PRUNING SEARCH SPACE BY DOMINANCE RULES IN BEST FIRST SEARCH FOR THE JOB SHOP SCHEDULING PROBLEM

María R. Sierra
Dept. of Mathematics, Statistics and Computing, University of Cantabria
Facultad de Ciencias, Avda. de los Castros, E-39005 Santander, Spain

Ramiro Varela
Artificial Intelligence Center, Dept. of Computing, University of Oviedo
Campus de Viesques, E-33271 Gijón, Spain

Keywords: Heuristic Search, Best First Search, Pruning by Dominance, Job Shop Scheduling.

Abstract: Best-first graph search is a classic problem solving paradigm capable of obtaining exact solutions to optimization problems. As it usually requires a large amount of memory to store the effective search space, in practice it is only suitable for small instances. In this paper, we propose a pruning method, based on dominance relations among states, for reducing the search space. We apply this method to an A^* algorithm that explores the space of active schedules for the Job Shop Scheduling Problem with makespan minimization. The A^* algorithm is guided by a consistent heuristic and it is combined with a greedy algorithm to obtain upper bounds during the search process. We conducted an experimental study over a conventional benchmark. The results show that the proposed method is able to reduce both the space and the time in searching for optimal schedules so as it is able to solve instances with 20 jobs and 5 machines or 9 jobs and 9 machines. Also, the A^* is exploited with heuristic weighting to obtain sub-optimal solutions for larger instances.

1 INTRODUCTION

In this paper we propose a method based on dominance properties to reduce the effective space in best-first search. The method is illustrated with an application of the A^* algorithm (Hart et al., 1968; Nilsson, 1980; Pearl, 1984) to the Job Shop Scheduling Problem (JSSP) with makespan minimization. We establish a sufficient condition for a state n_1 dominates another state n_2 so as n_2 can be pruned. Also, we have devised a rule to evaluate this condition efficiently. The overall result is a substantial reduction in both the time and mainly in the space required for searching optimal schedules.

Over the last decades, a number of methods has been proposed in the literature to deal with the JSSP with makespan minimization. In particular there are some exact methods such as the branch and bound algorithm proposed in (Brucker et al., 1994) or the backtracking algorithm proposed in (Sadeh and Fox, 1996).

As the majority of the efficient methods for the JSSP with makespan minimization, the Brucker's algorithm relies on the concept of critical path, i.e. a longest path in the solution graph representing the processing order of operations in a solution. In particular, the branching schema is based on reversing orders on the critical path. The main problem of the methods based on the critical path is that they can not be efficiently adapted to objective functions other than makespan.

The algorithm proposed in (Sadeh and Fox, 1996) is guided by variable and value ordering heuristics and its branching schema is based on starting times of operations. It is not as efficient as the Brucker's algorithm for makespan minimization, but it can be easily adapted for other classic objective functions such as total flow time or tardiness minimization. In this paper, we consider the search space of active schedules in order to evaluate the proposed method for pruning by dominance. This search space is suitable for any objective function.

The paper is organized as follows. In section 2 the JSSP is formulated. Section 3 describes the search space of active schedules for the JSSP. Section 4 sumarizes the main characteristics of A^* algo-

R. Sierra M. and Varela R. (2008).
PRUNING SEARCH SPACE BY DOMINANCE RULES IN BEST FIRST SEARCH FOR THE JOB SHOP SCHEDULING PROBLEM.
In *Proceedings of the Third International Conference on Software and Data Technologies - PL/DPS/KE*, pages 273-280
DOI: 10.5220/0001896102730280

rithm. In section 5, the heuristic used to guide A^* for the JSSP with makespan minimization is described. Section 6 introduces the concepts of dominance and establishes some results and an efficient rule to test dominance for the JSSP. Section 7 reports results from the experimental study. Finally, section 8 summarizes the main conclusions and outlines some ideas for future research.

2 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 constraints: precedence, capacity and no-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 $R_v = R_w$. No-preemption requires that the machine is assigned to an operation without interruption during its whole processing time. The objective is to come up with a feasible schedule such that the completion time, i.e. the *makespan*, is minimized.

