AI Communications 00 (20xx) 1–17
DOI 10.3233/AIC-140630
IOS Press

Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules

Carlos Mencía [a], María R. Sierra [a], Miguel A. Salido [b], Joan Escamilla [b] and Ramiro Varela [a,*]
[a] *Departamento de Informática, Universidad de Oviedo, Gijón, Spain*
E-mails: {menciacarlos, sierramaria, ramiro}@uniovi.es
[b] *Instituto Universitario de Automática e Informática Industrial, Universidad Politécnica de Valencia, Valencia, Spain*
E-mails: {msalido, jescamilla}@dsic.upv.es

Abstract. The job shop scheduling problem with an additional resource type has been recently proposed to model the situation where each operation in a job shop has to be assisted by one of a limited set of human operators. We confront this problem with the objective of minimizing the total flow time, which makes the problem more interesting from a practical point of view and harder to solve than the version with makespan minimization. To solve this problem we propose an enhanced dept-first search algorithm. This algorithm exploits a schedule generation schema termed $OG\&T$, two admissible heuristics and some powerful pruning rules. In order to diversify the search, we also consider a variant of this algorithm with restarts. We have conducted an experimental study across several benchmarks. The results of this study show that the global pruning rules are really effective and that the proposed algorithms are quite competent for solving this problem.

Keywords: Depth first search, pruning by dominance, job shop scheduling problem, operators, restarts

1. Introduction

We face a variant of the job-shop scheduling problem in which the processing of an operation on a given machine requires the assistance of one of a limited number of available operators. This problem has been recently proposed in [2] with the objective of minimizing the makespan; it is termed *JSO*(n, p), where n represents the number of jobs and p represents the number of available operators. In this paper, we consider minimizing the total flow time. This objective function is often of more interest than the makespan in real environments [9] and at the same time it may make scheduling problems harder to solve [23].

Depth-first search is commonly used in combinatorial optimization [38] due to some properties as its anytime behavior or its low requirement of memory. To solve the *JSO*(n, p) problem, we propose a partially informed depth-first search algorithm [39] enhanced with global pruning rules. The definition of the search space and one of the heuristic estimations are borrowed from [49] where a best-first search algorithm is proposed for the same problem. Besides, a key component of our approach is the combination of depth-first search with a global pruning rule. This rule is defined and formalized in accordance with the formal definition given in [25] for dominance relations that guarantees a single optimal solution. To implement this pruning method, the expanded states have to be maintained in memory in order to be compared with each previously expanded state. This is space consuming and so it requires to establish a limit to prevent the algorithm from running out of memory. In this work, we use a single static method that limits the size of the memory dedicated to store expanded nodes. When the limit is reached, no more states are stored. In spite of this simple memory model, the pruning method is really efficient. As we will see, it allows the depth-first search algorithm to reduce the number of expansions in more than one order of magnitude meanwhile it is able to reach and certify

*Corresponding author: Ramiro Varela, Departamento de Informática, Universidad de Oviedo, Campus de Viesques s/n, 33271, Gijón, Spain. E-mail: ramiro@uniovi.es

an optimal solution. For large instances that cannot be solved to optimality, the global pruning rule allows the algorithm to obtain better solutions by a given time, even though it can expand less nodes by this time due to the overhead of dominance checking. Nevertheless, checking the dominance relations is not a time consuming task thanks to the use of hashing mechanism.

We also consider a variant of depth-first which restarts the search after a number of fails in accordance with a restart sequence. This is a technique of common use which helps the algorithms to diversify the search and so it is expected to be effective mainly in large search spaces.

Over the last years, global pruning rules that are based on dominance relations were widely used in constraint-based reasoning for breaking symmetries in some classes of problems [20]. In this context, rules more powerful than those intended for detecting symmetries were also proposed, as for example in [42] where three symmetric problems such as the Maximum-Density Still Life problem, the Steel Mill Slab Design and the Peaceable Armies of Queens were considered. The *JSO*(n, p) problem presents some symmetries as well; for example two states with the same subset of operations scheduled at the same times but with a different assignment of operators are actually symmetric. As we will see this kind of symmetries are detected by the proposed rule as well. Global pruning rules were also used in some scheduling problems, for example in [36] a dominance rule for the multiple resource constrained project scheduling problem is defined and then exploited in combination with a breadth-first search algorithm.

As far as we know, the best-first search algorithm given in [49] is the only approach proposed for the *JSO*(n, p) problem to minimize the total flow time. This method is very effective for solving small instances but its high memory requirements make it inappropriate for facing large instances. So we have considered an implementation on IBM ILOG CPLEX CP Optimizer to compare with. This is a commercial solver embedding powerful constraint propagation techniques and a self-adapting large neighborhood search method dedicated to scheduling [31] and it is expected to be very efficient for a variety of scheduling problems as it is pointed in [26], in particular when the cumulative demand for resources exceeds their availability as it happens, for example, in the Satellite Control Network Scheduling Problem confronted in [29].

CP Optimizer is often used to compare with other approaches to scheduling problems. For example, in [18] the authors confront a parallel machine scheduling problem with precedence constraints and setup times by means of a branch and bound procedure combined with a climbing discrepancy search algorithm. The results of this algorithm are compared with those from CP Optimizer and in some cases this solver achieves the best results. Also, in [37] the authors propose an iterative flattening algorithm for a job shop problem with blocking constraints. This method is able to improve the best known solutions to a lot of instances and then it is compared with CP Optimizer. Their results show that CP Optimizer reaches better solutions than the proposed algorithm in a number of instances.

We have conducted a thorough experimental study across conventional instances with different sizes and considering in each case different number of available operators. The purpose of this study is to assess our algorithms and to compare them with CP Optimizer as well. The results of these experiments show clearly that the proposed global pruning rules are really effective as they improve the performance of the algorithms so as they outperform a powerful solver such as CP Optimizer.

The remaining of the paper is organized as follows. In Section 2 we define the problem. Then, in Section 3, we describe the schedule generation scheme termed $OG\&T$ proposed in [49] which is used to define the search space for the depth-first search algorithm. Section 4 describes this search space and the heuristic estimation used to guide the search algorithm. In Section 5, we formalize the global pruning rules for the *JSO*(n, p) with total flow time minimization and demonstrate how these rules can be efficiently applied in combination with depth-first search. In Section 6 the diversification mechanism based on randomization and restarts is described. In Section 7, we report the experimental study. Finally, in Section 8, we summarize the conclusions and remark the original contributions of the paper; we also give some ideas for further research.

2. Problem formulation

Formally the job-shop scheduling problem with operators can be defined as follows. We are given a set of n jobs $\{J_1, \ldots, J_n\}$, a set of m resources or machines $\{R_1, \ldots, R_m\}$ and a set of p operators $\{O_1, \ldots, O_p\}$. Each job J_i consists of a sequence of v_i operations or tasks $(\theta_{i1}, \ldots, \theta_{iv_i})$. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, an integer duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ and an assisting operator $O_{\theta_{il}}$ to be deter-

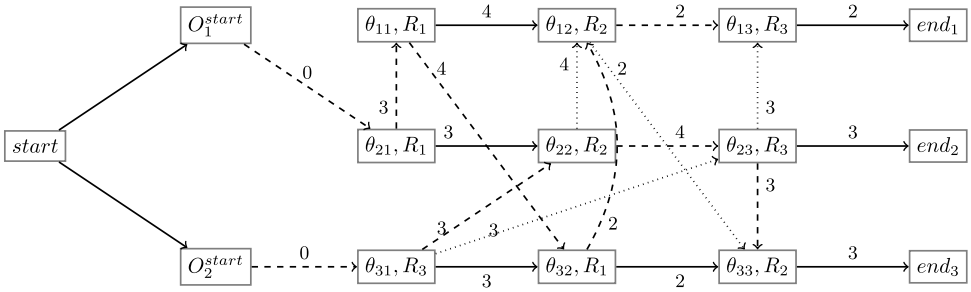


Fig. 1. A feasible schedule for a problem instance with 3 jobs, 3 machines and 2 operators.

