Combining Global Pruning Rules with Depth-First Search for the Job Shop Scheduling Problem with Operators

Carlos Mencía[1], María R. Sierra[1], Miguel A. Salido[2], Joan Escamilla[2], and Ramiro Varela[1]

[1] University of Oviedo, (Spain)
{menciacarlos,sierramaria,ramiro}@uniovi.es
[2] Polytechnic University of Valencia, (Spain)
{msalido,jescamilla}@dsic.upv.es

Abstract. We propose an enhanced depth-first heuristic search algorithm to face the job shop scheduling problem with operators. This problem extends the classical job shop scheduling problem by considering a limited number of human operators that assist the processing of the operations. We considered total flow time minimization as objective function which makes the problem harder to solve and more interesting from a practical point of view than minimizing the makespan. The proposed method exploits a schedule generation scheme termed $OG\&T$, two admissible heuristics and some powerful global pruning rules that require recording expanded states. We have conducted an experimental study across several benchmarks to evaluate our algorithm. The results show that the global pruning method is really effective and that the proposed approach is quite competent for solving this problem.

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 [3] 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 makespan in real environments [7] and at the same time it makes scheduling problems harder to solve [15].

To solve the $JSO(n, p)$ problem, we propose a partially informed depth-first search algorithm [24] enhanced with global pruning rules. The definition of the search space and the heuristic estimation are borrowed from [29] where a best-first search algorithm is proposed for the same problem. Besides, a key

Proceedings of the 19th RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2012).
In conjunction with AI*IA 2012, Rome, Italy, June 14–16, 2012.

component of our approach is the combination of depth-first search with a global pruning rule. This rule is defined in accordance with the formal definition given in [16] 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 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 to a given value. When this limit value 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 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 dominance checking. At the same time checking the dominance relations is not a time consuming task thanks to the use of hashing mechanism.

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 [13]. In this context, rules more powerful than those intended for detecting symmetries were also proposed, as for example in [26] 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; this kind of symmetries are detected by the proposed rule. Global pruning rules were also used in some scheduling problems, for example in [23] 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 [29] 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 [20] and it is often used to compare with other approaches to scheduling problems. For example, in [12] 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. This solver is expected to be very efficient for a variety of scheduling problems as it is pointed in [1], 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 [18].

We have conducted an experimental study across conventional instances, considering different number of available operators, to assess our algorithm and also to compare it with the aforementioned approach. The results of this study show clearly that the performance of the proposed depth-first search algorithm relies on the global pruning method and that it may outperform the CP implementation.

The remaining of the paper is organized as follows. In the next section we define the problem. Then, we describe the schedule generation scheme termed $OG\&T$ proposed in [29] which is used to define the search space for the depth-first search algorithm. Then, we describe this search space and the heuristic estimation used to guide the search algorithm. After that, 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. Finally we report the experimental study and we complete the paper with some general conclusions and 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 determined. 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[3]. This problem was first defined in [3] for makespan minimization and is denoted as $JSO(n,p)$.

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. We propose here to use a model for the $JSO(n,p)$ that is similar to that used in [3]. 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*, we introduce nodes to represent operators in this model. So, we have three main types of arcs: job, machine and operator arcs. Each

[3] 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.

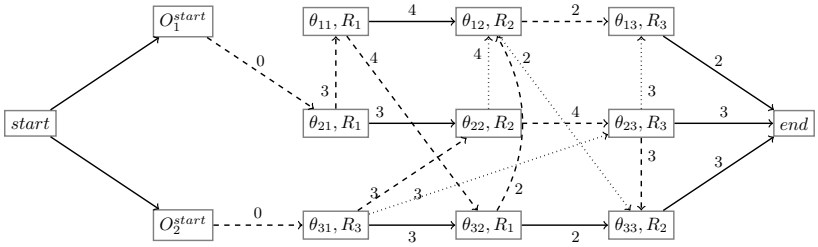

Fig. 1. A feasible schedule to a problem with 3 jobs, 3 machines and 2 operators.

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 *end* node.

In Figure 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.

3 Schedule Generation Schemes

We use here the $OG\&T$ algorithm proposed in [30]. This is a schedule generation scheme for the $JSO(n, p)$ which is an extension of the well-known $G\&T$ algorithm proposed by Giffler and Thompson in [14] for the classical job-shop scheduling problem. 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, $1 \leq i \leq p$. Let SC be the set of scheduled operations at an arbitrary time. Then, the next non-deterministic choice may be any operation of the set A defined as

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

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

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

where t_i, $1 \leq i \leq p$, is the time at which the operator O_i is 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.

