta_{21}), (\theta_{32} \to \theta_{11})\}$ respectively. Then, as $F_1 = \{(\theta_{21} \to \theta_{11}), (\theta_{21} \to \theta_{32})\}$, the next two successors would came from E_1^A by fixing $\{(\theta_{21} \to \theta_{11}), (\theta_{21} \to \theta_{32}), (\theta_{11} \to \theta_{21}), (\theta_{32} \to \theta_{21})\}$ and $\{(\theta_{21} \to \theta_{11}), (\theta_{21} \to \theta_{32}), (\theta_{21} \to \theta_{11}), (\theta_{32} \to \theta_{11})\}$ respectively. Then, $L_1 = \{(\theta_{21} \to \theta_{32}), (\theta_{11} \to \theta_{32})\}$, so the next successor would be added the arcs $\{(\theta_{21} \to \theta_{11}), (\theta_{21} \to \theta_{32}), (\theta_{21} \to \theta_{32}), (\theta_{11} \to \theta_{32}), (\theta_{13} \to \theta_{33})\}$. And finally, as $F_2 = \{(\theta_{33} \to \theta_{13})\}$, for the last candidate successor the arcs added would be $\{(\theta_{21} \to \theta_{11}), (\theta_{21} \to \theta_{32}), (\theta_{21} \to \theta_{32}), (\theta_{11} \to \theta_{32}), (\theta_{33} \to \theta_{13}), (\theta_{13} \to \theta_{33})\}$.

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 Section 5.1. 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.

In the example, only three out of the six candidates will give rise to feasible successors, which are obtained by fixing the sets of arcs: $\{(\theta_{11} \to \theta_{21}), (\theta_{11} \to \theta_{32})\}$, $\{(\theta_{32} \to \theta_{21}), (\theta_{32} \to \theta_{11})\}$ and $\{(\theta_{21} \to \theta_{11}), (\theta_{21} \to \theta_{32}), (\theta_{32} \to \theta_{11})\}$ respectively. The first one is represented in Figure 2(b).

5.5 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 (Carlier & Pinson, 1989), termed immediate selection[3]. This method is applied to each generated state and fixes additional arcs that have to be included in every feasible schedule, reachable from the state, having makespan lower than that of the incumbent solution, UB.

These arcs are fixed by the following procedure. 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.

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

$$r_c + p_c + p_j + q_j \geq UB, \quad (5)$$

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

The arc $(j \to c)$ is called *direct arc* and the procedure Select given in Algorithm 2 calculates all direct arcs for the state n. 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.

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

Algorithm 2 *PROCEDURE Select*

```
for all c, j ∈ I, c ≠ j do
  if r_c + p_c + p_j + q_j ≥ UB then
    fix the arc (j → c);
  end if
end for
```

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 the operations in J.

2) If

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

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), 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 the primal arcs $\{j \rightarrow c; j \in J\}$.

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

$$q_J = \max_{J' \subseteq J} \{\sum_{j \in J'} p_j + \min_{j \in J'} q_j\}. \tag{9}$$

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

In (Brucker et al., 1994a) and (Brucker, 2004) an algorithm is given that computes all primal and dual arcs in polynomial time. This algorithm is based on 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

Algorithm 3 *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 completion time of c;
4: If $s_c - p_c > r_c$ then $r_c = s_c - p_c$;

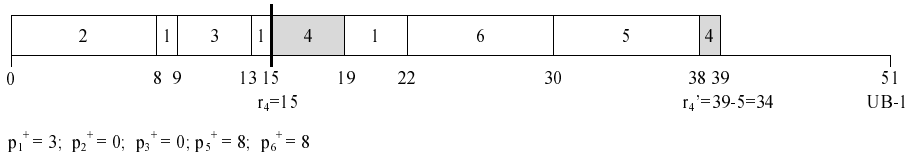


Figure 4: Calculation of earliest completion times s_c to $c = 4$

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 $r'_c = s_c - p_c$. Analogously for improving tails.

The algorithm for improving the head of an operation c is given in Algorithm 3.

Figure 4 shows the schedule built by the Algorithm 3 to improve the head of operation $c = 4$ with $UB = 52$ for the same instance as in Figure 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 $r'_4 + p_4 + p_5 + q_5 = 60 \geq 52$. 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, we do not include the proofs here and refer to the interested readers to the works by P. Brucker et al. (Brucker, 2004) and (Brucker et al., 1994a), and J. Carlier et al. (Carlier & Pinson, 1994).

Theorem 4. *(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 5. *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.*

Finally, the algorithm used to fix additional disjunctive arcs is given in Algorithm 4.

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 fact that the additional arcs might influence the heads and tails of all operations. So, in principle steps (1) to (4) should be repeated as long as new disjunctive arcs get fixed. However, after

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;*}

(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 the incumbent upper bound, UB, can be derived from the current heads and tails.