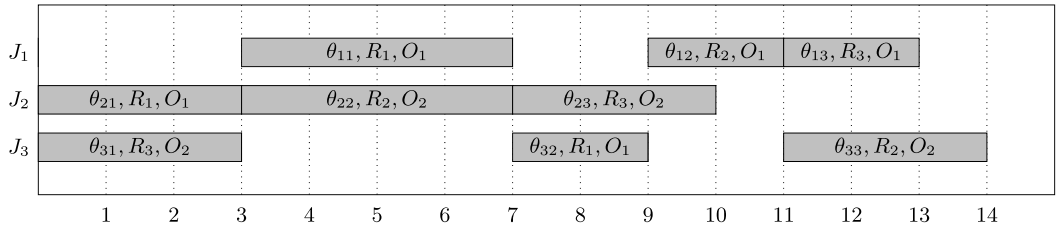


Fig. 2. Gantt chart of the schedule represented in Fig. 1.

mined. A feasible schedule is a complete assignment of starting times and operators to operations that satisfies the following constraints: (i) the operations of each job are sequentially scheduled, (ii) each machine can process at most one operation at any time, (iii) no preemption is allowed and (iv) each operation is assisted by one operator and one operator cannot assist more than one operation at the same time. The objective is finding a feasible schedule that minimizes the sum of the completion times of all jobs, i.e. the total flow time. This problem was first defined in [2] for makespan minimization and is denoted as *JSO*(n, p).[1]

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

Scheduling problems are usually represented by means of a disjunctive model [45]. We propose here to use a model for the *JSO*(n, p) that is similar to that used in [2]. Figure 1 shows a solution graph for an instance with 3 jobs, 3 machines and 2 operators. In addition to the nodes that represent operations and the dummy nodes *start* and end_i, $1 \leqslant i \leqslant n$, we introduce nodes to represent operators in this model, denoted O_j^{start}, $1 \leqslant j \leqslant p$. So, we have three main types of arcs: job, machine and operator arcs. Each type define the sequence of operations in the same job, machine or operator respectively. Operator nodes are linked to the first operation assisted by the operator. Also, there are arcs from the *start* node to each operator node and from the last operation of each job to the corresponding end_i node.

In Fig. 1, discontinuous arrows represent operator arcs. So, the sequences of operations assisted by operators O_1 and O_2 are $(\theta_{21}, \theta_{11}, \theta_{32}, \theta_{12}, \theta_{13})$ and $(\theta_{31}, \theta_{22}, \theta_{23}, \theta_{33})$ respectively. In order to simplify the picture, if there are two arcs between the same pair of nodes, only the operator arc is drawn. Continuous arrows represent job arcs and dotted arrows represent machine arcs; in these cases only arcs not overlapping with operator arcs are drawn. In this example, the completion times of jobs J_1, J_2 and J_3 are 13, 10 and 14 respectively, so the schedule has a total flow time of 37. Figure 2 shows a Gantt chart for the schedule represented by Fig. 1. As we can see, all the constraints are satisfied and, as there are only two operators available, no more than two operations are ever processed in parallel.

In the same way as the classic job-shop scheduling problem, the *JSO*(n, p) is a particular case of other problems such as the multi-resource shop scheduling with resource flexibility defined in [12] or the more general resource-constrained project scheduling problem (RCPSP).

[1] Note that from the point of view of constraint programming, this problem may be naturally formulated as a classical job shop scheduling problem with an additional cumulative resource of capacity p.

3. Schedule generation schemes

A schedule generation scheme is a strategy to generate and enumerate a set of schedules. We use here the $OG\&T$ algorithm proposed in [50] for the *JSO*(n, p). However, any of the schedule generations schemes proposed for the RCPSP could be in principle applied to the *JSO*(n, p) as well. Among others, we could consider the serial and parallel methods proposed in [27], or the precedence tree, delay alternatives and extension alternatives surveyed in [8]. Some of these methods are summarized in [3] (Chapter 5) and [15] (Chapter 6). In the sequel, we first describe the $OG\&T$ algorithm and then clarify its advantages with respect to the above alternatives particularized to the *JSO*(n, p) problem.

3.1. The OG&T algorithm

This scheme is an extension of the well-known $G\&T$ algorithm proposed by Giffler and Thompson in [21] for the classical job-shop scheduling problem. The $OG\&T$ algorithm is thoroughly explained and formalized in [50]; in this section we present a brief description of how it works, so that it could be implemented. In this scheme, the operations are scheduled one at a time following a sequence of non-deterministic choices. When an operation u is scheduled, its preceding operation in the job sequence, denoted PJ_u, was already scheduled if this operation exists. At this time, u is assigned a starting time st_u and an operator O_i, $1 \leqslant i \leqslant p$.

Let *SC* be the set of scheduled operations at an arbitrary time. Then, the next operation to be scheduled may be, in principle, any operation of the set A defined as

$$A = \left\{v \notin SC, \nexists PJ_v \vee (PJ_v \in SC)\right\} \tag{1}$$

i.e., the set that includes the first unscheduled operation of each job that has at least one unscheduled operation. If the operation u in A is selected, the starting time of u is given by its head r_u which is calculated as

$$r_u = \max\left\{r_{PJ_u} + p_{PJ_u}, r_v + p_v, \min_{1 \leqslant i \leqslant p} t_i\right\}, \tag{2}$$

where t_i, $1 \leqslant i \leqslant p$, is the time at which the operator O_i becomes available and v denotes the last operation scheduled having $R_v = R_u$. At the same time, the operator O_i that is available at the latest time before r_u, i.e.

$$i = \arg\max\{t_j; t_j \leqslant r_u; 1 \leqslant j \leqslant p\} \tag{3}$$

is assigned to assist the operation u. Let v^* be the operation in A having the earliest completion time if it were scheduled next, i.e.

$$v^* = \arg\min\{r_u + p_u; u \in A\}. \tag{4}$$

The set of non-deterministic choices may be reduced to the subset $A' \subset A$

$$A' = \{u \in A; r_u < r_{v^*} + p_{v^*}\}. \tag{5}$$

Moreover, the set of choices can be further restricted in the following way. Let $\tau_0 < \cdots < \tau_k$ be the sequence of all times along the interval $[\min\{r_u; u \in A'\}, r_{v^*} + p_{v^*})$, where each τ_i is given by the head of some operation in A' or the time at which some operator becomes available. Let p'_i be the number of operators available in the subinterval $[\tau_i, \tau_{i+1})$ and let m'_i be the number of different machines that are required by the operations in A' which may be processed along this subinterval. Then, A' may be reduced as long as the following operations are maintained:

(i) The operations requiring the machine R_{v^*}.
(ii) For each interval $[\tau_i, \tau_{i+1})$ with $m'_i > p'_i$, the operations requiring $m'_i - p'_i$ machines other than R_{v^*}.

The set of operations obtained in this way is termed B and it is clear that $|B| \leqslant |A'| \leqslant |A|$. An important property of this schedule generation scheme is that if the number of operators is large enough, in particular if $p \geqslant \min(n, m)$ so as *JSO*(n, p) becomes $J||\Sigma C_i$, it is equivalent to the $G\&T$ algorithm. In [50] a full description of $OG\&T$ is given together with a formal proof of its dominance property.

We can illustrate how $OG\&T$ algorithm works by means of an example. Let us consider the state in Fig. 3. Here we have a partial schedule for an instance with 5 jobs, 5 machines and 3 operators. The first operations of the 4 first jobs are already scheduled, so $A = \{\theta_{12}, \theta_{22}, \theta_{32}, \theta_{42}, \theta_{51}\}$. The critical operation is $v^* = \theta_{51}$, therefore the operation θ_{12} is discarded as $8 = r_{\theta_{12}} \geqslant r_{\theta_{51}} + p_{\theta_{51}} = 8$ and $A' = \{\theta_{22}, \theta_{32}, \theta_{42}, \theta_{51}\}$.

In order to obtain the set B, the interval $[T, C)$ is divided into two subintervals $[\tau_0, \tau_1) = [4, 5)$ and

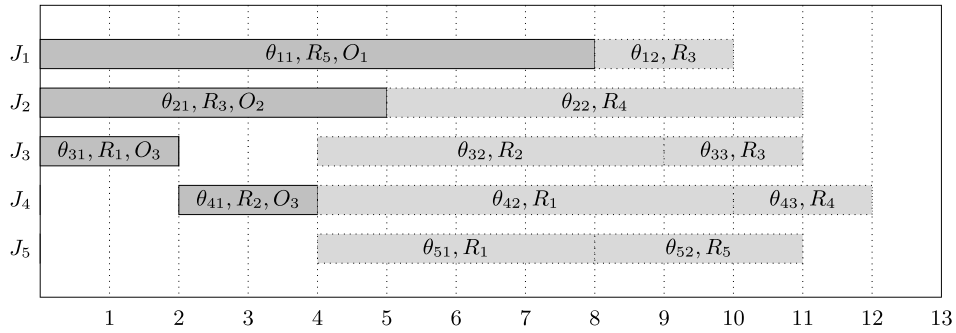


Fig. 3. Partial schedule for a problem instance with 5 jobs, 5 machines and 3 operators.

$[\tau_1, \tau_2) = [5, 8)$; $\tau_1 = 5$ is given by both the completion time of operation θ_{21} and the time at which the operator O_2 gets available. In this case, R_{v^*} is R_1, so the operations θ_{42} and θ_{51} must be included in B. Then, from the first interval $[\tau_0, \tau_1) = [4, 5)$, as $m'_0 = 2$ and $p'_0 = 1$, one more machine have to be considered. The only option is R_2, so the operation θ_{32} must be included in B as well. Finally, from the second interval $[\tau_1, \tau_2) = [5, 8)$, with $m'_2 = 3$ and $p'_2 = 2$, the operations requiring R_2 or R_4 must be included. As R_2 was already included from a previous interval, no new operations are included in this step. So, finally $B = \{\theta_{22}, \theta_{32}, \theta_{42}\}$.