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

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 same machine as v^*.
(ii) For each interval $[\tau_i, \tau_{i+1})$ with $m'_i > p'_i$, the operations required by at least $m'_i - p'_i$ machines.

The set of operations obtained in this way is termed B and it is clear that $|B| \leq |A'| \leq |A|$. An important property of this schedule generation scheme is that if the number of operators is large enough, in particular if $p \geq \min(n, m)$ so as $JSO(n, p)$ becomes $J||\Sigma C_i$, it is equivalent to the $G\&T$ algorithm. In [30] a full description of $OG\&T$ is given together with a formal proof of its dominance property.

4 Search Algorithm

As we have pointed out, we use here a partially informed depth-first search algorithm [24]. 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 $f^*(s)$. 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 [31] 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. The optimal solution to this relaxed problem, denoted as $f_{PS}(s)$, is a lower bound on $f^*(s)$. So, we compute $h_{PS}(s) = f_{PS}(s) - g(s)$.

The second heuristic 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 $(P, NC_{ini}||\sum C_i)$, can be optimally solved applying the SPT (Shortest Processing Time) rule [2,28]. 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 $h(s) = \max(h_{PS}(s), h_{OP}(s))$.

Input A state s.
Output The heuristic estimation $h_{OP}(s)$.
  Build a $(P, NC_{ini}||\sum C_i)$ 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;
    Update $t = t + p_\theta$;
  return The total flow time of the built schedule for **P**- $g(s)$;

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

5 Dominance Rules

The effective search tree may be reduced by means of dominance relations among states similar to that exploited in [31] for the classic job-shop scheduling problem. Given two search states s_1 and s_2, s_1 dominates s_2 iff $f^*(s_1) \leq f^*(s_2)$. In general, dominance relations cannot be easily established, but in some particular cases an effective condition for dominance can be defined. For the above search space, an efficient and effective dominance rule is defined as follows. If s_1 and s_2 are states having the same operations scheduled, SC, then s_1 dominates s_2 if the following three conditions hold:

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

where $r_v(s)$ and $av(s)$ denote the head of v 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 i.

From conditions (1) and (3) 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 (2) 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 $t_1 \leq \cdots \leq t_p$ 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, $1 \leq j \leq p$). 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 $t'_1 \leq \cdots \leq t'_p$, 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 $av(s_1) \geq av(s_2)$ iff $t'_{1i} \leq t'_{2i}, 1 \leq i \leq p'$.

The implementation of the dominance rules can be done as follows. When a state s is considered for expansion, s is compared to all the expanded states having the same operations scheduled. 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) and (3) 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) and (3) then $f(s_1) \leq f(s_2)$.*

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.

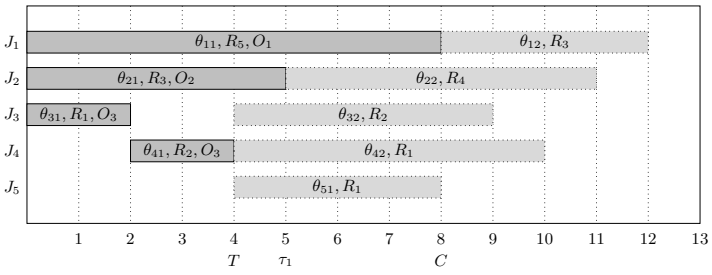

Fig. 2. A partial schedule to a problem with 5 jobs, 5 machines and 3 operators.

So, when a state s is expanded, it has only to be compared with states s' in CLOSED having the same operations scheduled and $f(s') \leq f(s)$.

As both heuristics h_{PS} and h_{OP} are obtained from problem relaxations, the evaluation functions defined as $f_{PS}(s) = g(s) + h_{PS}(s)$ and $f_{OP}(s) = g(s) + h_{OP}(s)$ fulfill the condition of Proposition 1. As $f(s) = \max(f_{PS}(s), f_{OP}(s))$, the condition may be evaluated on f, due to the fact that $f(s_1) > f(s_2)$ implies that at least one of the conditions $f_{PS}(s_1) > f_{PS}(s_2)$ or $f_{OP}(s_1) > f_{OP}(s_2)$ holds.