In the sequel 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 weighed 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 is represented by an acyclic subgraph G_s of G, $G_s = (V, A \cup H)$, where $H = \cup_{i=1,\ldots,M} H_i$, H_i being a processing ordering for the operations requiring R_i. The makespan is the cost of a *critical path*. A critical path is a longest path from node *start* to node *end*.

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 nodes of v respectively on its job.

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

$$\begin{aligned} r_v &= \max\{\max_{w\in P(v)}(r_w + p_w), r_{PJ_w} + p_{PJ_w}\} \\ q_v &= \max\{\max_{w\in S(v)}(p_w + q_w), p_{SJ_v} + q_{SJ_v}\} \end{aligned} \quad (1)$$

where $P(v)$ denotes the disjunctive predecessors of v, i.e. operations requiring machine R_v which are scheduled before v. Analogously, $S(v)$ denotes the disjunctive successors of v. Hence, the value $r_v + p_v + q_v$ is a lower bound of the best schedule that can be reached from the partial schedule. This lower bound may be improved from the Jackson's preemptive schedule, as we will see in section 5.

3 THE SEARCH SPACE OF ACTIVE SCHEDULES

A schedule is *active* if for an operation can start earlier at least another one should be delayed. Maybe the most appropriate strategy to calculate active schedules is the *G&T* algorithm proposed in (Giffler and Thomson, 1960). This is a greedy algorithm that produces an active schedule in a number of $N * M$ steps.

At each step *G&T* makes a non-deterministic choice. Every active schedule can be reached by taking the appropriate sequence of choices. Therefore, by considering all choices, we have a complete search tree for strategies such as branch and bound, backtracking or A^*. This is one of the usual branching schemas for the JSSP, as pointed in (Brucker and Knust, 2006), and it is the approach taken, for example, in (Varela and Soto, 2002) and (Sierra and Varela, 2005).

Algorithm 1 shows the expansion operation that generates the full search tree when it is applied successively from the initial state, in which none of the operations are scheduled yet.

In the sequel, we will use the following notation. Let O denote the set of operations of a problem instance, and n_1 and n_2 be two search states. In n_1, O can be decomposed into the disjoint union $SC(n_1) \cup US(n_1)$, where $SC(n_1)$ denotes the set of operations scheduled in n_1 and $US(n_1)$ denotes the unscheduled

Algorithm 1: SUC(state n). Algorithm to expand a state n. When it is successively applied from the initial state, i.e. an empty schedule, it generates the whole search space of active schedules.

1. $A = \{v \in US(n);\ PJ_v \in SC(n)\}$;
2. Let $v \in A$ the operation with the lowest completion time if it is scheduled next, that is $r_v + p_v \leq r_u + p_u, \forall u \in A$;
3. $B = \{w \in A;\ R_w = R_v$ and $r_w < r_v + p_v\}$;

for each $w \in B$ **do**

4. $SC(n') = SC(n) \cup \{w\}$ and $US(n') = US(n) \backslash \{w\}$;
   $\backslash *$ *w gets scheduled in the current state* $n' * \backslash$
5. $G_{n'} = G_n \cup \{w \rightarrow v;\ v \in US(n'), R_v = R_w\}$;
   $\backslash *$ *st_w is set to r_w in n' and the arc(w,v) is added to the partial solution graph* $* \backslash$
6. $c(n,n') = max\{0, (r_w + p_w) - max\{(r_v + p_v), v \in SC(n)\}\}$;
7. Update heads of operations in $US(n')$ accordingly with expression (1);
8. Add n' to successors;

end for

9. return successors;

ones. $D(n_1) = |SC(n_1)|$ is the depth of node n_1 in the search space. Given $O' \subseteq O$, $\boldsymbol{r}_{n_1}(O')$ is the vector of heads of operations O' in state n_1. $\boldsymbol{r}_{n_1}(O') \leq \boldsymbol{r}_{n_2}(O')$ iff for each operation $v \in O'$, $r_v(n_1) \leq r_v(n_2)$, $r_v(n_1)$ and $r_v(n_2)$ being the head of operation v in states n_1 and n_2 respectively. Analogously, $\boldsymbol{q}_{n_1}(O')$ is the vector of tails.