3.2. Other schedule generation schemes

As we have seen, the $OG\&T$ algorithm relies on partial schedules which are built by scheduling operations one at a time. So, it may be compared with other schemes proposed to RCPSP which rely on building partial schedules as well. In general, these schemes are considered for makespan minimization, but they may be adapted for other regular objectives such as the total flow time. It is important to remark that, in general, these schemes are described in combination with some search strategy, such as a dispatching rule. However, we consider them here just as branching schemes that define search trees, which are independent from the search strategy used to explore them.

The serial scheme proposed in [27] and the precedence tree described in [8] are quite similar, both of them build up a partial schedule considering an eligible set of operations (those whose predecessors in the partial order are already scheduled) which, if particularized to the *JSO*(n, p) problem, would be the same as the set A calculated by the $OG\&T$ algorithm in expression (1). So, they would generate a larger number of successors than $OG\&T$ for the majority of the expanded states.

The delay alternatives method was proposed in [13] and considers all the operations already scheduled but not finished at some time t_g at the level g of the search, defined by the earliest completion time of an operation in process at level $g - 1$, together with the unscheduled operations such that all their predecessors have been scheduled with a completion time not greater than t_g. Then, it calculates maximal subsets of these operations that can be processed in parallel. Each one of these maximal subsets defines a successor of the current state. Hence, it may require delaying an operation previously scheduled. So, as it is quite different from the $OG\&T$ algorithm a comparison is not trivial. Nevertheless, if there are many resource conflicts in a branching step of the delay alternatives method, a lot of maximal subsets of operations to be scheduled in parallel would have to be considered. Hence, the branching factor may be larger than that of the $OG\&T$ if the method is applied to the *JSO*(n, p) problem. The extension alternatives method was proposed in [53] and it is similar to delay alternatives. The main difference is that the extension alternatives does not consider delaying operations that were previously scheduled. The parallel scheme described in [27] is similar to the extension alternatives, the main difference being that in the parallel method the operations would be scheduled one at a time if it were particularized to the *JSO*(n, p).

So, the advantage of $OG\&T$ is that it may be able to generate a search tree with lower size than that of the search spaces generated by the above methods proposed for the RCPSP if they were adapted to the *JSO*(n, p). This is quite natural as $OG\&T$ is a method designed specifically to cope with the *JSO*(n, p) problem, in the same way as the classic $G\&T$ is an algorithm specialized in the classic job-shop scheduling problem and so it generates a more appropriate search

space than the particularizations of the schemes from RCPSP.

4. Search algorithm

As we have pointed out, we use here a partially informed depth-first search algorithm [39]. This algorithm starts from an initial state and, in each step, it expands the first one of the set of candidate states stored in the OPEN list. The successors of each expanded state are sorted by non-decreasing f-values and then inserted at the beginning of the OPEN list. f is an admissible heuristic function so as $f(s)$ returns an optimistic estimation of the cost of the best schedule that can be reached from state s, denoted as $f^*(s)$. So, as it is usual, f-values are used to prune states s with $f(s) \geqslant UB$, where UB is the cost of the incumbent solution. In the following subsections we describe the main components of the depth-first search algorithm.

4.1. Search space

The search space is derived from the $OG\&T$ schedule generation scheme for a problem instance $\mathcal{P}$. In the initial state, none of the operations are scheduled yet. In intermediate states, a subset of operations *SC* are already scheduled. To obtain the successors of a state defined by *SC*, a set B is built as it is indicated in Section 3 and then one successor state is generated from each operation $u \in B$ in which u is scheduled at its current head r_u. From the dominance property of $OG\&T$, it follows that the search tree includes at least one optimal solution.

4.2. Heuristic functions

The evaluation function is $f(s) = g(s)+h(s)$, where $g(s)$ denotes the total flow time accumulated in the state s, and $h(s)$ is a heuristic function that estimates the additional cost required to reach a solution from s. We consider two admissible heuristics derived from problem relaxations.

The first one, termed h_{PS}, is borrowed from [51] where the problem $J||\Sigma C_i$ is considered: it relies on relaxing non-preemption and operator constraints, and the capacity constraints for all but one of the machines. This heuristic estimation, denoted $f_{\text{PS}}(s)$, is a lower bound on $f^*(s)$. So, we compute $h_{\text{PS}}(s) = f_{\text{PS}}(s) - g(s)$.

Algorithm 1. Calculating the heuristic h_{OP} for a state s

Require: A state s.
Ensure: The heuristic estimation $h_{\text{OP}}(s)$.
  Build a $(P, NC_{\text{ini}}||\sum C_i)$ instance **P** relaxing the problem represented by the state s as it is indicated in the text;
  for each operation θ in **P** from shortest to largest processing times **do**
    Select the operator O available at the earliest time t;
    Schedule θ at time t assisted by operator O;
  return The total flow time of the built schedule for **P**- $g(s)$;

The second heuristic, termed h_{OP} is original.[2] It is obtained from relaxing constraints other than operators'. To obtain a polynomial relaxation, the capacity constraints of the machines and the heads of the unscheduled operations are relaxed. So, in the relaxed problem the operators play the role of identical parallel machines available at different times and the unscheduled operations of each job are joined into one only operation released at time 0. This problem, denoted $(P, NC_{\text{ini}}||\sum C_i)$, can be optimally solved applying the SPT (Shortest Processing Time) rule [1,46]. Algorithm 1 shows the calculation of heuristic h_{OP} for a state s. It is easy to see that this algorithm runs in a time of order $O(\max(n \times m, n \log n))$.

Finally, we take $h(s) = \max(h_{\text{PS}}(s), h_{\text{OP}}(s))$.

5. Dominance rules

The effective search tree may be reduced by means of dominance relations among states. Given two search states s_1 and s_2, s_1 dominates s_2 iff $f^*(s_1) \leqslant f^*(s_2)$. so if s_1 and s_2 are in memory at the same time, the s_2 can be discarded. In general, dominance relations cannot be easily established, but in some particular cases an effective condition for dominance can be defined. Dominance rules has been defined for many scheduling problems, in most of the cases considering makespan minimization. It is worth mentioning some rules defined for a number of variants of the RCPSP, as the dominance rule proposed in [36] and the cutset rule

[2] In [49], a heuristic termed h_{OP} is proposed, yet it comes from a different problem relaxation.

proposed in [14] and [52]. In the sequel, we propose a dominance rule specific for the *JSO*(n, p) and then compare this rule with those proposed for the RCPSP.

5.1. Dominance rule for the JSO(n, p) problem with total flow time minimization

We propose a dominance rule[3] which is an extension of that defined in [51] for the classic job-shop scheduling problem and the search space induced by the original $G\&T$ algorithm. Here, for the *JSO*(n, p) problem and the search space derived from the $OG\&T$ algorithm, we define the following dominance rule. Let s_1 and s_2 be two search states. We define the relation *DOM* as

Definition 1 (*DOM* relation). s_1 *DOM* s_2 iff

(1) $SC(s_1) = SC(s_2) = SC$.
(2) $r_v(s_1) \leqslant r_v(s_2)$, for all $v \notin SC$.
(3) $\sum_{\theta_{iv_i} \in SC} r_{\theta_{iv_i}}(s_1) \leqslant \sum_{\theta_{iv_i} \in SC} r_{\theta_{iv_i}}(s_2)$.
(4) $av(s_1) \geqslant av(s_2)$.

Where $r_v(s)$, $SC(s)$ and $av(s)$ denote the head of v, the operations scheduled and the availability of operators in state s, respectively. It must be taken into account that θ_{iv_i} is the last operation of job J_i.

Condition (1) restricts the relation to states with the same operations scheduled. From conditions (2) and (4) it follows that the subproblem represented by the unscheduled operations *SC* is less costly for state s_1 than it is for s_2 and condition (3) means that the accumulated flow time due to the jobs with all their operations scheduled is not greater in s_1 than it is in s_2.

The availability of operators in a state can be evaluated as follows. Let $t_1 \leqslant \cdots \leqslant t_p$ be the times at which the operators get idle in the state s (here it is worth noting that the operator available at time t_i is any O_j, $1 \leqslant j \leqslant p$). If u^* is the unscheduled operation with the lowest head in s, then none of the operators can get busy again before r_{u^*}, so we can consider that the operators are actually available for the unscheduled operations at times $t'_1 \leqslant \cdots \leqslant t'_p$, where $t'_i = \max(r_{u^*}, t_i)$. So, the availability of operators in state s is defined as the ordered vector $av(s) = (t'_1, \ldots, t'_p)$. On the other hand, if x is the number of jobs and y is the number of machines with unscheduled operations in *SC*, then the maximum number of operators required to schedule the remaining operations in these states is limited by $p' = \min(p, x, y)$, so $av(s_1) \geqslant av(s_2)$ iff $t'_{1i} \leqslant t'_{2i}$, $1 \leqslant i \leqslant p'$.

