

This article was downloaded by: [UOV University of Oviedo]

On: 31 October 2014, At: 03:38

Publisher: Taylor & Francis

Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK



International Journal of Production Research

Publication details, including instructions for authors and subscription information:

<http://www.tandfonline.com/loi/tprs20>

An efficient hybrid search algorithm for job shop scheduling with operators

Carlos Mencía^a, María R. Sierra^a & Ramiro Varela^a

^a Department of Computing, University of Oviedo, Gijón, Spain

Published online: 29 Jul 2013.

To cite this article: Carlos Mencía, María R. Sierra & Ramiro Varela (2013) An efficient hybrid search algorithm for job shop scheduling with operators, International Journal of Production Research, 51:17, 5221-5237, DOI: [10.1080/00207543.2013.802389](https://doi.org/10.1080/00207543.2013.802389)

To link to this article: <http://dx.doi.org/10.1080/00207543.2013.802389>

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the "Content") contained in the publications on our platform. However, Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Any opinions and views expressed in this publication are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor and Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden. Terms & Conditions of access and use can be found at <http://www.tandfonline.com/page/terms-and-conditions>

An efficient hybrid search algorithm for job shop scheduling with operators

Carlos Mencía, María R. Sierra and Ramiro Varela*

Department of Computing, University of Oviedo, Gijón, Spain

(Received 21 August 2012; final version received 22 April 2013)

We confront the job shop scheduling problem with human operators and total flow time minimisation. To solve this problem we propose a hybrid search algorithm that interleaves best-first and depth-first search. The efficiency of this algorithm relies on the proper combination of diversification and intensification capabilities of best-first and depth-first searches, respectively, as well as two admissible heuristics and some global pruning rules designed from problem domain knowledge. We conducted an experimental study on a benchmark set with small-, medium- and large-size instances. The results of this study show that the algorithm is efficient thanks to the combination of all its components and that it compares favorably with an outstanding constraint programming solver specialised in scheduling problems.

Keywords: job shop scheduling with operators; total flow time; best-first search; depth-first search; hybrid search algorithms; anytime algorithms; global pruning rules

1. Introduction

The job shop scheduling problem (JSP) with an additional resource type and with the objective of minimising the makespan has been recently proposed by Agnetis et al. (2011) where it is denoted $JSO(n, p)$. This problem is motivated by manufacturing processes in which part of the work is done by human operators sharing the same set of tools. The problem is formalised as a classic JSP in which the processing of an operation on a given machine requires the assistance of one of p available operators.

Agnetis et al. (2011) made a thorough study of this problem and established the minimal NP -hard cases. Also, they proposed a number of exact and approximate algorithms to cope with this problem which are evaluated across a set of instances generated from the minimal relevant cases. The results of their experimental study show that instances with 3 jobs, 3 machines, 2 operators and a number of 30 operations per job may be hard to solve to optimality.

The classic JSP has interested researchers over the last decades because of its difficulty and its proximity to real life problems. In some cases the pure JSP is able to model exactly a real problem, for example Meeran and Morshed (2012) model a work shop at a steel mill as a classic JSP. However, in most cases, some additional characteristics are required in order to model a particular problem. So, inspired by different problems, a number of variants of the JSP have been considered in the literature. To name just a few recent ones, we can consider the following. Driebel and Monch (2012) consider a JSP in which an operation can be processed in one from a set of identical parallel machines. Liu et al. (2012) introduce a job value function that deteriorates over time for weighting the total flow time of sub-jobs. Niu et al. (2012) consider different operation modes so that the time taken and the cost spent by one operation depend on the mode in which it is performed. Also, Rabiee et al. (2012) consider a JSP where each operation may be processed by any from a subset of the machines with the objective of minimising the makespan and the total operation cost.

Zouba et al. (2009) remark that research in classical scheduling theory has mainly concentrated on machine and job characteristics such as machine availability or job processing sequences; and that the models have ignored in most of the cases considerations linked with operator interventions on the machines by assuming their number to be infinite. However, human resources are getting more and more critical in production systems, and it is not enough to concentrate only on the physical resources in order to expect a reasonable solution. So, they are being considered in some recent models. For instance, Artigues et al. (2009) and Guyon et al. (2012) consider integrated employee-timetable and job shop scheduling problems, in which human operators with different skills have to be assigned to machines and shifts in a job shop production line. In these problems, the objective is minimising the total cost of the assignments. Ruiz-Torres et al. (2012) model a real JSP in pharmaceutical industry; the resources are technicians with different skills and only a subset of them can be working in parallel.

