Pruning Search Spaces by Dominance Rules: A Case Study in the Job Shop Scheduling

María R. Sierra* and Ramiro Varela**

*Department of Mathematics, Statistics and Computing, University of Cantabria,
Avda. de los Castros s/n, 39005 Santander, Spain
Tel: +34-4-2202201, Fax: +34-8-5182125
E-mail: sierramr@unican.es
**Department of Computing,
Artificial Intelligence Center, University of Oviedo,
Campus of Viesques s/n, 33271 Gijón, Spain
Tel: +34-8-5182032, Fax: +34-8-5182125
E-mail: {mariasierra, ramiro}@aic.uniovi.es
*Corresponding author

Abstract: In this paper, we propose a pruning method, based on dominance relations among states, for reducing the search space in best-first search. We apply this method to an A* algorithm that explores the space of active schedules for the Job Shop Scheduling Problem with makespan minimization. We conducted an experimental study over conventional benchmarks. The results show that the proposed method is able to reduce both the space and the time in searching for optimal schedules and that it outperforms other approach taken from the literature.

Keywords: job shop scheduling, heuristic search, best first search, pruning by dominance

Reference to this paper should be made as follows: Sierra, M.R. and Varela, R.(xxxx) 'Pruning Search Spaces by Dominance Rules: A Case Study in the Job Shop Scheduling', *Int. J. of Reasoning-based Intelligent Systems*, Vol. x, No. x, pp.xxx–xxx.

Biographical Notes: María R. Sierra received her Graduate in Computer Science at the University of Oviedo. She has done her doctoral studies at this University and now she is Assistant Lecturer at the Department of Maths, Statistic an Computing of the University of Cantabria. Her research interests include scheduling problems and heuristic search. She has published research papers at national and international journals, conference proceedings as well as books chapters.

Ramiro Varela received his PhD in Computer Science at the University of Oviedo. He is an Associate Professor with this University at the Department of Computer Science and member of the Artificial Intelligence Centre. His research interests include heuristic search and evolutionary computation. He has published research papers at national and international journals, conference proceedings as well as books chapters.

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 (1968); Nilsson (1980); Pearl (1984)] to the Job Shop Scheduling Problem (JSSP) with makespan minimization. We established 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 (1994) or the backtracking algorithm proposed in Sadeh (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 the order of operations 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 (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. For this reason we have chosen to compare with the method proposed in Sadeh (1996) in the experimental study.

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^* algorithm. In section 5, the heuristic used to guide A^* for the JSSP is described. Section 6 introduces the concept 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.

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

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

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 (see 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 (1960). This is a greedy algorithm that produces an active schedule in a number of $N * M$ steps. At each step, $G\&T$ algorithm 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 suitable 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 (2006), and it is the approach taken, for example, in Varela (2002) and Sierra (2005).

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, 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 now scheduled in n' to r_w and the arc(w, v) is added to the 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;

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 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 Nilsson (1980). 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 $P^*_{s\text{-}o}$ which has the lower cost to achieve an objective. 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.

5 A Heuristic for the JSSP

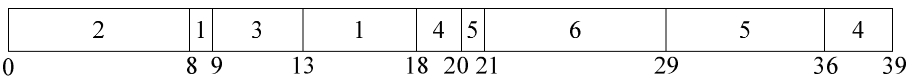
In order to devise a heuristic estimation, we have used a problem relaxation. The residual problem represented by a state n is given by the unscheduled operations in n together with their heads and tails, i.e. the triplet $P(n) = (US(n), \mathbf{r}_n(US(n)), \mathbf{q}_n(US(n)))$. Problem relaxation is made in two steps. Firstly, for each machine m with a requiring operation in $US(n)$, the simplified problem $P(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m),\ \mathbf{q}_n(US(n)|_m))$ is considered, where $US(n)|_m$ denotes the unscheduled operations in n requiring machine m. Problem $P(n)|_m$ 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 (1989, 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 (1989, 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 $P(n)|_m$, denoted $JPS(P(n)|_m)$, provides a lower bound of the completion time of problem $P(n)$, denoted $C_{max}(P(n))$. As $f^*(n) = max\{g(n), C_{max}(P(n))\}$, the value $JPS(P(n)|_m)$ is a lower bound of $f^*(n)$ too. So, to obtain

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

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) a lower bound of $h^*(n)$, the value of the largest completion time of operations in $SC(n)$, i.e. $g(n)$, should be considered and the heuristic, termed h_{JPS}, is calculated as

$$(2)\; h_{JPS}(n) = max\{0, JPS(J(n)) - g(n)\}; \quad JPS(J(n)) = \max_{m \in R}\{JPS(J(n)|_m)\}$$

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 operations 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 (1999) a method is proposed for the Project Scheduling Problem and in Korf (2003) and Korf (2004) similar methods are proposed for the Bin Packing Problem and the two-dimensional Cutting Stock Problem respectively.