The implementation of the dominance rules can be done as follows. When a state s is considered for expansion, s is firstly compared to all the expanded states having the same operations scheduled (condition (1)). This can be done efficiently if the expanded states are stored in a CLOSED list implemented as a hash table where the key values are bit-vectors representing the scheduled operations. Moreover, this rule may be improved from the following result that establishes a sufficient condition for the conditions (1), (2), (3) and (4) not to hold simultaneously.

Proposition 1. *If the heuristic estimation is obtained from a problem relaxation, i.e., $f(s)$ is the cost of an optimal solution to the relaxed problem obtained from s in accordance with that problem relaxation, and the states s_1 and s_2 fulfill all conditions* (1), (2), (3) *and* (4) *then $f(s_1) \leqslant f(s_2)$.*

Proof. It is trivial, as any solution to the relaxed instance obtained from s_2 is a solution to the relaxed instance obtained from s_1. □

So, when a state s is expanded, it has only to be compared for the relation *DOM* with states s' in CLOSED having $f(s') \leqslant f(s)$, as $f(s') > f(s)$ implies that at least one of the four conditions (1), (2), (3) and (4) does not hold.

As both heuristics h_{PS} and h_{OP} are obtained from problem relaxations, the evaluation functions defined as $f_{\text{PS}}(s) = g(s) + h_{\text{PS}}(s)$ and $f_{\text{OP}}(s) = g(s) + h_{\text{OP}}(s)$ fulfill the condition of Proposition 1. Moreover, if we consider $f_{\max}(s) = \max(f_{\text{PS}}(s), f_{\text{OP}}(s))$, the relation *DOM* has not to be checked for the state s' if $f_{\max}(s') > f_{\max}(s)$, due to the following result.

Proposition 2. *If $f_{\max}(s_1) > f_{\max}(s_2)$, then at least one of the conditions $f_{\text{PS}}(s_1) > f_{\text{PS}}(s_2)$ or $f_{\text{OP}}(s_1) > f_{\text{OP}}(s_2)$ holds.*

Proof. Let us suppose that $f_{\max}(s_1) > f_{\max}(s_2)$ and that $f_{\text{PS}}(s_1) \leqslant f_{\text{PS}}(s_2) \wedge f_{\text{OP}}(s_1) \leqslant f_{\text{OP}}(s_2)$. If $f_{\max}(s_1) = f_{\text{PS}}(s_1)$, then $f_{\text{PS}}(s_1) = f_{\max}(s_1) > f_{\max}(s_2) = \max(f_{\text{PS}}(s_2), f_{\text{OP}}(s_2)) \geqslant f_{\text{PS}}(s_1)$, which leads to the contradiction $f_{\text{PS}}(s_1) > f_{\text{PS}}(s_1)$. Analogously if $f_{\max}(s_1) = f_{\text{OP}}(s_1)$. □

From Proposition 1, we have the following result.

[3] This rule was discussed in the conference [49] and used in combination with a best-first search algorithm. In this paper we give a formal description and analysis for the first time.

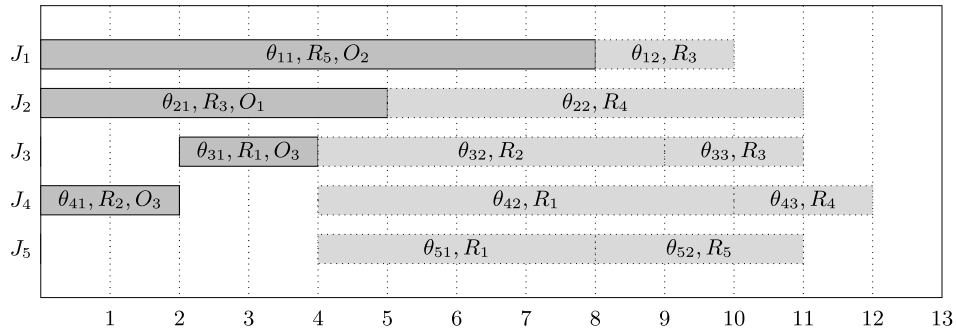


Fig. 4. Another partial schedule for an instance with 5 jobs, 5 machines and 3 operators similar to that in Fig. 3.

Corollary 1. *If s_1 DOM s_2 then s_1 dominates s_2.*

Proof. If we consider the trivial problem relaxation where none of the constraints are relaxed, as $f^*(s)$ is the cost of the optimal solution to the real problem, from Proposition 1, it follows that $f^*(s_1) \leqslant f^*(s_2)$. □

To demonstrate the application of this rule we can consider the search state represented in Fig. 4, which is similar to that of Fig. 3 with the same operations scheduled, but exchanging the order of operations θ_{31} and θ_{41}, which are both assisted by the operator O_3, and the operators assisting operations θ_{11} and θ_{21}. These states dominate each other, so one of them can be discarded. In this example the heads of the unscheduled operations and the operators availability are the same in both states. However, other situations might appear where these values are not the same in two states while one of them dominates the other.

Note that this pruning method generalizes the procedure for checking duplications as in these situations the nodes dominate each other.

As we have pointed out, we defined the rule *DOM* following [25] so that it guarantees a single optimal solution. According to [25], a dominance relation D should satisfy the following properties:

(1) D is a partial order (i.e. a reflexive, antisymmetric and transitive relation),
(2) if $s_1\, D\, s_2$ then s_2 is not a proper descendant of s_1,
(3) if $s_1\, D\, s_2$ and $s_1 \neq s_2$ then $s_1\, D\, s_{2^*}$ for all descendants s_{2^*} of s_1.

So, we have to do the following remark here. Clearly, the relation *DOM* fulfills the above conditions (1) and (2). However, condition (3) does not hold as the relation *DOM* is defined so as for two nodes s and s' not having the same operations scheduled $s\, DOM\, s'$ is evaluated to false. We have defined *DOM* in this way for the sake of efficiency. In [47,48], we have defined a similar condition for the classic JSSP with makespan minimization which was not restricted to states with the same operations scheduled. By means of experimental study we have demonstrated that the number of states pruned along the search is larger and the number of states expanded is lower with the unrestricted condition than they were with the same condition restricted to states with the same operations scheduled. However the number of conditions that have to be checked along the search takes so long time that it does not make up for the lower number of nodes expanded.

5.2. Comparison of DOM with other rules

As we have pointed out, the rules defined for the RCPSP in [14,36] and [52] could be adapted to the *JSO*(n, p). However, we have to be aware that they are defined for makespan minimization while we consider here total flow time instead, and this fact may make the adaptation non trivial. In spite of that, there are some clear differences.

For example, the dominance rule defined in [36] relies only on the scheduled activities, which are divided in two sets: those completed at a "current time" (the *Finished set*) and those which are still in progress (the *Active set*). The dominance rule requires the *Finished set* and the *Active set* to be the same in both states and that the starting times of the operations in one of the active sets to be equal or larger than they are in the other, in order to this state can be pruned. However, the rule *DOM* can prune states in situations where this condition does not hold.

Regarding the cutset rule defined in [13,14] and [52], it also considers two states s_1 and s_2 with the same operations scheduled. With this rule, one candi-

date successor of s_1 can be pruned if the starting time of the operation scheduled to generate this successor state is larger or equal than the accumulated makespan in s_2 and the resource availability in s_1 is not greater than that in s_2. So, it allows to prune neither s_1 nor s_2, but some candidate successor of one of them. In our case, *DOM* would prune all the successors of s_1, independently from the starting time of the operation scheduled to generate each one of them.

6. Enhancing the search algorithm with restarts and randomization

One important drawback of depth-first search (*DF*) is the high price it has to pay when the heuristic makes a wrong decision, as the whole space beyond the selected node has to be explored before considering a different option. Moreover, the heuristics are more likely to fail at shallow levels of the search, which makes things worse. To overcome this inconvenient, a number of techniques such as limited discrepancy search (LDS) [24] or *DF* with restarts and randomized heuristics [22] were devised, being the key point of all of them that of diversifying the search. So, LDS explores firstly the branches of the search tree with less heuristic discrepancies, and *DF* with restarts and randomized heuristics performs a series of randomized depth-first searches limited by a number of steps determined by a restart strategy.

Diversifying the search was quite effective in a variety of constraint optimization problems, as it usually allows the algorithms to come up with high-quality solutions by a reasonable time. For instance, *DF* with restarts is currently the default search strategy used by the IBM ILOG CPLEX CP Optimizer, which is recognized as an outstanding solver, and it is also used by successful search strategies for scheduling problems such as Solution-Guided Multi-Point Constructive Search [7]. However, due to doing restarts, these algorithm may take more time than simple *DF* to prove an optimal solution.