In the first case the inconsistency is clear as all the arcs fixed in G_n must be included in the solution graph of any solution, with cost lower than UB, reachable from n. But a solution graph must be acyclic.

In the second case, even if G_n has no cycles, there is an inconsistency as a solution with cost lower than UB containing the arcs fixed in G_n cannot exist. In order to check this 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 commented in Section 5.1, 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 Algorithm Immediate Selection we will use the notation $n_{UB} = IS(n, UB)$, where UB is the current upper bound, n is a state before IS, i.e. just after expansion as before or after candidate. If one of the previous conditions for inconsistency is produced, n_{UB} is an inconsistent state and it is pruned, otherwise n_{UB} is a new state such that all arcs fixed in n_{UB} are also fixed in any solution reachable from n having cost lower than UB. In order to simplify notation, in the following, n will represent a search state after immediate selection with the current upper bound at the time it was generated.

5.6 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.

Guiding the search by knowledge from the problem domain has important benefits. The sooner good upper bounds being found, the more effective the immediate selection procedure, what leads to reducing the search space, computing more accurate heuristic estimations, reaching better upper bounds sooner and so on.

As it was pointed in Section 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.

6. Heuristic improvement by constraint propagation

To improve the heuristic estimations, we use the following strategy taken from (Carlier & Pinson, 1990) for computing lower bounds for the JSSP. It has also inspired the effective *shaving* technique proposed in (Martin & 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* $n \in \boldsymbol{S}$

$$f_{IS}(n) = \max\{P' | n_{P'} \, is \, inconsistent\}. \tag{10}$$

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.

$$r_i(n_1) \leq r_i(n_2) \, and \, q_i(n_1) \leq q_i(n_2), \forall i \tag{11}$$

$$G_{n_1} \subseteq_{SG} G_{n_2} \tag{12}$$

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 (11) and (12) hold.*

Proof. See Appendix. □

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 (11) and (12) are preserved.*

Proof. Condition (12) trivially holds as the graphs are not modified by the algorithm. To prove that condition (11) 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 Section 5.1, fulfill $r_v(n_1) \leq r_v(n_2)$. Analogous reasoning may be done for the new tails. So, condition (11) 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 (11) and (12). 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 (11), 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 $h_{IS}(n) = f_{IS}(n)-g(n)$.

6.1 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 monotoniciy 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. $\Rightarrow$ 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. $\Rightarrow$ 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 (11) and (12) 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 6. *h_{IS} is monotonic.*

6.2 Consequences of the monotoniciy 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 (10) 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.

6.3 Heuristic refinement

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

Theorem 7. *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*

$$f'_{IS}(n) = \max\{f_{JPS}(n_P)|n_P \, is \, consistent\}. \tag{13}$$

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 8. *$f_{IS}(n) \geq f'_{IS}(n)$, 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.

7. 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 & 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.

7.1 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. So, a larger number of nodes are expected to 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.

7.1.1 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 (Tai01-10, 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 section 6.3 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 section 6.3 has no effect in these experiments.

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

7.1.2 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 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

$$maxArcs(N, M) = M \times \frac{(N-1)^2 + (N-1)}{2}. \tag{14}$$

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 5 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

$$100 \times \frac{h_{IS}(n) - h_{JPS}(n)}{h_{JPS}(n)}. \tag{15}$$

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 6 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 5 and 6 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 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.

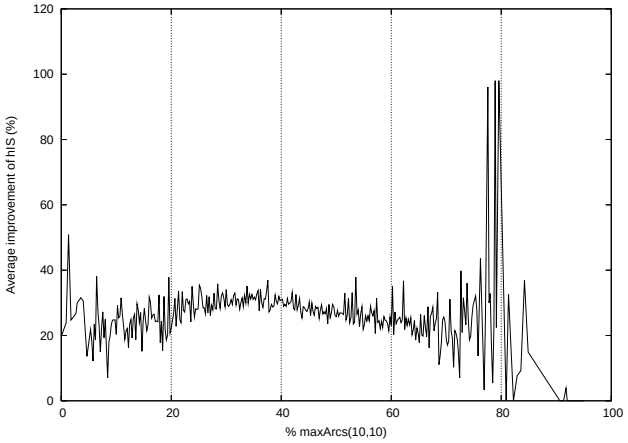


Figure 5: Distribution of heuristic improvements in the effective search space depending on the number of arcs fixed

