Partially Informed Depth-First Search for the Job Shop Problem

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

Abstract

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

Introduction

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

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

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

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

The Job Shop Scheduling Problem

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

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

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

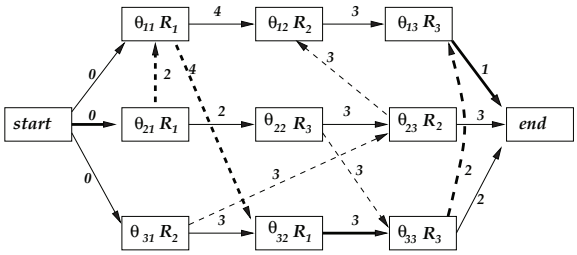

Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines. Bold face arcs show a critical path whose length (makespan) is 12

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

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

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

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

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

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

Brucker's Algorithm

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

A Partially Informed Depth-First Search Algorithm

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

State Representation

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

Heuristic Estimation

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

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

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

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

Upper Bounds

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

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

Expansion Mechanism

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

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

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

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

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

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

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

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

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

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

Fixing Additional Arcs by Constraint Propagation

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

Algorithm 2 PROCEDURE Select

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

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

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

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

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

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

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

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

1) If

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

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

2) If

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

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

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

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

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

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

This fact leads to the following problem.

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

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

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

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

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

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

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

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

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

Search Strategy

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

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

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

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

Heuristic Improvement by Constraint Propagation

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

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

1) G_{n_r} contains a cycle.

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

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

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

Monotonicity of h_{IS}

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

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

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

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

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

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

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

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

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

1) As G_{n_r} is a subgraph of $G_{n'_r}$, if G_{n_r} contains a cycle, then $G_{n'_r}$ contains a cycle as well.

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

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

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

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

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

Analysis of the Effectiveness of h_{IS}

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

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

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

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

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

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

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

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

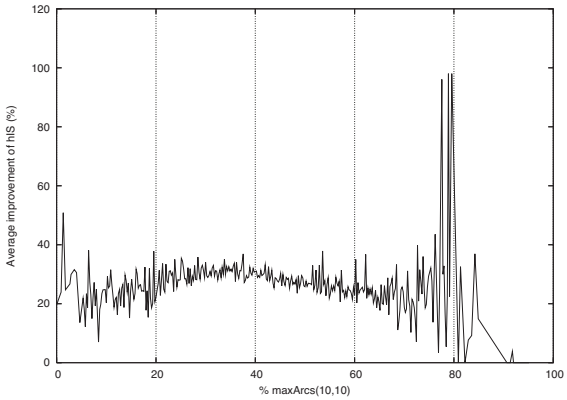

Figure 2: Distribution of heuristic improvements in the effective search space depending on the number of arcs fixed

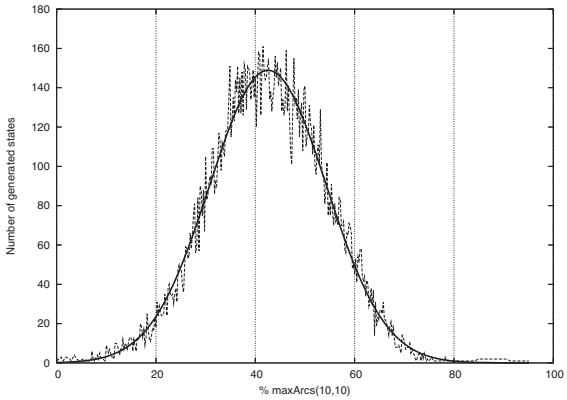

Figure 3: Distribution of states in the effective search space depending on the number of arcs fixed

Table 1: Results for instances of size 10×10

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

Table 2: Results for selected instances

		DF						
	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 3: Results for instances of size 20×15

		DF						
	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

Computational Results

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

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

Table 2 shows the results from the second set of instances. In this case all algorithms were run for 1 hour. The average error reached by BB is $5,37\%$ for upper bounds and $3,42\%$ for lower bounds with respect to the best known bounds. As we can observe, DF is better than BB in all cases. When h_{IS} is used, the improvement in lower bounds is about 35% in all cases, independently of the level up to h_{IS} is applied, with exception of the level 20%. This is due to the fact that the instance $LA38$ is solved optimally in this case and so the lower bound is much better than in the remaining ones. The fact that the lower bound is almost the same for all levels is quite reasonable due to the depth-first search. The lower bound is the lowest $f(\cdot)$ of a node in the open list and, as the instances are very large, this value might correspond to a

successor of the initial state due to the fact that the algorithm is not able to expand all of these successors after one hour. For the same reason the lower bound at level 0% is the same as that of BB. However, the quality of the upper bounds improves in direct ratio with the level up to h_{IS} is applied. In this case it is worth to exploit the improved heuristic beyond level 50%. In this case the improvement with respect to BB is about 50%. Moreover, BB does not reach an optimal solution to the instance FT20, whereas DF solves it in just a few seconds using h_{IS}.

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

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

Conclusions

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

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

Acknowledgments

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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