*Corresponding author. Email: ramiro@uniovi.es

In this paper we deal with the $JSO(n, p)$ with the objective of minimising the total flow time. This objective function is often more interesting than the makespan in real environments (Brucker and Knust 2006) and at the same time it makes scheduling problems harder (González et al. 2010). As it is pointed out by Framinan and Leisten (2003), total flow time is one of the most important performance measures, as it can lead to stable utilisation of resources, rapid completion of some jobs and so minimising the work-in-progress management cost. Therefore it is a very important objective in many environments, for instance, electronics, chemicals, textile, food and service industries (Lin et al. 2012).

To solve the $JSO(n, p)$ problem we propose a hybrid algorithm that interleaves best-first and depth-first searches. The objective is exploiting the advantages of these heuristic search algorithms and avoiding some of their inconveniences at the same time. Both the search strategies are exhaustive so that they may eventually come up with a proven optimal solution. Best-first diversifies the search as it keeps open many paths towards candidate solutions and so it requires an exponential amount of memory. For this reason, it often fails to reach a solution due to the memory getting exhausted; nevertheless it may return a tight lower bound when a solution is not reached. On the other hand, depth-first intensifies the search on a particular region of the search space and so it can take a very long time to reach an optimal or near optimal solution.

We propose to combine best-first and depth-first search so that the second is issued from some of the search states selected by the first to be expanded next. In this way, we expect to achieve a proper balance between diversification and intensification so that the combined algorithm comes eventually with near optimal solutions and tight lower bounds at the same time. The equilibrium between search intensification and diversification effort is the key for success in other search algorithms such as hybrid metaheuristics (Talbi 2009; Glover and Laguna 1993), where a global searcher such as an evolutionary algorithm is often combined with some local searcher such as tabu search or path relinking to obtain outstanding algorithms (Vela et al. 2010; González et al. 2012). One advantage of the proposed combination is that it may reach near optimal solutions and tight lower bounds, and therefore it may certify the optimality of solutions at difference of hybrid metaheuristics.

The algorithm is guided by two heuristic estimations, which rely on two problem relaxations, and it is enhanced with some global pruning rules. The use of these rules require bookkeeping expanded states in order for each new expanded state to be compared with some of the states expanded previously for some dominance relation. As we will see in the experimental study, the pruning method is a key component of the proposed algorithm. In spite of the fact that it consumes memory, it allows the algorithm to reduce drastically the effective search space.

To assess the performance of the algorithm and to compare it with other methods, we have conducted an experimental study in which we analyzed the contribution of each component to the algorithm's performance and compared the algorithm with an implementation on IBM CPLEX CP Optimiser (CP). This is a commercial solver embedding constraint propagation rules and local search methods dedicated to scheduling (Laborie 2009; IBM 2009) and it is often used to compare with other approaches to scheduling problems (Gacias et al. 2010). The results of this study show that the hybrid algorithm is efficient, thanks to the proper combination of its components and that it compares favorably with CP.

The remainder of the paper is organised as follows. In Section 2 we define the problem and propose a disjunctive model for it. Then, in Section 3, we give some background on heuristic search and describe the proposed hybrid algorithm termed A*DFS. In Section 4 we show how the proposed hybrid algorithm is adapted for the $JSO(n, p)$. The design and results of the experimental study are reported in Section 5. Finally, the main conclusions of the paper and some guidelines for further research are given in Section 6.

2. The job shop scheduling problem with operators

Formally the JSP with operators can be defined as follows. We are given a set of n jobs $\{J_1, \dots, J_n\}$, a set of m resources or machines $\{R_1, \dots, R_m\}$ and a set of p operators $\{O_1, \dots, O_p\}$. Each job J_i consists of a sequence of v_i operations or tasks $(\theta_{i1}, \dots, \theta_{iv_i})$. 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 pre-emption 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 to find a feasible schedule that minimises the sum of the completion times of all jobs, i.e. the total flow time. This problem was first defined by Agnetis et al. (2011) for makespan minimisation 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 by Agnetis et al. (2011). 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 , for each job J_i , we introduce nodes O_j^{start} , $1 \leq j \leq p$, to represent operators in this model. So, we have three main types of arcs: job,