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

Definition 6.1. *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. 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. The sufficient condition for dominance is formalized in the following two results.

Proposition 6.2. *Let n_1 and n_2 be two states such that $SC(n_2) \subseteq SC(n_1)$ and $\boldsymbol{r}_{n_1}(US(n_1)) \leq \boldsymbol{r}_{n_2}(US(n_1))$, then the following conditions hold:*

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

2. $JPS(P(n_1)) \leq JPS(P(n_2))$.
3. $C_{max}(P(n_1)) \leq C_{max}(P(n_2))$.

Proof. Condition 1 comes from the fact that each operation $v \in US(n_1)$ is an unscheduled operation in both states n_1 and n_2. Consequently, v has not any disjunctive successor yet. So, according to equations (1), $q_v(n_1) = p_{SJ_v} + q_{SJ_v}(n_1)$ 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, $\boldsymbol{q}_{n_1}(US(n_1)) = \boldsymbol{q}_{n_2}(US(n_1))$.

To prove condition 2, let us denote $P(n_2|n_1)$ to the problem comprising operations in $US(n_1)$ but considering the heads of these operations as in state n_2, i.e. $P(n_2|n_1) = (US(n_1),\ \boldsymbol{r}_{n_2}(US(n1)),\ \ \boldsymbol{q}_{n_2}(US(n_1))))$. Problems $P(n_2|n_1)$ and $P(n_1)$ have the same operations, and the head of each operation in $P(n_1)$ is lower or at least equal than it is in $P(n_2|n_1)$, while the tails are equal. Therefore, any preemptive schedule for operations $US(n_1)|_m$ with heads and tails as they are in problem $P(n_1)$ is also a feasible preemptive schedule for these operations with heads and tails as they are in $P(n_2|n_1)$. So, it is clear that $JPS(P(n_1)) \leq JPS(P(n_2|n_1))$. Through analogous reasoning $JPS(P(n_2|n_1)) \leq JPS(P(n_2))$, as the operations in $P(n_2|n_1)$ are a subset of those in $P(n_2)$ and the heads and tails of the operations in common are the same.

Finally, condition 3 can be proved through similar reasoning as condition 2, as $C_{max}(P(n_1)) \leq C_{max}(P(n_2|n_1)) \leq C_{max}(P(n_2))$.

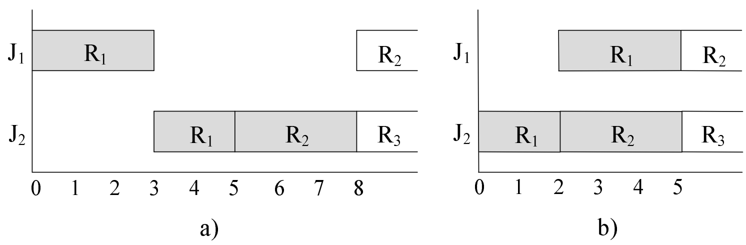
Theorem 6.3. *Let n_1 and n_2 be two states such that $n_2 \notin P^*_{s\text{-}n_1}$. If $SC(n_2) \subseteq SC(n_1)$, $\boldsymbol{r}_{n_1}(US(n_1)) \leq \boldsymbol{r}_{n_2}(US(n_1))$ and $f(n_1) \leq f(n_2)$, then the following conditions hold, where $D(n)$ denotes the depth of node n in the search tree:*

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

Proof. Condition (1) is trivial from $SC(n_2) \subseteq SC(n_1)$, as $D(n) = |SC(n)|$. From condition (1) the only possibility for condition (2) not to hold is that $n_1 = n_2$, but this can not be true due to $n_2 \notin P^*_{s\text{-}n_1}$. So condition (2) holds.