We propose here to enhance the *DF* algorithm described in the two previous sections by means of restarts. Following [54], we have to devise two elements: (1) a method for introducing randomization in the search, and (2) a strategy for deciding when to restart. To do that, we propose:

(1) For introducing randomness in the search, ties in the f-values are broken randomly when the successors of the last expanded state are sorted before being inserted in the OPEN list.

(2) To use a restart sequence,[4] in particular that proposed by Luby et al. [32], which has been proven to be log optimal when the runtime distribution of the algorithm is not known. Following this strategy, in the ith restart the algorithm performs a number of $u \cdot t_i$ search steps,[5] where u is a constant and t_i is given by the following expression:

$$t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^{k-1}, \\ t_{i-2^{k-1}+1} & \\ & \text{if } 2^{k-1} \leqslant i < 2^k - 1. \end{cases} \tag{6}$$

The sequence given in expression (6) is of the form $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots)$ so, as it increases very slowly we scale it by a factor $u = 100$. This way, at the restart i, the *DF* algorithm is run for $100 \times t_i$ steps. The CLOSED list is cleared in each restart.

7. Computational results

The purpose of the experimental study is to assess our proposals, the depth-first search algorithm (*DF*) and its version with restarts and randomization (*DFR*), and to compare them with an implementation on IBM ILOG CPLEX CP Optimizer tool (*CP*).

In the *CP* implementation, *JSO*(n, p) is modeled like a classical job shop scheduling problem where the p operators are naturally modeled as a nonrenewable cumulative resource of capacity p. In the experiments, the solver was set to exploit constraint propagation on no overlap (*NoOverlap*) and cumulative function (*CumulFunction*) constraints to extended level. The search strategy used was depth-first search with restarts (default configuration).

We have experimented across two benchmarks with 710 instances in all. The first one is that proposed in [2] by Agnetis et al., but considering total flow time as objective function instead of makespan. All these instances has $n = 3$ and $p = 2$ and are characterized by the number of machines (m), the number of operations per job ($v_{\max}$) and the range of processing times

[4] A restart sequence is an infinite sequence $S = \langle t_1, t_2, \ldots \rangle$ of positive integers, whose meaning is "run the algorithm for t_1 steps; if this does not yield a proven optimal solution then restart the algorithm and run for t_2 steps, and so on".

[5] For this purpose, we consider a step of the algorithm is completed when finding a deadend in the search or the OPEN list becomes empty.

(p_i). A set of small instances was generated combining three values of each parameter: $m = 3, 5, 7$; $v_{\max} = 5, 7, 10$ and $p_i = [1, 10], [1, 50], [1, 100]$. Also, a set of larger instances was generated with $m = 3$, combining $v_{\max} = 20, 25, 30$ and $p_i = [1, 50], [1, 100], [1, 200]$. In all cases, 10 instances were considered from each combination. The sets of small instances are identified by numbers from 1 to 27: the first set corresponded to the triplet 3–5–10, the second was 3–5–50 and so on. The sets of large instances are identified analogously by labels from $L1$ to $L9$. For all these instances the optimal solution is known and it was obtained by the best-first search algorithm proposed in [49].

The second benchmark includes conventional instances taken from the OR-library [6]: $LA01 - 05$ (10 jobs $\times$ 5 machines), $LA06 - 10$ (15 $\times$ 5), $LA11 - 15$ (20 $\times$ 5), $LA16 - 20$ (10 $\times$ 10), $LA21 - 25$ (15 $\times$ 10), $LA26 - 30$ (20 $\times$ 10), $LA31 - 35$ (30 $\times$ 10) and $LA36 - 40$ (15 $\times$ 15). For each instance, all values in the interval $[1, \min(n, m)]$ are considered as the number of operators p. For many of these instances the optimal solution is still unknown.

In this study, we have given the algorithms a time limit of 300 seconds. Also, *DF* and *DFR* have been given a memory limit of 4 GB to store expanded states when exploiting the global pruning rule. As *DFR* and *CP* are non-deterministic, we ran these algorithms 10 times for each instance. The target machine was Intel Xeon (2.26 GHz), 24 GB RAM. The algorithms are coded in C++.

7.1. Evaluation of the pruning method

The first part of the experimental study is intended to assess the effectiveness of using the dominance relation by *DF* and *DFR* for pruning states. To this aim, we have solved all the instances with up to 100 operations with *DF* and *DFR* in two different modes: without pruning (DF_{NP} and DFR_{NP} resp.) and with pruning (DF_{P} and DFR_{P} resp.).

Table 1 shows the results from the first set of instances averaged for subsets of instances with the same number of operations per job $v_{\max}$. It also includes the results from *CP*, which will be commented in the next section. For each algorithm, we report the time taken in seconds (T), the number instances with proven optimal solutions (#Sol) and the mean relative error in percentage w.r.t. the optimal solution (%Err). For the non-deterministic algorithms we report the mean relative error in percentage of the best and average solutions found and Pearson's variation coefficient in percentage (VC(%)) as a measure of dispersion. Additionally, we report the average number of expanded nodes (#Exp) for *DF* and *DFR*.

As it can be observed in Table 1, the small instances (SMALL) were easily solved by all the algorithms, but both DF_{P} and DFR_{P} took much less time and expanded much fewer states that DF_{NP} and DFR_{NP} respectively; in some cases the difference was of orders of magnitude. At the same time, DF_{P} achieved slightly better results than DFR_{P}. It was in the large instances (LARGE) where there were significant differences among the algorithms. DF_{NP} and DFR_{NP} were able to certify the optimality in 2 and 0 instances by the time limit respectively, while DF_{P} and DFR_{P} certified the optimality in 88 and 50 out of the 90 instances respectively and consequently they also layout much lower error in percentage. In the largest instances, there is a significant difference in favor of the methods with pruning w.r.t. the same methods without it, which makes it clear the utility of the global pruning method in reducing the search space.

For the second set of instances neither of the algorithms can reach the optimal solution in most of the cases. Table 2 summarizes the number of proven optimal solutions and the error in percentage of the solutions reached by the four methods, averaged for instances with the same number of jobs and machines. The errors were computed w.r.t. the lower bounds obtained by the best-first algorithm given in [49] after 300 s. As we can observe, *DF* reached better solutions when exploiting the global dominance rule for all the subsets of instances, and it achieved the best results overall for the smallest instances of size 10×5. At the same time, DFR_{P} outperforms DFR_{NP} in all the subsets of instances and it reached less error in percentage than DF_{P} for the three subsets with the largest instances. It is also remarkable the very low values of Pearson's variation coefficient for both DFR_{P} and DFR_{NP} what indicates that these methods are very stable.

7.2. Comparison with other methods

In the comparison with other methods, the dominance rule was always exploited. As it can be seen in Table 1, both DF_{P} and DFR_{P} were able certify much more solutions and reached much less error in percentage than *CP* in the instances proposed by Agnetis et al. So, the comparison among the algorithms was made in more detail across the 350 LA instances, which are much harder to solve and they make it possible to study the impact of the number of operators available on

Table 1
Summary of results from instances with 3 jobs and 2 operators

		SMALL			LARGE		
		1–9	10–18	19–27	L1–L3	L4–L6	L7–L9
DF_{NP}	T (s)	0.02	0.03	0.76	291.97	300.00	300.00
	#Sol.	90/90	90/90	90/90	2/30	0/30	0/30
	%Err.	0.00	0.00	0.00	2.40	3.67	4.24
	#Exp.	73.58	311.08	4746.47	630,215.27	498,189.73	393,827.57
DF_{P}	T (s)	0.01	0.03	0.14	12.23	57.60	148.97
	#Sol.	90/90	90/90	90/90	30/30	30/30	28/30
	%Err.	0.00	0.00	0.00	0.00	0.00	0.01
	#Exp.	54.13	135.78	469.81	34,635.27	120,805	260,828.43
DFR_{NP}	T (s)	0.01	0.10	4.95	12.23	57.60	148.97
	#Sol.	90/90	90/90	90/90	0/30	0/30	0/30
	%Err. Best	0.00	0.00	0.00	2.48	3.04	3.17
	%Err. Avg.	0.00	0.00	0.00	2.57	3.11	3.25
	VC (%)	0.00	0.00	0.00	0.11	0.07	0.08
	#Exp.	75.51	723.35	35,660.80	1,003,424.79	738,295.47	594,811.89
DFR_{P}	T (s)	0.01	0.03	0.17	59.48	228.19	294.63
	#Sol.	90/90	90/90	90/90	30/30	20/30	2/30
	%Err. Best	0.00	0.00	0.00	0.00	0.32	0.79
	%Err. Avg.	0.00	0.00	0.00	0.00	0.38	1.02
	VC(%)	0.00	0.00	0.00	0.00	0.08	0.21
	#Exp.	55.96	145.18	604.72	161,305.55	471,061.35	545,180.95
CP	T (s)	0.02	0.09	0.74	281.00	300.00	300.00
	#Sol.	90/90	90/90	90/90	4/30	0/30	0/30
	%Err. Best	0.00	0.00	0.00	0.49	1.52	1.60
	%Err. Avg.	0.00	0.00	0.00	0.49	1.56	1.64
	VC(%)	0.00	0.00	0.00	0.00	0.02	0.01