4 BEST-FIRST SEARCH

For best-first search we have chosen the A^* Nilsson's algorithm (Hart et al., 1968; Nilsson, 1980; Pearl, 1984). A^* starts from an initial state s, a set of goal nodes Γ and a transition operator SUC such that for each node n of the search space, $SUC(n)$ returns the set of successor states of n. Each transition from n to n' has a positive cost $c(n,n')$. $P^*_{s\text{-}n}$ denotes the minimum cost path from node s to node n. The algorithm searches for a path $P^*_{s\text{-}o}$ with the optimal cost, denoted C^*.

The set of candidate nodes to be expanded are maintained in an ordered list $OPEN$. The next node to be expanded is that with the lowest value of the evaluation function f, defined as $f(n) = g(n) + h(n)$; where $g(n)$ is the minimal cost known so far from s to n, (of course if the search space is a tree, the value of $g(n)$ does not change, otherwise this value has to be updated as long as the search progresses) and $h(n)$ is a heuristic positive estimation of the minimal distance from n to the nearest goal.

If the heuristic function underestimates the actual minimal cost, $h^*(n)$, from n to the goals, i.e. $h(n) \leq h^*(n)$, for every node n, the algorithm is admissible, i.e. it returns an optimal solution. Moreover, if $h(n_1) \leq h(n_2) + c(n_1,n_2)$ for every pair of states n_1, n_2 of the search graph, h is consistent. Two of the properties of consistent heuristics are that they are admissible and that the sequence of values $f(n)$ of the expanded nodes is non-decreasing.

The heuristic function $h(n)$ represents knowledge about the problem domain, therefore as long as h approximates h^* the algorithm is more and more efficient as it needs to expand a lower number of states to reach the optimal solution.

Even with consistent and well-informed heuristics, the cost of the search becomes prohibitive for not-too-large instances. In that case, it is possible to relax the requirement of admissibility and modify the algorithm to obtain near optimal solutions. Maybe, the most common technique to do that is dynamic weighting of the heuristic h. The rationale behind weighting is to enlarge the value of $h(n)$ so as it is closer to $h^*(n)$. In order to do that it is common to use an evaluation function of the form proposed in (Pohl, 1973)

$$f(n) = g(n) + P(n)h(n) \tag{2}$$

where $P(n) \geq 1$ is the weighting factor; this factor may be calculated as

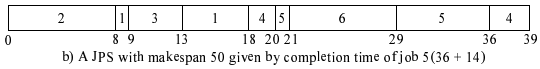
$$P(n) = 1 + K(1 - d(n)/D) \tag{3}$$

where $K \geq 0$ is a parameter and $d(n)$ and D are the depth of node n in the search space and the maximum depth of a node respectively. With dynamic weighting it is expected that the number of nodes expanded to reach a solution is lower than that with the original A^*, but the admissibility is not preserved; however the cost of the first solution state reached is not larger than $C^*(1+K)$. As this node is not usually optimal, it makes sense to leave A^* searching for more solutions after the first one.

Also, best first search may be combined with greedy algorithms to obtain upper bounds during the search. For example, just before to expand a node n, the greedy algorithm can be run to solve the subproblem represented by n. If this process results to be very time consuming, greedy algorithm may be run with a small probability. This is the approach taken in our experimental study.

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

a) An OMS problem instance



b) A JPS with makespan 50 given by completion time of job 5(36 + 14)

Figure 1: The Jackson's Preemptive Schedule for an OMS problem instance.

5 A HEURISTIC FOR THE JSSP

Here, we use a heuristic for the JSSP based on problem relaxations. The residual problem represented by a state n is given by the unscheduled operations in n together with their heads and tails, i.e. the triplet $J(n) = (US(n),\ \mathbf{r}_n(US(n)),\ \mathbf{q}_n(US(n)))$. In state n a number of jobs in J have all their operations scheduled, whilst the remaining ones have some operations not scheduled yet, these subsets of J will be denoted as

$$J_{US}(n) = \{J_i \in J; \exists j, 1 \leq j \leq M, \theta_{ij} \in US(n)\}$$
$$J_{SC}(n) = J \backslash J_{US}(n) \tag{4}$$

Also, we denote by $C_{max}(J_{SC}(n))$ to the maximum completion time of jobs in $J_{SC}(n)$, i.e.

$$C_{max}(J_{SC}(n)) = max\{r_{\theta_{iM}}(n) + p_{\theta_{iM}}, J_i \in J_{SC}(n)\} \tag{5}$$

with $C_{max}(J_{SC}(n)) = 0$ if $J_{SC}(n) = \varnothing$.