To prove condition (3), let us remember that $f(n) = max\{g(n), JPS(P(n))\}$ and $f^*(n) = max\{g(n), C_{max}(P(n))\}$. As $JPS(P(n_1)) \leq JPS(P(n_2))$, from

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



$f(n_1) \leq f(n_2)$ it follows that either (a) $g(n_1) \leq g(n_2)$ or (b) $g(n_1) > g(n_2)$ and $JPS(P(n_2) \geq g(n_1)$. If (a) holds, as $C_{max}(P(n_1)) \leq C_{max}(P(n_2))$ it follows that $f^*(n_1) \leq f^*(n_2)$. If (b) holds, as $C_{max}(P(n_2)) \geq JPS(P(n_2)) \geq g(n_1) > g(n_2)$, then $f^*(n_1) \leq C_{max}(P(n_2)) \leq f^*(n_2)$. Then 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. To establish that node n_1 dominates node n_2 the following conditions should be verified

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

so in principle each time a new node n_1 appears during the search, this node should be compared with any other node n_2 reached previously. When $f(n_1) = f(n_2)$, 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 we simplify this process and proceed as follows:

(i) Each time a node n is selected by A^* for expansion, n is matched with every node n' in $OPEN$ such that $f(n) = f(n')$. As both nodes are in $OPEN$, only conditions (2) and (3) have to be tested. 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.

(ii) If node n is not pruned in step (i), it is compared with those nodes n' in the $CLOSED$ list such that $D(n') \geq D(n)$. This is a necessary condition for n' dominates n and it also implies condition 1 for n and n'. Moreover, $f(n') \leq f(n)$ due to h_{JPS} being consistent. So, only conditions 2 and 3 have to be checked. If n' dominates n, then n is pruned.

In step (i), 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 (ii), nodes n' with $D(n') \geq D(n)$ may be efficiently searched in $CLOSED$ by ordering this list accordingly with the depth of the expanded states. Regarding step (ii), n might dominate n' if $f(n) = f(n')$; in this case, n' and every descendant of n' at any level of the search, some of which might be in $OPEN$, may be pruned. This requires searching over the whole $CLOSED$ list and keeping trace of successors for each of the expanded nodes. In our experiments, we have not considered this possibility because it doesn't make up for the cost of searching and keeping links from parents to children. The reason for this is that most of the nodes in $OPEN$ that are pruned in this way became also pruned from comparison with other nodes.

7 Experimental study

For experimental study we have chosen the set of 15 problems LA01-15 from the *OR-library* (*http://people.brunel.ac.uk/ mastjjb/jeb/info.html*). These are small and medium size instances, LA01 to LA05 are 10×5 (10 jobs and 5 machines), LA06 to LA10 are 15×5 and LA11 to LA15 are 20×5. Also, for the purpose of comparison with other methods, we have considered the set of instances proposed in Sadeh (1996). This is a set of 60 instances of size 10×5 which is organized in 6 groups of 10 problems each. Each group is characterized by two parameters: BK (number of bottleneck resources) and RG (range parameter). A resource is a bottleneck if it appears at the same position in the machine sequence of all jobs. RG controls the distribution of release dates and deadlines of jobs.

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 $3Ghz$ with $1Gb$ 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 have an idea about the size of the search space for these instances. We have given a time limit of 3600 seconds for each run.

Table 1 summarizes the results of this experiment. As we can observe, when pruning is not applied, instances 10, 11 and 13 remain unsolved due to memory getting exhausted. On the other hand, with pruning applied in its full extension, i.e. by comparing the expanded node n' with all node n in $CLOSED$ with $D(n) \geq D(n')$, instances 10 and 13 are solved but instance 11 is not solved either; in this case the memory is not exhausted in the time limit. The remaining instances are solved in both cases. For all instances, the number of nodes expanded is lower when pruning is applied (here it is important to remark that the memory consumed is in direct ratio with the number of expanded nodes), while the time is similar in average, but it is larger in some cases. So, we have experimented by restricting comparisons to nodes n in $CLOSED$ with $D(n) = D(n')$. In this case, the time is clearly lower, even though the number of expanded nodes augments slightly. In spite of that, instance 11 still remains unsolved due to time limit.