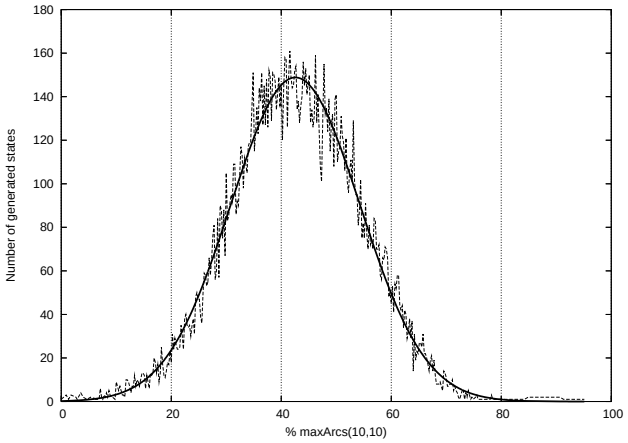


Figure 6: Distribution of states in the effective search space depending on the number of arcs fixed

7.2 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 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 $f(initial)$.

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

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.

		$\%level$ of use of h_{IS} in Depth First						
	BB	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

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.

		%*level* of use of h_{IS} in Depth First						
	BB	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.

		%*level* of use of h_{IS} in Depth First						
	BB	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

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.

		%*level* of use of h_{IS} in Depth First						
	BB	0	20	35	50	65	80	100
E_{UB}	100,00	77,63	64,62	50,93	46,33	42,26	44,72	45,35
E_{LB}	100,00	115,86	91,30	91,30	91,30	91,30	91,30	91,30

the solution to instance $LA38$ 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

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.

		$\%level$ of use of h_{IS} in Depth First						
	BB	0	20	35	50	65	80	100
E_{UB}	100,00	76,86	68,12	71,92	68,36	62,93	61,61	61,81
E_{LB}	100,00	100,00	70,84	70,84	70,84	70,84	70,84	70,84

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 $13,01\%$ 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. 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%.

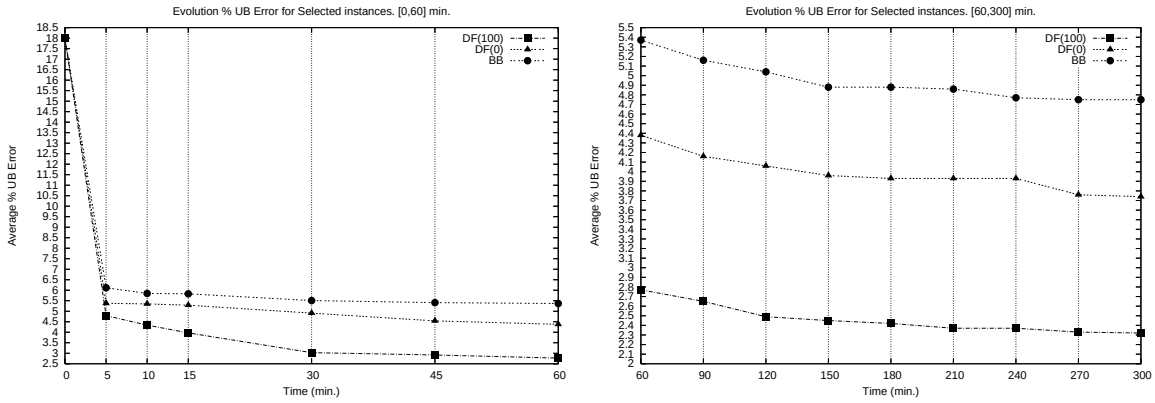
Let us now consider the results after five hours. Tables 5 and 6 summarize these results from the Selected and the 20×15 instances respectively. For the Selected instances, BB reaches an average error of $4,40$ for upper bounds and $2,44$ for lower bounds. For 20×15 instances these errors are $11,63$ 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 $45,45\%$, while it was $52,39\%$ in a time limit of one hour. However, the error in the lower bounds gets worse: $91,30\%$, 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 $LA24$ and $LA25$, and DF also solves the instance $FT20$. While in five hours the instances $LA21$ and $LA38$ get solved by the two algorithms as well, and the instance $LA40$ is solved by BB. For the 20×15 instances, the error in the upper bound gets a little bit worse, $61,81\%$ versus $58,20\%$, 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, Figures 7 and 8 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} ($DF(100)$).