A problem relaxation can be made in the following two steps. Firstly, for each machine m required by at least one operation in $US(n)$, the simplified problem $J(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m),\ \mathbf{q}_n(US(n)|_m))$ is considered, where $US(n)|_m$ denotes the unscheduled operations in n requiring machine m. Problem $J(n)|_m$ is known as the One Machine Sequencing (OMS) with heads and tails, where an operation v is defined by its head r_v, its processing time p_v over machine m, and its tail q_v. This problem is still *NP-hard*, so a new relaxation is made: the no-preemption of machine m. This way an optimal solution to this problem is given by the Jackson's preemptive schedule (JPS) (Carlier and Pinson, 1989; Carlier and Pinson, 1994).

Figure 1 shows an example of OMS instance and a JPS for it. 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 jobs are completely scheduled, schedule the ready operation with the largest tail on machine m. Carlier and Pinson proved in (Carlier and Pinson, 1989; Carlier and Pinson, 1994) that calculating the JPS has a complexity of $O(K \times log_2(K))$, where K is the number of operations.

The JPS of problem $J(n)|_m$ provides a lower bound of $f^*(n)$ due to the fact that the heads of operations of $US(n)|_m$ are adjusted from the scheduled operations $SC(n)$. So, taking the largest of these values over machines with unscheduled operations and taking into account the value $C_{max}(J_{SC}(n))$, a lower bound of $f^*(n)$ is obtained. Then, to obtain a lower bound of $h^*(n)$, the value of the largest completion time of operations in $SC(n)$, i.e. $g(n)$, should be discounted and the heuristic, termed h_{JPS}, is calculated as

$$h_{JPS}(n) = max\{C_{max}(J_{SC}(n)), JPS(J(n))\} - g(n)$$
$$JPS(J(n)) = \max_{m \in R}\{JPS(J(n)|_m)\} \tag{6}$$

As h_{JPS} is devised from a problem relaxation, it is consistent (Pearl, 1984).

6 DOMINANCE PROPERTIES

Given two states n_1 and n_2, we say that n_1 dominates n_2 if and only if the best solution reachable from n_1 is better, or at least of the same quality, than the best solution reachable from n_2. In some situations this fact can be detected and then the dominated state can be early pruned.

Let us consider a small example. Figure 2 shows the Gantt charts of two partial schedules, with three operations scheduled, corresponding to search states for a problem with 2 jobs and 3 or more machines. If the second operation of job J_1 requires R_2 and the third operation of J_2 requires R_3, it is easy to see that the best solution reachable from the state of Figure 2a can not be better than the best solution reachable from the state of Figure 2b. This is due to the residual problem of both states comprising the same set of operations and in the first state the heads of all opera-

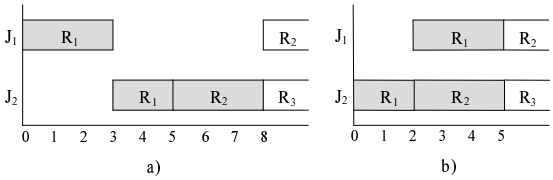


Figure 2: Partial schedules of two search states, state b) dominates state a).

tions are larger or at least equal than the heads in the second state. So, the state of Figure 2a may be pruned if both states are simultaneously in memory.

Of course, a good heuristic will lead the search to explore first the state of Figure 2b if both of them are in *OPEN* at the same time. However, at a later time, the state of Figure 2a and a number of its descendants might also be expanded. Consequently, early pruning of this state can reduce the space and, if the comparison of states for dominance is done efficiently, also the search time.