Table 2
Errors in percentage of the solutions reached by the four algorithms averaged for each group of instances with the same size ($n \times m$)

Size	DF_{NP}		DF_{P}		DFR_{NP}				DFR_{P}			
	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err. Best	%Err. Avg.	VC (%)	#Sol.	%Err. Best	%Err. Avg.	VC (%)
10×5	7/25	2.58	15/25	1.24	5/25	2.73	2.83	8.13E−04	10/25	1.40	1.53	1.01E−03
15×5	5/25	6.56	6/25	5.43	5/25	5.48	5.60	8.40E−04	5/25	5.24	5.31	6.47E−04
20×5	5/25	7.20	5/25	6.72	5/25	6.45	6.51	5.83E−04	5/25	6.19	6.27	6.93E−04
10×10	5/50	6.61	10/50	5.55	5/50	5.65	5.72	5.20E−04	7/50	5.02	5.11	7.05E−04

Note: The time limit is 300 s.

the effectiveness of the solving methods. Also, as both *DFR* and *CP* have been shown to be very stable, in this section we only report the average results across the 10 runs of each algorithm.

Table 3 reports the results from the three algorithms averaged for subsets of instances with the same size ($n \times m$). For each subset it shows the number of instances solved to optimality (#Sol.) and the average error in percentage of the solutions reached w.r.t. the best known lower bounds (%Err.). Overall, *DF* and *DFR* achieve better results than *CP* in both the number of instances solved to optimality and the average error reached by the time limit. As we can observe, *CP* is not able to solve to optimality any instance, while *DF* solves 56 and *DFR* solves 47. This is quite reasonable, as in the easiest instances *DF* is able to traverse the

Table 3
Errors in percentage of the solutions reached by the three algorithms averaged for each group of instances with the same size ($n \times m$)

Size	*CP*		*DF*		*DFR*	
	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.
10×5	0/25	2.37	**15/25**	**1.24**	10/25	1.53
15×5	0/25	6.03	**6/25**	5.43	5/25	**5.31**
20×5	0/25	7.73	**5/25**	6.72	**5/25**	**6.27**
10×10	0/50	5.31	**10/50**	5.55	7/50	**5.11**
15×10	0/50	11.18	**5/50**	11.71	**5/50**	**10.28**
20×10	0/50	13.02	**5/50**	12.48	**5/50**	**11.01**
30×10	0/50	17.81	**5/50**	11.58	**5/50**	**10.23**
15×15	0/75	12.73	**5/75**	12.01	**5/75**	**10.74**
Avg. Err.		9.52		8.34		**7.56**
Sum. Sol.	0		**56**		47	

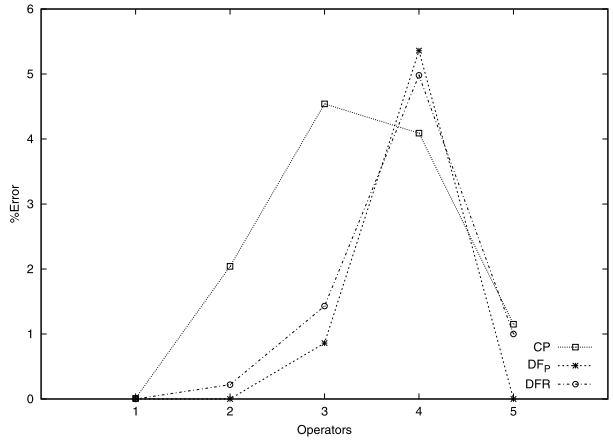
Notes: The time limit is 300 s. Bold values remark the best results.

hole search space by the given time, and *DFR* cannot reach a restart with a number of fails large enough to certify an optimal solution. Nevertheless, *DFR* certifies the same number of optimal solutions as *DF* in 5 out of 8 subsets of instances.

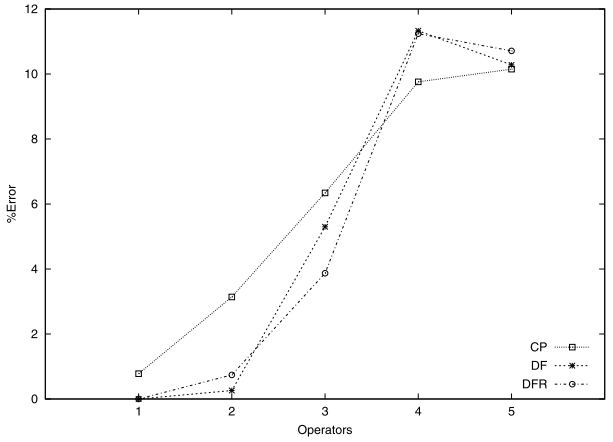
Regarding the quality of the solutions returned, *DFR* achieves the best results, as it is better than *CP* in all the subsets of instances and outperforms *DF* in all but the smallest instances of size 10×5. At the same time, *DF* is better than *CP* in average and is only outperformed in the instances of size 10×10 and 15×10. It is also remarkable that it is in the largest instances with size 30×10 where there is more difference in favor of *DF* and *DFR* w.r.t. *CP*.

Now, we analyze in more detail the performance of the algorithms w.r.t. the number of operators available. Figure 5 contains a graph for each subset of instances with the same size showing the error in percentage of the solutions reached by the three algorithms averaged for instances with the same number of operators available. From these results, we can make the following observations:

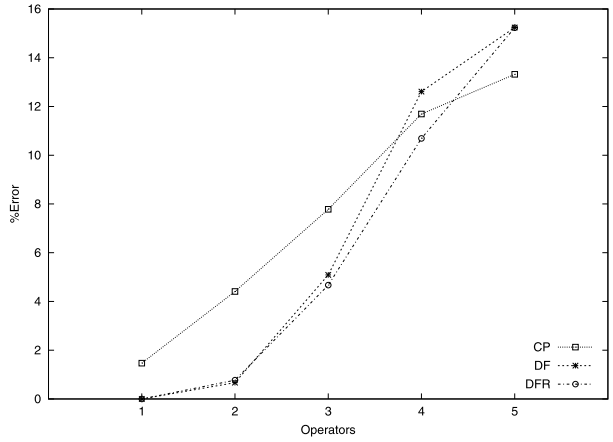
(1) Firstly, for the smallest instances of sizes 10×5 and 15×5 *DF* reaches the best results overall. Here, *CP* only reaches the best results when there are 4 operators available. In the remaining cases *DF* exhibits better results than *DFR* in more configurations, especially for instances of size 10×5, where *DF* achieves the best results in 4 out of 5 subsets of instances.
(2) For instances larger than 15×10, the average error of the solutions computed by *DFR* is less than or at least equal to that from *DF* in almost all the configurations of number of jobs, machines and operators. *DF* is only better than *DFR* for instances of size 10×10 with 8, 9 and 10 operators. Consequently, the improvement in the effectiveness due to performing restarts, allows *DFR* to compute better solutions than *CP* in more cases than *DF*.
(3) There seem to be two trends regarding the difficulty of the instances w.r.t. the number of operators available and how *DF* and *DFR* compare against *CP*. On the one hand, for small or squared instances the algorithms produce an average error that reaches a maximum with a number of operators slightly larger than a half of the maximum possible, and then it decays as there are more operators. In these instances, *CP* computes solutions of higher quality than *DF* and *DFR* in those few intermediate cases, whereas it is outperformed in the rest of the cases, being the difference in favor of *DF* and *DFR* greater for instances with few operators available. On the other hand, rectangular instances larger than 15×5 seem to make the algorithms behave in a different way. In these cases, the average error increases in direct ratio with the number of operators available, and both *DF* and *DFR* reach better solutions than *CP* up to a number of operators, from which *CP* achieves the best solutions. That number of operators increases with the size of the instances: *DF* and *DFR* outperform CP when there are less than 5 and 6 operators respectively for instances of 15×10, when there are less than 7 and 8 operators respectively for 20×10 instances. For the largest instances of size 30×10, *DF* reaches better solutions than