In the second series of experiments, we have enhanced A^* with upper bounds calculation by means of a greedy algorithm. As it was done in Brucker (1994, 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. When $P < 1$, the results are averaged over 20 runs for each instance. Table 2 reports results from a set of experiments with different values of P. Let us firstly consider the first two parts of this table. In the first part, results from no pruning are reported; while in the second one the results came from applying pruning by dominance in its full extension. As we can observe, the number of expanded nodes is always much lower with pruning than it is without it; and this number is in inverse ratio with the value of P, as it can be expected. However, the time taken is similar or even larger with pruning. The third part of Table 2 shows results from restricting comparison to nodes with the same depth in the search tree. In this case, the search takes lower time than it takes in previous experiments, while the number of expanded nodes is only a little bit larger. So, this seems to be the best choice. As we can observe from the average values reported in Table 2, pruning in its full extension reduces the number of expanded nodes in about 82% with respect to the non-pruning version, while the time increases in about 40%. However, pruning restricted to nodes of the same depth reduces the expanded nodes in about 80% and also the time in more than 50%. Overall, we can conclude that the proposed method that combines pruning by dominance with probabilistic calculation of upper bounds is efficient when searching in the space of active schedules for makespan minimization.

In the last series of experiments we consider the benchmark proposed in Sadeh (1996). Table 3 summarizes the results obtained across these instances. For each group of 10 instances with the same values of parameters RG and BK, this table reports the number of generated and expanded nodes as well as the time taken. These values are averaged first over the 20 runs with each instance and then over the 10 instances of each group. All instances got solved in all runs. As we can observe, the time taken and the number of generated and expanded nodes are much lower with pruning by dominance. Being the differences more significative for the hardest instances, i.e. those in the second, fourth and sixth groups. In principle, these results are not directly comparable with those reported in Sadeh (1996). The reason for this is that Sadeh and Fox have considered a decision version of the problem with due dates. In average, these due dates are at least a 20% larger

Table 1 Summary of results of pruning by dominance over instances LA01-15.

Results obtained without considering UBs during the search, i.e. $UB = \infty$						
	No pruning		Pruning by dominance $D(n) \geq D(n')$		Pruning by dominance $D(n) = D(n')$	
Inst.	Expanded	Time(s)	Expanded	Time(s)	Expanded	Time(s)
1	418	0	158	0	165	0
2	57103	27	9454	78	10509	7
3	249	1	216	0	217	0
4	63969	30	7309	33	7888	6
5	8397	4	3518	10	3731	3
6	14270	9	1935	5	2220	3
7	1853	1	1158	2	1243	2
8	2926	3	1494	2	1517	2
9	678	0	436	1	439	0
10	**280582**	**182**	37894	1142	52713	71
11	**131470**	**143**	**72067**	**3600**	**105449**	**272**
12	1689	1	952	3	965	2
13	**111891**	**141**	13111	89	13599	33
14	258	0	257	0	257	0
15	76967	93	20022	275	22068	46

bold *indicates time limit (3600 s.) or memory limit getting exhausted.*

Table 2 Summary of results combining pruning by dominance with probabilistic calculation of heuristic solutions during the search over instances *LA*01-15. When $P < 1$, the results are averaged over 20 runs for each instance. The heuristic algorithm is run from the initial state and then for each expanded state with probability P. Instances not included in this table get solved at the initial state, i.e. $f(start) = First\ UB = C^*$

	$P = 1$		$P = 0,1$		$P = 0,01$	
Inst.	Expanded	Time(s)	Expanded	Time(s)	Expanded	Time(s)
No pruning						
1	23	0	400	0	418	0
2	57082	151	57091	38	57102	27
3	152	1	189	0	248	0
4	63892	158	63892	42	63892	30
7	26	1	1384	2	1852	2
8	2904	10	2911	3	2924	2
12	18	0	25	0	696	1
13	6	1	15	0	96968	126
15	76863	457	76918	127	76963	95
Average						
	22330	87	22536	24	33451	31
Pruning by dominance						
1	23	0	89	0	158	0
2	9433	105	9438	80	9453	78
3	128	1	170	0	209	0
4	7152	52	7152	34	7214	32
7	26	1	752	2	1098	2
8	1472	8	1479	3	1491	2
12	18	1	28	0	545	2
13	6	1	18	0	7226	50
15	19950	397	20002	287	20021	276
Average						
	4245	63	4348	45	5268	49
Pruning by dominance, restricting matching to states of the same depth						
1	23	0	145	0	165	0
2	10488	36	10497	10	10505	7
3	128	1	180	0	215	0
4	7714	28	7714	8	7801	6
7	26	1	1049	2	1118	2
8	1495	7	1503	2	1516	2
12	18	0	26	0	127	1
13	6	1	14	0	10206	25
15	4655	178	22042	58	22066	46
Average						
	4657	28	4797	19	5969	10