Pruning by dominance is not new in heuristic search. For example, in (Nazaret et al., 1999) a similar method is proposed for the Resource Constrained Project Scheduling Problem (RCPSP), but no clear rules are given to apply it during the search; and in (Korf, 2003) and (Korf, 2004) various rules are proposed for the Bin Packing Problem and the two-dimensional Cutting Stock Problem respectively that allow pruning some of the siblings of a node *n* at the time of expanding this node.

More formally, we define dominance among states as it follows.

Definition 1. *Given two states n_1 and n_2, such that $n_1 \notin P^*_{s\text{-}n_2}$ and $n_2 \notin P^*_{s\text{-}n_1}$, n_1 dominates n_2 if and only if $f^*(n_1) \leq f^*(n_2)$.*

Of course, establishing dominance among any two states is problem dependent and it is not easy in general. Therefore, to define an efficient strategy, it is not possible to devise a complete method to determine dominance and apply it to every pair of states of the search space. So, what we have done is establishing a sufficient condition for dominance for the JSSP with makespan minimization. As we will see, this condition can be efficiently evaluated, so as the whole process of testing dominance is efficient, at the cost of not detecting all dominated states.

Theorem 1. *Let n_1 and n_2 be two states such that $US(n_2) = US(n_1) = US$, $f(n_1) \leq f(n_2)$ and $\mathbf{r}_{n_1}(US) \leq \mathbf{r}_{n_2}(US)$, then the following conditions hold:*

1. $\boldsymbol{q}_{n_1}(US) = \boldsymbol{q}_{n_2}(US)$.
2. *n_1 dominates n_2.*

Proof 1. *Condition* 1 *comes from the fact that each operation $v \in US$ is an unscheduled operation in both states n_1 and n_2 and so it has not any disjunctive successor yet. So, according to equations (6), $q_v(n_1) = p_{SJ_v} + q_{SJ_v}(n_1)$ and $q_v(n_2) = p_{SJ_v} + q_{SJ_v}(n_2)$. As $q_{end}(n_1) = q_{end}(n_2) = 0$, reasoning by induction from node end backwards, we have finally $q_v(n_1) = q_v(n_2)$. Hence, $\mathbf{q}_{n_1}(US) = \mathbf{q}_{n_2}(US)$.*

To prove condition 2 *we can reason as follows. Let us denote as $C^*_{max}(J(n))$ to the optimal makespan of subproblem $J(n)$. Hence, for a state n, $f^*(n) = max\{C_{max}(J_{SC}(n)), C^*_{max}(J(n))\}$, $f(n) = max\{C_{max}(J_{SC}(n)), JPS(J(n))\}$, being $JPS(J(n)) \leq C^*_{max}(J(n))$.*

From $\mathbf{r}_{n_1}(US) \leq \mathbf{r}_{n_2}(US)$ and $\mathbf{q}_{n_1}(US) = \mathbf{q}_{n_2}(US)$ it follows that $C^*_{max}(J(n_1)) \leq C^*_{max}(J(n_2))$, as every schedule for problem $J(n_2)$ is also a schedule for $J(n_1)$, and for analogous reason considering preemtive schedules, it also follows that $JPS(J(n_1)) \leq JPS(J(n_2))$. From this result and $f(n_1) \leq f(n_2)$ it follows that

(a) $C_{max}(n_1) \leq C_{max}(n_2)$ or

(b) $C_{max}(n_1) > C_{max}(n_2)$ and $C_{max}(n_1) \leq JPS(J(n_2))$.

In the case (a) as $f^*(n_1) = max\{C_{max}(J_{SC}(n_1)), C^*_{max}(J(n_1))\}$, analogous for $f^*(n_2)$, it follows that $f^*(n_1) \leq f^*(n_2)$.

In the case (b) $f^*(n_2) = C^*_{max}(J(n_2))) \geq max\{C_{max}(J_{SC}(n_1)), C^*_{max}(J(n_1))\} = f^*(n_1)$. So n_1 dominates n_2.

6.1 Rule for Testing Dominance

From the results above, we can devise rules for testing dominance to be included in the A^* algorithm. In principle each time a new node n_1 appears during the search, this node could be compared with any other node n_2 reached previously. In this comparison, it should be verified if n_1 dominates n_2 and also if n_2 dominates n_1. If one of the nodes is dominated, it can be pruned. It could be the case that both n_1 dominates n_2 and n_2 dominates n_1; in this case either of them, but not both, can be pruned.