To demonstrate the application of this rule we can consider a search state similar to that of Figure 2 with the same operations scheduled, but exchanging the order of operations θ_{31} and θ_{41} on the machine R_1. 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.

6 Computational Results

The purpose of the experimental study is to assess our proposal (DF) and to compare it with an implementation on IBM ILOG CPLEX CP Optimizer tool (CP). In this 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 560 instances in all. The first one is that proposed in [3], 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 (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 [29].

The second benchmark includes conventional instances taken from the OR-library [6]: small instances as $LA01 - 05$ (10 jobs $\times 5$ machines), medium size instances as $LA06 - 10$ (15×5), $LA11 - 15$ (20×5), $LA16 - 20$ (10×10) and large instances as $LA36 - 40$ (15×15). 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 has been given a memory limit of 4 GB to store expanded states when exploiting the global pruning rule. As CP is non-deterministic, we report the average results across 10 executions. The target machine was Intel Xeon (2,26 GHz), 24 GB RAM. The algorithms are coded in C++.

We have solved all the instances with CP and DF in two different modes: without pruning (DF_{NP}) and with pruning (DF_P). 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}. For each algorithm, we report the time taken in seconds (T), the number of solved instances (#Sol) and the mean relative error in percentage w.r.t. the optimal solution (%Err). Additionally, we report the average number of expanded nodes for both DF algorithms (#Exp). As it can

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
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.	0,00	0,00	0,00	0,49	1,56	1,64
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	630215,27	498189,73	393827,57
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	34635,27	120805	260828,43

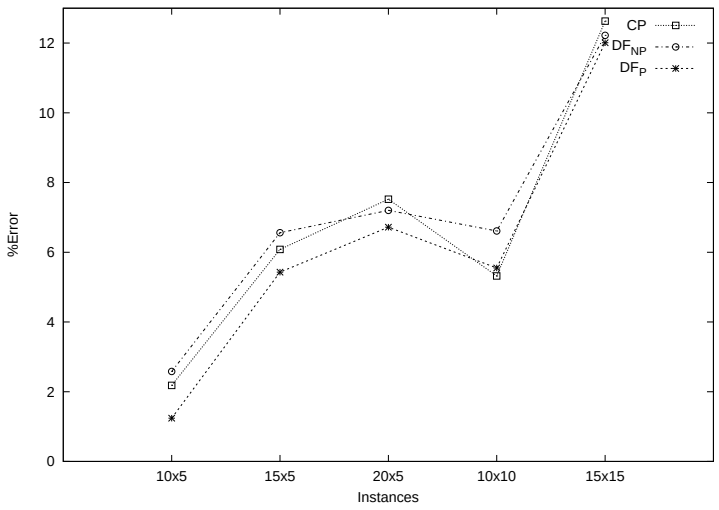

Fig. 3. Errors in percentage obtained with the three algorithms CP, DF_{NP} and DF_P averaged for the LA instances with the same size.

be observed, the small instances (SMALL) were easily solved by the three algorithms, but DF_P took the lowest time and expanded much less states than DF_{NP}. It was in the large instances (LARGE) where there were significant differences among the algorithms. CP and DF_{NP} were only able to certify the optimality in 4 and 2 instances respectively, while DF_P certified the optimality in all but 2 instances and consequently it also layouts much lower error in percentage. Also, the number of expanded states was in average almost one order of magnitude lower with DF_P than it was with DF_{NP}, what makes it clear the utility of the global pruning method.

For the second set of instances neither of the algorithms can reach the optimal solution in most of the cases. Figure 3 summarizes the error in percentage of the solutions reached by the three methods. These errors were computed w.r.t. the lower bounds obtained by the best-first algorithm given in [29] after 300 s. As we can observe, DF reached better solutions when exploiting the global dominance rule, and in this case it was better in average than CP for 4 subsets of instances with the same size while it was worse for the subset of 10×10 instances.

Tables 2 and 3 report results from CP and DF_P for the subsets with 5 machines and 10 machines respectively of the second benchmark averaged for the instances with the same number of operators (#Op) in each subset. For these instances CP reached the time limit of 300 s. without certifying any optimal solution, while DF_P could certify the optimal solution for a number of instances with 5 machines, this number decreasing with the number of jobs. Also, the error in percentage was lower for DF_P in about 1,76; 1,12 and 1,12 for instances with 5, 10 and 15 machines respectively in average w.r.t. CP.