than the optimal makespan. In their experimental study, they reach solutions for 52 instances in a time of about 3 or 4 seconds on a DECstation 5000/200, while the remaining 8 instances remain unsolved even taking a much larger time. As they report solutions fulfilling the due date constrains and these due dates are considerably larger than the optimal makespan, it is expected that their solutions are far from being optimal. So, from all these considerations, we can consider our approach more efficient than that reported in Sadeh (1996).

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 obtain upper bounds during the search. We have reported results from an experimental study over instances taken from the *OR-library* and from Sadeh (1996). These experiments show that the proposed method of pruning by dominance, in combination with the greedy algortithm , is efficient as it allows to save both space and time. Furthermore, the method is much more efficient than the backtracking algorithm proposed in Sadeh (1996).

As future work, we plan to combine the pruning strategy with constraint propagation techniques, such as those proposed in Dorndorf (2000, 2002), as it is done in the branch and bound algorithm described in Brucker (1994, 2004). Also, we plan to apply the pruning by dominance method to other scheduling problems which are harder to solve than the JSSP with makespan minimization such as the JSSP with total flow time or tardiness minimization; and the the JSSP with setup times. Also, we will confront other problems such as the Travelling Salesman Problem or the Cutting-Stock Problem. 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 both cases.

Table 3 Summary of results with Heuristic h_{JPS} with pruning by dominance over the *Sadeh* instances. Time limit is 3600s.

Results obtained considering UBs during the search with $P = 0.01$						
	No pruning			Pruning by dominance		
Subset Inst.	Generated	Expanded	Time(s)	Generated	Expanded	Time(s)
$BK = 1, RG = 0,0$	336	148	0	269	118	0
$BK = 2, RG = 0,0$	79936	32118	19	9568	3899	5
$BK = 1, RG = 0,1$	158	74	0	150	71	0
$BK = 2, RG = 0,1$	116911	45260	26	8441	3423	5
$BK = 1, RG = 0,2$	140	73	0	113	58	0
$BK = 2, RG = 0,2$	1543	714	0	625	276	0

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Education under research project TIN2007-67466-C02-01. The authors thank the anonymous reviewers and the participants in conference EPIA 2007 for their comments and suggestion on earlier versions of this paper.

References

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

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

Brucker, P., Knust, S. (2006) *Complex Scheduling*, Springer.

Carlier, J., Pinson, E. (1989) 'An algorithm for solving the job-shop problem', *Management Science*, Vol. 35, No. 2, pp.164–176.

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

Dorndorf, U., Pesch, E., Phan-Huy, T.(2000) 'Constraint propagation techniques for the disjunctive scheduling problem', *Artificial Intelligence*, Vol. 122, pp.189–240.

Dorndorf, U., Pesch, E., Phan-Huy, T.(2002) 'Constraint propagation and problem descomposition: A preprocessing procedure for the job shop problem', *Annals of Operations Research*, Vol. 115, pp.125–142.

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

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*, Vol. 4, No. 2, pp.100–107.

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

Korf, R. (2004) 'Optimal Rectangle Packing: New Results', *In Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS04)*, pp.132–141.

Hart, P., Nilsson, N., Raphael, B. (1999) 'The multiple resource constrained project scheduling problem: A breadth-first approach', *European Journal of Operational Research*, Vol. 112, pp.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.

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

Sierra, M., Varela, R. (2005) 'Optimal scheduling with heuristic best first search', *AI*IA 2005. Advances in Artificial Intelligence*, Springer-Verlag, Vol. 3673, pp.173–176.

Varela, R., Soto, E. (2002) 'Scheduling as heuristic search with state space reduction', Springer-Verlag, Vol. 2527, pp.815–824.