Obviously, this rule does not seem very efficient. So, in order to reduce the number of evaluations, we proceed as follows:

1. Each time a node n is selected by A^* for expansion, n is compared with every node n' in *OPEN* such that $D(n_1) = D(n_2)$ and $f(n) = f(n')$. If any of the nodes become dominated, it is pruned. In the case that both n dominates n' and n' dominates n, n' is pruned.
2. If node n is not pruned in step 1, it is compared with those nodes n' in the *CLOSED* list such that $US(n') = US(n)$ ($f(n') \leq f(n)$ as a consequence of the consistency of the heuristic h_{JPS}). If n' dominates n, then n is pruned.

In step 1, n is not compared with nodes n' in *OPEN* with $f(n) < f(n')$. In this situation, n could dominate n', but this will be detected later if n' is selected for expansion, as n will be in *CLOSED*. In step 1 nodes n' with $D(n') = D(n)$ in *OPEN* are efficiently searched as *OPEN* is organized as an array

with a component for each possible depth, from 1 to $N * M$, and in each component a list of nodes sorted by f values is stored. Similarly, in step 2, nodes n' with $US(n') = US(n)$ may be efficiently searched in $CLOSED$ as this structure is organized as a hash table with the hash function returning the set of unscheduled operations $US(n)$ for each state n.

7 EXPERIMENTAL STUDY

For experimental study we have chosen two sets of instances taken from the *OR-library* (*http://people.brunel.ac.uk/∼mastjjb/jeb/info.html*). First we have chosen 6 instances of size 20×5 (20 jobs and 5 machines): $LA01$ to $LA05$ and $FT20$. Then we have chosen instances of size 10×10. The reason for these selection is that these sizes are in the threshold of what our approach is able to solve. We used an A^* prototype implementation coded in C++ language developed in Builder C++ 6.0 for Windows, the target machine was Pentium 4 at 3*Ghz* with 2*Gb RAM*.

To evaluate the efficiency of the proposed pruning method, we first solved these instances without considering upper bounds. So, none of the generated states n can be pruned from the condition $f(n) \geq UB$ and these nodes should be inserted in the $OPEN$ list, even though they will never be expanded due to heuristic h_{JPS} being admissible. Moreover, in this case A^* only completes the search either when a solution state is reached or when the computational resources (memory available or time limit) are exhausted. This allows us to estimate the size of the search space for these instances. We have given a time limit of 3600 seconds for each run.

Columns 2 to 5 of Table 1 summarizes the results of this experiment. As we can observe, when pruning is not applied, instances $LA11$ and $LA13$ remain unsolved due to memory getting exhausted. On the other hand, when pruning is applied 5 of the six instances get solved and the number of expanded nodes is much lower in all 5 cases. Also, the time taken is lower.

In the second experiments, we have enhanced A^* by calculating upper bounds by means of a greedy algorithm. As it was done in (Brucker et al., 1994; Brucker, 2004) we have used the $G\&T$ algorithm with a selection rule based on JPS computations restricted to the machine required by critical operations, i.e. those of set B in Algorithm 1. Here, with a given probability P, a solution is issued from the expanded node. Columns 6 and 7 of Table 1 reports results from a set of experiments with $P = 0,01$; in this case the results are averaged over 20 runs. As we can observe, in this case all 6 instances get solved, being both the time taken and the number of expanded nodes less than they are in the experiments without upper bounds calculation.

Table 1: Summary of results of pruning by dominance over instances $LA01$-15 and $FT20$. The last two columns show results combining pruning by dominance with probabilistic calculation of heuristic solutions during the search (the heuristic algorithm is run from the initial state and then for each expanded state with probability $P = 0.01$, the results are averaged over 20 runs for each instance.