Table 2. Summary of results from the LA instances with 5 machines.

	$10jobs$				$15jobs$				$20jobs$			
	CP		DF_P		CP		DF_P		CP		DF_P	
#Op.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.
1	0/5	0,01	5/5	0,00	0/5	0,78	5/5	0,00	0/5	1,47	5/5	0,00
2	0/5	2,04	5/5	0,00	0/5	3,14	1/5	0,26	0/5	4,41	0/5	0,67
3	0/5	4,54	0/5	0,86	0/5	6,34	0/5	5,30	0/5	7,78	0/5	5,09
4	0/5	4,09	0/5	5,36	0/5	9,76	0/5	11,33	0/5	11,69	0/5	12,61
5	0/5	0,21	5/5	0,00	0/5	10,38	0/5	10,28	0/5	12,25	0/5	15,24
Avg. Err.		2,18		1,24		6,08		5,43		7,52		6,72

Table 3. Summary of results from LA instances with 10 jobs and 10 machines.

	CP		DF_P	
#Op.	#Sol.	%Err.	#Sol.	%Err.
1	0/5	0,33	5/5	0,00
2	0/5	2,50	5/5	0,00
3	0/5	4,55	0/5	2,40
4	0/5	5,37	0/5	5,79
5	0/5	7,81	0/5	9,33
6	0/5	7,44	0/5	13,64
7	0/5	8,31	0/5	10,97
8	0/5	6,09	0/5	5,47
9	0/5	5,80	0/5	4,23
10	0/5	5,06	0/5	3,69
Avg. Err.		5,32		5,55

Table 3 reports the results for LA instances with 10 jobs and 10 machines. In this case, the average error was lower for DF_P than it was for CP, and only DF_P was able to certify the optimality of the solutions for the instances with one or two operators. However, the difference was lower than it was for the LA instances with 5 machines. In this case, CP was better than DF_P for the instances with 4, 5, 6 and 7 operators. We conjecture that the reason for the decreasing of performance for DF_P w.r.t. CP in these instances is due to the heuristic estimation being worse for an intermediate number of operators than it being for a small or a large number. This is quite reasonable as the heuristic h_{OP} is expected to return better estimations when the number of operators is small and so these are the most critical resources. On the contrary, the heuristic h_{PS} is expected to be better when the number of operators is large and so the critical resources are the machines as it happens in the classical job shop problem.

Similar behavior can be observed in Figure 4 which summarized the results for the largest instances with 15 jobs and 15 machines. As we can see, DF_P is also the best one when the number of available operators is small or large, but it is outperformed by CP for some intermediate values of p. At the same time,

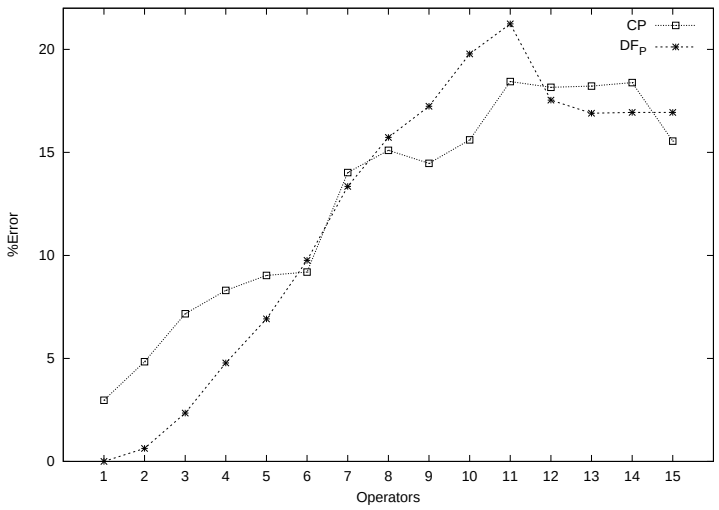

Fig. 4. Errors in percentage obtained with CP and DF_P from the LA instances with 15 jobs and 15 machines.

CP shows a more uniform behavior regarding the quality of the solutions found due to diversifying the search thanks to the use of restarts.

From this study, we claim that the proposed global pruning method, combined with the $OG\&T$ algorithm and the heuristics h_{PS} and h_{OP} allows a classic depth-first search algorithm to be quite competitive with a powerful solver such as CP Optimizer, using depth-first search enhanced with restarts, in solving the $JSO(n, p)$ problem with total flow time minimization. Also, this algorithm may be improved with a more sophisticated recording scheme.