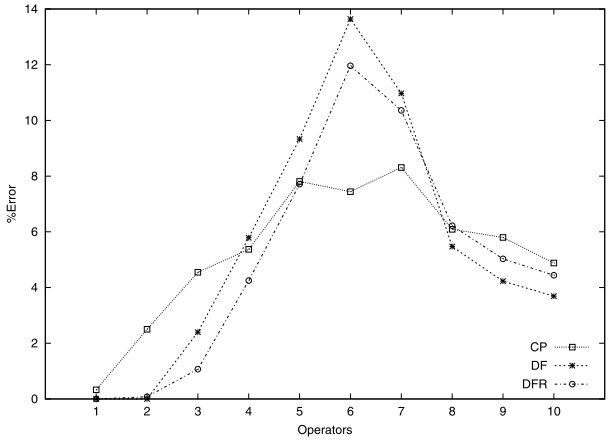
(a) 10×5



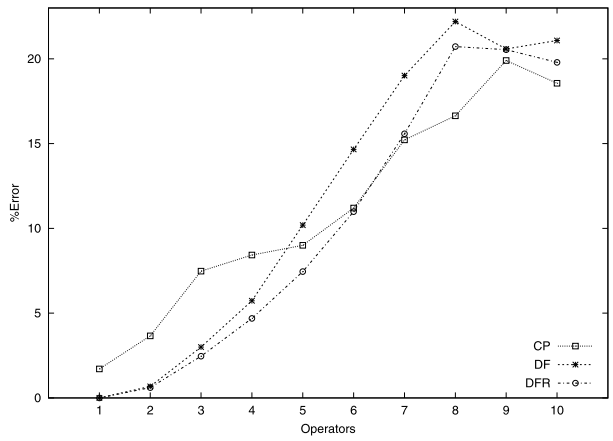
(b) 15×5



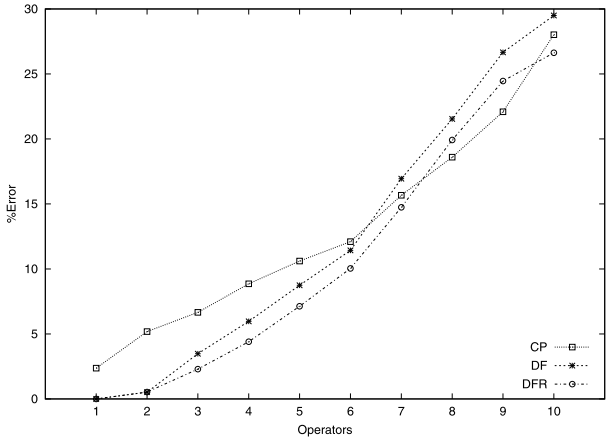
(c) 20×5



(d) 10×10

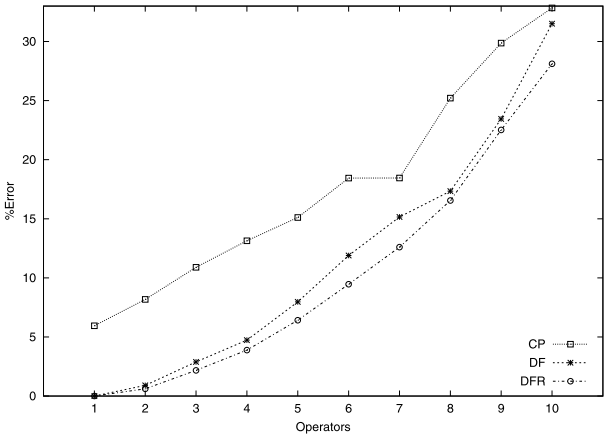


(e) 15×10

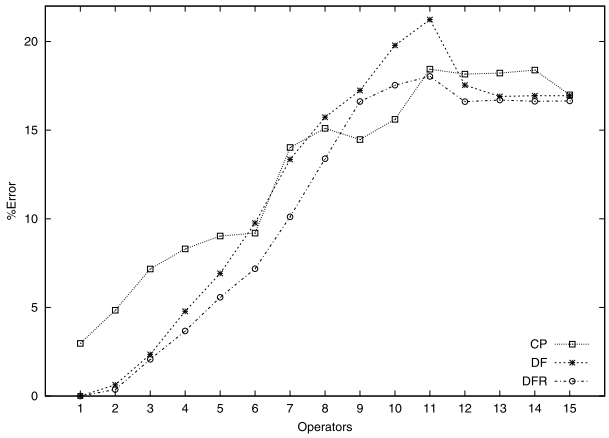


(f) 20×10

Fig. 5. Errors in percentage from *CP*, *DFS* and *DFR* across the *LA* set averaged for instances with the same size and number of operators. The time limit is 300 s.



(g) 30×10



(h) 15×15

Fig. 5. (Continued).

Table 4

p-values from Wilcoxon paired tests of the results from *DF*, *CP* and *DFR* by 300 s when each group of instances with the same size ($n \times m$) is taken as population sample. The alternative hypothesis is that the errors from M1 are smaller than those from M2

Size	M1 / M2	*DF* / *CP*	*DFR* / *DF*	*DFR* / *CP*
10×5		4.13E−03	9.66E−01	2.19E−02
15×5		6.68E−02	7.12E−01	3.36E−02
20×5		4.25E−02	1.66E−02	6.23E−03
10×10		4.54E−01	6.24E−02	1.27E−01
15×10		8.22E−01	5.85E−04	2.04E−02
20×10		4.84E−02	9.92E−06	1.84E−02
30×10		3.89E−07	8.54E−06	3.89E−07
15×15		3.37E−02	1.96E−06	9.29E−04

CP in all cases and *DFR* is the best algorithm for any number of operators.

In order to enhance the conclusions of the experimental comparison we have conducted some statistical analysis. Following [19], we have used paired Wilcoxon tests samples. We have used as alternative hypothesis that the errors from a method M1 are smaller than those from a method M2. So, we have analyzed three combinations of methods (*DF* vs *CP*, *DFR* vs *DF* and *DFR* vs *CP*). The results of these tests are summarized in Table 4 where the p-values are indicated when each group of instances with the same size are taken as population sample. As we can observe, the null hypothesis is rejected at a high level of significance in almost all cases, showing that there is a strong statistical evidence that *DF* outperforms *CP* in 5 out of the 8 subsets of instances (3 of them with the largest instances). Also, the Wilcoxon tests support that *DFR* outperforms *CP* in 7 subsets and it is remarkable that there is a very strong statistical evidence that *DFR* outperforms both *DF* and *CP* in the largest instances.

8. Conclusions

We have seen that the job-shop scheduling problem with operators with the objective of minimizing the total flow time can be efficiently solved by means of a partially informed depth-first search algorithm. The proposed method uses local pruning rules in the expansion mechanism, which is inspired in the $OG\&T$ schedule generation scheme, as it can discard some operations from the canonical set of candidate operations to be scheduled next. And it also uses global pruning rules by keeping in memory some expanded states and then checking every new expanded state against them for some dominance relation. To do that we have given an effective pruning condition that can be efficiently implemented. We have also seen that this algorithm is able to improve to a great extent when it is used with restarts and randomization. We have conducted an experimental study across several benchmarks and the results show that our approach is competitive for solving this problem.

The efficiency of the proposed algorithms relies on a proper combination of elements; some of them are proposed in this paper, while others are taken from previous works. In this sense, the main contributions of this paper are the following:

- We propose a novel heuristic estimation termed h_{OP}. This heuristic is based on a polynomial problem relaxation, so it is admissible and monotonic.

- We formalize the dominance relation *DOM* for the *JSO*(n, p) problem with total flow time minimization and study its properties. In particular we prove it is sound and propose a sufficient condition for it to hold, which depends on the heuristic estimation. This condition allows the algorithm to exploit an efficient rule for pruning states.
- We propose two effective algorithms for solving the *JSO*(n, p) problem. A depth-first search algorithm and a variant that diversifies the search by means of restarts (following Luby's sequence) and randomized heuristics. The efficiency of both algorithms relies on the previous pruning rule.
- We carried out a large experimental study considering instances with different sizes and number of available operators. We analyzed the results using both classical descriptive statistics and advanced inferential statistics, such as Wilcoxon paired tests, showing that our approaches are really effective and that the dominance relation is an important component for this success.

8.1. Future research

From the experimental study, we have identified a number of directions for future research. The first one will be aimed at refining the dominance rules, given by the conditions established in Section 5, so as a larger number of dominance cases can be determined.

Secondly, we plan to explore adaptive models for storing in memory those states which are expected to dominate more states that are expanded after them. Thus, a depth-first search algorithm would further exploit the pruning rules. Concretely, we will try to generalize the method *Symmetry Breaking via Dominance Detection* [17] to be applied with dominance rules more general than symmetry tests. This technique would allow a depth-first search algorithm to exploit a dominance relation using only linear space on the maximum depth of the search tree. In order to use this method, we will need to extend the proposed pruning rules so that two states at different depths in the search tree could be compared efficiently.