	No Pruning		Pruning		Pruning + UB $P = 0,01$	
Inst.	Exp.	T.(s)	Exp.	T.(s)	Exp.	T.(s)
$LA11$	**131470**	**143**	**105449**	**272**	1	0
$LA12$	1689	1	965	2	127	1
$LA13$	**111891**	**141**	13599	33	10206	25
$LA14$	258	0	257	0	1	0
$LA15$	76967	93	22068	46	22066	46
$FT20$	9014	7	2756	4	2753	5

bold *indicates memory getting exhausted.*

In the third series of experiments we apply the same method to a set of instances with size 9×9 obtained from the ORB set by eliminating the last job and the last machine. The results are reported in Table 2. As we can observe, when pruning is not applied only 3 out of the 10 instances get solved; while 7 instances get solved with pruning and for the 3 previously solved the number of expanded nodes is much lower as well. However in this case the effect of the greedy algorithm is almost null for the instances solved. But for the 3 instances unsolved, it seems that the greedy algorithm allows to prevent many states to be included in $OPEN$, as the memory gets exhausted after a larger number of expanded nodes.

In the last series of experiments, we have considered the original ORB set with instances of size 10×10. As only one of these instances gets solved to optimality with the exact algorithm, we applied the heuristic weighting with $K = 0,01$. In this case, A^* is not stopped after reaching a solution state, but it runs until the memory gets exhausted or the $OPEN$ list gets empty. Table 3 summarizes the results of these experiments. As we can observe, only for instance 10 the $OPEN$ list gets empty and so the optimal solution is reached. For the remaining instances the memory gets exhausted before reaching the optimal solution, being the mean error in percent 2,86. This error is much larger than that expected from the weighted heuristic. The reason for this is that with $K = 0,01$ A^* never reaches a solution node and the solution returned is

Table 2: Summary of results from $ORBR(9 \times 9)$ instances obtained from reduction of *ORB* instances. The results of the last two columns are averaged for 20 runs.

	No Pruning		Pruning		Pruning + UB $P = 0,01$	
Inst	Exp.	T.(s)	Exp.	T.(s)	Exp.	T.(s)
1	229245	133	36043	42	36043	45
2	268186	149	31714	31	31708	34
3	**467267**	**333**	**265217**	**1121**	**315933**	**1745**
4	**494016**	**329**	79629	116	79628	122
5	**588378**	**332**	**278995**	**738**	**379328**	**1320**
6	**430959**	**320**	182174	454	181384	457
7	561617	335	74528	164	74192	165
8	**427836**	**315**	**260231**	**1448**	**350872**	**2301**
9	**614638**	**352**	272595	947	271024	950
10	**525388**	**325**	106407	165	102494	158

bold *indicates memory getting exhausted.*

the best one reached by the greedy algorithm.

Overall, we can conclude that the proposed method for pruning by dominance allows to reduce drastically the size of the effective search space; being this reduction more relevant for the most difficult problems. As we can observe in Table 2 for the instances that get solved in both cases, i.e. with pruning and without it, the number of expanded states is reduced almost in an order of magnitude when pruning is exploited. This reduction is less significative for instances of size 20×5, as it is shown is Table 1, which are easier to solve. However, the effect of the greedy algorithm over these instances is clearly more significative than it is over the instances of size 9×9. This is due to the fact that the heuristic estimations obtained from the Jackson's Preemptive Schedules are much more accurate for 20×5 instances than they are for 9×9 ones. Hence, both the greedy algorithm and the A* itself are much more efficient as they are guided by better heuristic knowledge. These results agree with those of other experiments, not reported here, that we have done with less informed heuristics. In this case the reduction of the effective search space for instances 20×5 was also of almost an order of magnitude, but in any case the performance was worse than that of heuristic h_{JPS}. Hence we conjecture that the effect of the pruning by dominance is in inverse ratio with the knowledge of the heuristic estimation, so it may be especially interesting for complex problems where the current heuristics are not very much accurate, as it is the case of the RCPSP.

Table 3: Summary of results combining pruning by dominance, UB calculation ($P = 0,01$) and heuristic weighting ($K = 0,01$), over $ORB(10 \times 10)$ instances.