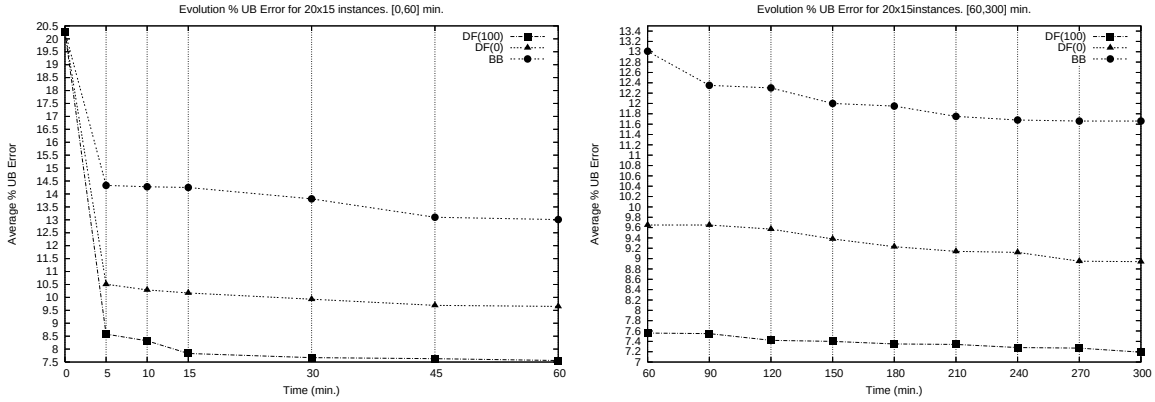
In this case, the values reported are the errors in percentage of the upper bound w.r.t. the best known solution. Figures 7(a) and 8(a) show the evolution along the first hour and Figures 7(a) and 8(b) shows the evolution during the remaining four hours with a different scaling.

For the 20×15 instances, the final values are around $11,66$; $8,94$ and $7,19$ for algorithms BB, DF with h_{JPS} and DF with h_{IS} respectively. So, the relative improvement of DF w.r.t. BB when both of them exploit h_{JPS} is $24,1\%$, and the relative improvement of h_{IS} w.r.t. h_{JPS} when both of them are used in combination with DF is $18,2\%$. The improvement of DF with h_{IS} w.r.t. BB is $42,3\%$. Moreover, in Figure 8 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.



(a) (b)

Figure 7: Evolution % UB Error for Selected instances. Time interval [0, 300] minutes



(a) (b)

Figure 8: Evolution % UB Error for 20x15 instances. Time interval [0, 300] minutes

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 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.

8. Conclusions and future work

We have seen that partially informed depth-first search is more efficient than chronological backtracking when both of them explore the same search space for the JSSP. Even though the latter requires a negligible amount of memory, the first 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 greedy 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, provided that powerful constraint propagation rules and greedy algorithms are 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 JSSP. Also, 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 & Ginsberg, 1995). In the first case, it is expected that better lower bounds may be obtained as well.

We will also try to adapt the algorithms to cope with objective functions other than makespan, such as lateness (Balas, Simonetti, & Vazacopoulos, 2008), total weighted tardiness (Essafi, Mati, & Dauzère-Pérès, 2008) or total flow time (Sierra & Varela, 2010), which are very interesting from the point of view of real-life problems and in some cases they make the problem harder to solve. Finally, we will consider some other variants of scheduling problems closer than JSSP to real environments. For example scheduling problems with operators (Agnetis, Flamini, Nicosia, & Pacifici, 2011), (Mencía, Sierra, Mencía, & Varela, 2011), problems with sequence-dependent setup times (Brucker & Thiele, 1996), (Vela, Varela, & González, 2010), and scheduling problems with uncertain processing times

(Petrovic, Duenas, & Petrovic, 2007), (González Rodríguez, Puente, Vela, & Varela, 2008), (Rodríguez, Vela, Puente, & Hernández-Arauzo, 2009).

Acknowledgments

This research has been supported by the Spanish Ministry of Science and Innovation under research project MICINN-FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP09105.

Appendix A. Proof of Lemma 1

In this appendix we give the proof of Lemma 1.

Lemma 1. *Let n_1 and n_2 be two states fulfilling conditions (11) and (12). 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 (11) and (12) 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 (11) 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 4 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 (11) holds after the procedure Improving Heads. Condition (12) 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 $(i \to j)$ 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 $(i \to j)$ 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. *J. Scheduling*, *14*(3), 225–237.

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

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*. Kluwer Academic Publishers, Norwell, MA, USA.

Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, *41*(11), 1069–1072.

Beck, J. C. (2007). Solution-guided multi-point constructive search for job shop scheduling. *J. Artif. Intell. Res. (JAIR)*, *29*, 49–77.

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

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

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

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

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.

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

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. *Artif. Intell.*, *122*, 189–240.

Essafi, I., Mati, Y., & Dauzère-Pérès, S. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers and Operations Research*, *35*, 2599–2616.

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

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

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*, *38 (3)*, 655–666.

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.

Hart, P., Nilsson, N., & 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., & 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. *Artif. Intell.*, *27*, 97–109.

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

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

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 *IWINAC (2)*, pp. 305–314.

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

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

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

Petrovic, D., Duenas, A., & Petrovic, S. (2007). Decision support tool for multi-objective job shop scheduling problems with linguistically quantified decision functions. *Decision Support Systems*, *43(4)*.

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

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

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.

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. *J. Artif. Intell. Res. (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.

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. *J. 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.

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.

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.