7 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 conducted an experimental study across several benchmarks and the results show that our approach is competitive for solving this problem.

7.1 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 rule, given by the three 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* [11] 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 [22], and has been shown particularly effective for solving scheduling problems [4,5]. For example, *edge finding* algorithms [8,10,19,32] 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 [17] 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 [21] 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 [27] where the durations are modeled with fuzzy numbers, and so developing methods to obtain robust schedules [25,9].

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 guaranteeing the admissibility of the method. In the same context, we also plan to exploit strategies that diversify the search among promising regions of the search space, such as depth-first search with restarts. We already have some preliminary results with both approaches that seem very promising.

Acknowledgments

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. Modeling with IBM ILOG CP Optimizer - practical scheduling examples. *IBM*, 2009.
2. I. Adiri, J. Bruno, E. Frostig, and A. Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26:679–696, 1989.
3. A. Agnetis, M. Flamini, G. Nicosia, and A. Pacifici. A job-shop problem with one additional resource type. *Journal of Scheduling*, 14(3):225–237, 2010.
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*, 21:5–15, 2010. 10.1007/s10845-008-0203-4.
6. J. E. Beasley. Or-library: Distributing test problems by electronic mail. *J Oper Res Soc*, 41(11):1069–1072, 1990.
7. P. Brucker and S. Knust. *Complex Scheduling*. Springer, 2006.
8. J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26:269–287, 1990.
9. L. Climent, M. A. Salido, and F. Barber. Robustness in dynamic constraint satisfaction problems. *Int. Journal of Innovative Computing Information and Control*, 8(4):2513–2532, 2012.
10. U. Dorndorf, E. Pesch, and T. Phan-Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122:189–240, 2000.
11. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2001.
12. B. Gacias, C. Artigues, and P. Lopeza. Parallel machine scheduling with precedence constraints and setup times. *Computers and Operations Research*, 37:2141–2151, 2010.
13. I. Gent, K. Petrie, and J. Puget. *Chapter 10. Symmetry in Constraint Programming*, pages 327–374. Elsevier Science Inc., New York, NY, USA, 2006.
14. B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.
15. 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*, pages 33–41, 2010.
16. T. Ibaraki. The power of dominance relations in branch-and-bound algorithms. *Journal of the Association for Computing Machinery*, 24(2):264–279, 1977.
17. A. Kovács and J. Beck. A global constraint for total weighted completion time for unary resources. *Constraints*, 16:100–123, 2011. 10.1007/s10601-009-9088-x.
18. 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.

19. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artif. Intell.*, 143(2):151–188, 2003.
20. 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)*, pages 148–162, 2009.
21. C. Mencía, M. R. Sierra, and R. Varela. Partially informed depth first search for the job shop problem. In *Proceedings of ICAPS'2010*, pages 113–120. AAAI Press, 2010.
22. P. Meseguer. Towards 40 years of constraint reasoning. *Progress in Artificial Intelligence*, pages 1–19, 2012. 10.1007/s13748-011-0006-2.
23. 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*, 112:347–366, 1999.
24. J. Pearl. *Heuristics: Intelligent Search strategies for Computer Problem Solving*. Addison-Wesley, 1984.
25. N. Policella, A. Cesta, A. Oddi, and S. F. Smith. Solve-and-robustify. *J. Scheduling*, 12(3):299–314, 2009.
26. S. Prestwich and J. C. Beck. Exploiting dominance in three symmetric problems. In *in: Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, pages 63–70, 2004.
27. J. Puente, C. R. Vela, and I. González-Rodríguez. Fast local search for fuzzy job shop scheduling. In *ECAI 2010*, pages 739–744, 2010.
28. G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 5:1–15, 2000.
29. M. R. Sierra, C. Mencía, and R. Varela. Optimally scheduling a job-shop with operators and total flow time minimization. *CAEPIA 2011, Advances in Artificial Intelligence, Lecture Notes in Computer Science*, 7023:193–202, 2011.
30. M. R. Sierra, C. Mencía, and R. Varela. Searching for optimal schedules to the job-shop problem with operators. *Technical report. Computing Technologies Group. University of Oviedo*, 2011.
31. 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):111–119, 2010.
32. P. Vilím, R. Barták, and O. Čepek. Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints*, 10:403–425, 2005. 10.1007/s10601-005-2814-0.