Instance	Optimum	Best found	Exp. nodes	T.(s)
*ORB*01	1059	**1078**	**193019**	**206**
*ORB*02	888	**915**	**207282**	**198**
*ORB*03	1005	**1071**	**156886**	**225**
*ORB*04	1005	**1052**	**161222**	**199**
*ORB*05	887	**893**	**209521**	**202**
*ORB*06	1010	**1050**	**189990**	**200**
*ORB*07	397	**405**	**216814**	**203**
*ORB*08	899	**917**	**257795**	**213**
*ORB*09	934	**970**	**180866**	**202**
*ORB*10	944	944	22775	23

bold *indicates memory getting exhausted*

8 CONCLUSIONS

In this paper we propose a pruning method based on dominance relations among states to improve the efficiency of best-first search algorithms. We have applied this method to the JSSP considering the search space of active schedules and the A^* algorithm. To do that, we have defined a sufficient condition for dominance and a rule to evaluate this condition which is efficient as it allows to restrict comparison of the expanded node with only a fraction of nodes in *OPEN* and *CLOSED* lists. This method is combined with a greedy algorithm to compute upper bounds during the search. We have reported results from an experimental study over instances taken from the *OR-library*.

These experiments show that the proposed method of pruning by dominance, combined with the greedy algorithm to obtain upper bounds during the search process, is efficient as it allows to save both space and time. Also we have combined this method with a weighting heuristic method that allows to obtain non-optimal solutions for large instances.

In comparison with other methods, our approach is more efficient than the backtracking algorithm proposed in (Sadeh and Fox, 1996), which is not able to solve instances of size 10×5; but it is less efficient than the the branch and bound algorithm described in (Brucker et al., 1994; Brucker, 2004), which is able to solve instances of size 10×10 or even larger. Only one of the instances considered in our experimental study, the $FT20$, can not be solved to optimality by this algorithm.

The Brucker's algorithm exploits a sophisticated branching schema based on the concept of critical path which is not applicable to objective functions other than the makespan. However, the branching schema based on *G&T* algorithm is also suitable for objective functions such as total flow time or tardiness. So it is expected that our approach is to be effi-

cient in these cases as well.

As future work, we plan to improve our approach with better heuristic estimations, new pruning rules and more efficient greedy algorithms to obtain upper bounds. Also, we plan to combine the pruning strategy with constraint propagation techniques, such as those proposed in (Dorndorf et al., 2000; Dorndorf et al., 2002), as it is done in the branch and bound algorithm described in (Brucker et al., 1994; Brucker, 2004).

It would be also interesting to apply the pruning by dominance method to other search spaces for the JSSP with makespan minimization and to other scheduling problems which are harder to solve such as the JSSP with total flow time or tardiness minimization; and the the JSSP with setup times.

We will confront other problems such as the Travelling Salesman Problem, the Cutting-Stock Problem or the RCPSP. As search spaces of these problems have similar characteristics to the space of active schedules for the JSSP, we expect to obtain similar improvement of efficiency in all cases.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish Ministry of Science and Education under research project TIN2007-67466-C02-01. The authors thank the anonymous referees for their suggestions which have contributed to improve the paper.

REFERENCES

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

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. and Knust, S. (2006). *Complex Scheduling*. Springer.

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

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

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2002). Constraint propagation and problem descomposition: A preprocessing procedure for the job shop problem. *Annals of Operations Research*, 115:125–142.

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

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

Korf, R. (2003). An improved algorithm for optimal bin-packing. In *Proceedings of the 13th International Conference on Artificial Intelligence (IJCAI03)*, pages 1252–1258.

Korf, R. (2004). Optimal rectangle packing: New results. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS04)*, pages 132–141.

Nazaret, T., Verma, S., Bhattacharya, S., and Bagchi, A. (1999). The multiple resource constrained project scheduling problem: A breadth-first approach. *European Journal of Operational Research*, 112:347–366.

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

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

Pohl, I. (1973). The avoidance of relative catastrophe, heuristic competence, genuine dynamic weigting and computational issues in heuristic problem solving. In *Proceedings of IJCAI73*, pages 20–23.

Sadeh, N. and Fox, M. S. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86:1–41.

Sierra, M. and Varela, R. (2005). Optimal scheduling with heuristic best first search. *Proceedings of AI*IA'2005, Lecture Notes in Computer Science*, 3673:173–176.

Varela, R. and Soto, E. (2002). Sheduling as heuristic search with state space reduction. *Lecture Notes in Computer Science*, 2527:815–824.