The third line of research will be focused on enhancing our algorithm with constraint propagation. This technique has been a powerful tool in constraint reasoning since its inception [35], and has been shown particularly effective for solving scheduling problems [4,5]. For example, *edge finding* algorithms [10,16,30, 55] are able to efficiently detect situations where one operation must be processed the first or the last of a set of operations sharing the same resource in order to improve the current solution. These methods have been much more effective for makespan minimization than for summation cost functions, as the total flow time. Nevertheless, in [28] the authors propose a global constraint involving unary resources and weighted completion time that propagates constraints quite well. So, we plan to use this method for reducing the search space and devising better heuristic estimations for guiding the search, as it was done in [33,34] for the job shop scheduling problem with makespan minimization.

Also, the fact that there are human operators involved in the processing of the operations may reasonably lead to consider uncertainty in the durations of the operations. This way, another interesting line of research will be focused on dealing with uncertain processing times as it was done, for example, in [43] where the durations are modeled with fuzzy numbers, and so developing methods to obtain robust schedules [11,40,41].

Finally, in order to overcome the difficulties suffered by our approach in the cases where there are an intermediate number of available operators, we will try to devise good non-admissible heuristics by computing upper bounds to NP*-hard* relaxed problems. Concretely, we intend to guide the search using those estimations and use admissible heuristics for pruning, so still guaranteeing the admissibility of the method.

Acknowledgements

We are grateful to the anonymous referees for their comments and suggestions, that made it possible to improve this paper. This research has been supported by the Spanish Government under projects MEC-FEDER TIN-20976-C02-01 and TIN-20976-C02-02 and by the Principality of Asturias under Grant FICYT-BP09105.

References

[1] I. Adiri, J. Bruno, E. Frostig and A.H.G. Rinnooy Kan, Single machine flow-time scheduling with a single breakdown, *Acta Informatica* **26** (1989), 679–696.

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

[3] C. Artigues, S. Demassey and E. Néron, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, John Wiley & Sons Incorporated, 2008.

[4] P. Baptiste, C. Le Pape and W. Nuijten, *Constraint-Based Scheduling*, Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[5] R. Barták, M. Salido and F. Rossi, Constraint satisfaction techniques in planning and scheduling, *Journal of Intelligent Manufacturing* **21** (2010), 5–15.

[6] J.E. Beasley, Or-library: Distributing test problems by electronic mail, *J. Oper. Res. Soc.* **41**(11) (1990), 1069–1072.

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

[8] P. Brucker, A. Drexl, R. MÃhring, K. Neumann and E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* **112**(1) (1993), 3–41.

[9] P. Brucker and S. Knust, *Complex Scheduling*, Springer, 2006.

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

[11] L. Climent, M.A. Salido and F. Barber, Robustness in dynamic constraint satisfaction problems, *Int. Journal of Innovative Computing Information and Control* **8**(4) (2012), 2513–2532.

[12] S. Dauzère-Pérès, W. Roux and J.B. Lasserre, Multi-resource shop scheduling with resource flexibility, *European Journal of Operational Research* **107**(2) (1998), 289–305.

[13] E. Demeulemeester and W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, *Management Science* **38**(12) (1992), 1803–1818.

[14] E. Demeulemeester and W. Herroelen, New benchmark results for the resource-constrained project scheduling problem, *Management Science* **43**(11) (1997), 1485–1492.

[15] E.L. Demeulemeester and W.S. Herroelen, *Project Scheduling: A Research Handbook*, International Series in Operations Research & Management Science, Springer, 2002.

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

[17] T. Fahle, S. Schamberger and M. Sellmann, Symmetry breaking, in: *CP*, T. Walsh, ed., Lecture Notes in Computer Science, Vol. 2239, Springer, 2001, pp. 93–107.

[18] B. Gacias, C. Artigues and P. López, Parallel machine scheduling with precedence constraints and setup times, *Computers and Operations Research* **37** (2010), 2141–2151.

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

[20] I.P. Gent, K.E. Petrie and J.F. Puget, Symmetry in Constraint Programming, in: *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier Science Inc., New York, NY, USA, 2006, pp. 327–374, Chapter 10.

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

[22] C.P. Gomes, B. Selman and H.A. Kautz, Boosting combinatorial search through randomization, in: *AAAI/IAAI*, 1998, pp. 431–437.

[23] M.A. González, C.R. Vela, M.R. Sierra and R. Varela, Tabu search and genetic algorithm for scheduling with total flow time minimization, in: *COPLAS 2010*, 2010, pp. 33–41.

[24] W.D. Harvey and M.L. Ginsberg, Limited discrepancy search, in: *IJCAI (1)*, 1995, pp. 607–615.

[25] T. Ibaraki, The power of dominance relations in branch-and-bound algorithms, *Journal of the Association for Computing Machinery* **24**(2) (1977), 264–279.

[26] ILOG CP, *Modeling with IBM ILOG CP Optimizer – Practical Scheduling Examples*, IBM, 2009.

[27] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, *European Journal of Operational Research* **90**(2) (1996), 320–333.

[28] A. Kovács and J. Beck, A global constraint for total weighted completion time for unary resources, *Constraints* **16** (2011), 100–123.

[29] L. Kramer, L. Barbulescu and S. Smith, Understanding performance tradeoffs in algorithms for solving oversubscribed scheduling, in: *Proceedings 22nd Conference on Artificial Intelligence (AAAI-07)*, July, 2007.

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

[31] P. Laborie, IBM ILOG CP optimizer for detailed scheduling illustrated on three problems, in: *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR09)*, 2009, pp. 148–162.

[32] M. Luby, A. Sinclair and D. Zuckerman, Optimal speedup of Las Vegas algorithms, *Inf. Process. Lett.* **47**(4) (1993), 173–180.

[33] C. Mencía, M.R. Sierra and R. Varela, Partially informed depth first search for the job shop problem, in: *Proceedings of ICAPS'2010*, AAAI Press, 2010, pp. 113–120.

[34] C. Mencía, M.R. Sierra and R. Varela, Depth-first heuristic search for the job shop scheduling problem, *Annals of Operations Research* **206** (2013), 264–296.

[35] P. Meseguer, Towards 40 years of constraint reasoning, *Progress in Artificial Intelligence* **1** (2012), 25–43.

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

[37] A. Oddi, R. Rasconi, A. Cesta and S.F. Smith, Iterative improvement algorithms for the blocking job shop, in: *ICAPS*, 2012.

[38] L. Otten and R. Dechter, Anytime and/or depth-first search for combinatorial optimization, *AI Communications* **25**(3) (2012), 211–227.

[39] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[40] N. Policella, A. Cesta, A. Oddi and S.F. Smith, From precedence constraint posting to partial order schedules, *AI Communications* **20** (2007), 163–180.

[41] N. Policella, A. Cesta, A. Oddi and S.F. Smith, Solve-and-robustify, *J. Scheduling* **12**(3) (2009), 299–314.

[42] S. Prestwich and J.C. Beck, Exploiting dominance in three symmetric problems, in: *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004, pp. 63–70.

[43] J. Puente, C.R. Vela and I. González-Rodríguez, Fast local search for fuzzy job shop scheduling, in: *ECAI 2010*, 2010, pp. 739–744.

<uncited> [44] F. Rossi, P. van Beek and T. Walsh, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier Science Inc., New York, NY, USA, 2006.

[45] B. Roy and B. Sussmann, Les problèmes d'ordonnancement avec constraintes disjonctives, Note d.s. no. 9 bis, d6c, SEMA, Matrouge, Paris, 1964.

[46] G. Schmidt, Scheduling with limited machine availability, *European Journal of Operational Research* **5** (2000), 1–15.

[47] M.R. Sierra, Improving heuristic search algorithms by means of pruning by dominance. Application to scheduling problems, PhD thesis, University of Oviedo, 2009.

[48] M.R. Sierra, Improving heuristic search algorithms by means of pruning by dominance. Application to scheduling problems, *AI Communications* **26** (2013), 323–324.

[49] M.R. Sierra, C. Mencía and R. Varela, Optimally scheduling a job-shop with operators and total flow time minimization, in: *CAEPIA 2011, Advances in Artificial Intelligence*, Lecture Notes in Computer Science, Vol. 7023, 2011, pp. 193–202.

[50] M.R. Sierra, C. Mencía and R. Varela, New schedule generation schemes for the job-shop problem with operators, *Journal of Intelligent Manufacturing* (2013), DOI:10.1007/s10845-013-0810-6.

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

[52] A. Sprecher and A. Drexl, Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm, *European Journal of Operational Research* **107**(2) (1998), 431–450.

[53] J.P. Stinson, E.W. Davis and B.M. Khumawala, Multiple resource-constrained scheduling using branch and bound, *AIIE Transactions* **10**(3) (1978), 252–259.

[54] P. van Beek, Backtracking search algorithms, in: *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier Science Inc., New York, NY, USA, 2006, pp. 327–374, Chapter 3.

[55] P. Vilím, R. Barták and O. Čepek, Extension of O($n \log n$) filtering algorithms for the unary resource constraint to optional activities, *Constraints* **10** (2005), 403–425.