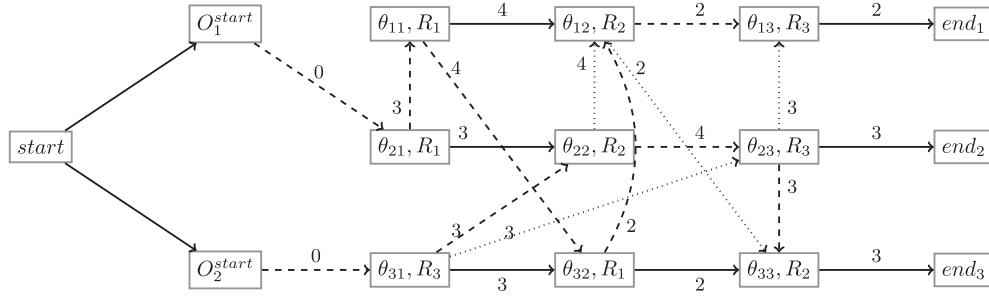


Figure 1. A feasible schedule to a problem with 3 jobs, 3 machines and 2 operators.

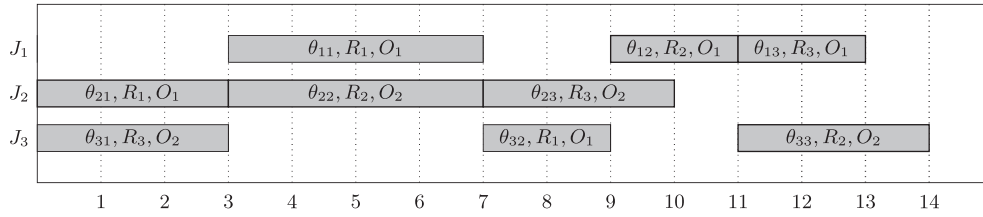


Figure 2. Gantt chart of the schedule represented in Figure 1.

machine and operator arcs. Each type defines 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 J_i to the corresponding end_i node. A critical path for job J_i is a longest weighted path from node *start* to node end_i . The total flow time of the solution is obtained by adding the cost of a critical path for each job J_i .

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 graph drawing, only the operator arc is drawn when there are two arcs between the same pair of nodes. Continuous arrows represent job arcs and dotted arrows represent machine arcs. In this example, the costs of the critical paths for jobs J_1 , J_2 and J_3 are 13, 10 and 14, respectively, so the total flow time of the schedule is 37.

Figure 2 shows a Gantt chart for the schedule represented by Figure 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.

3. The hybrid heuristic search algorithm A*DFS

In this section we start with a review of the basic notions about state space heuristic search and then introduce the hybrid algorithm proposed in this paper.

3.1 Heuristic search

A search problem is given by a quintuple (S, Γ, s, SUC, c) , where S is the set of states or nodes, $s \in S$ is the initial state, $\Gamma \subseteq S$ is the set of goals, SUC is the transition operator such that for a state $n \in S$, $SUC(n)$ returns the set of successor states of n , and each transition from n to $n' \in SUC(n)$ has a non negative cost $c(n, n')$. The states and the transitions between states conform to a graph, which is called *search space*. We consider here the simple case where the search space is a tree, as this structure of the search space naturally arises in a variety of combinatorial optimisation problems, especially scheduling and manufacturing problems. P_{s-n} denotes the path from s to n and $g(n)$ the cost of this path.

Starting from s and applying systematically the operator SUC , a search algorithm looks for an optimal solution,¹ i.e. a path P_{s-o} with $o = \arg \min \{g_{P_{s-o'}} | o' \in \Gamma\}$, the cost of this solution is denoted C^* . For this purpose, a tree search algorithm maintains the list OPEN to store those states that have been generated but not yet expanded. This list represents the frontier between the explored and unexplored regions of the search space. At each step, the search algorithm selects for expansion the

first state from the list OPEN. So, the criterion used to sort the nodes in OPEN, namely the *search strategy*, has implications on the completeness, admissibility and efficiency of the algorithm.

The performance of a search algorithm can be dramatically improved by exploiting the knowledge from the problem domain. This knowledge may be used for guiding the search towards promising regions of the search space, so reaching a goal state without the need of exploring every path in the search tree. This can be done by means of a heuristic evaluation function f , such that for each state n , $f(n)$ gives a measure of its promise. The properties of this function and the way it is used determine the properties of the search algorithm.

3.1.1 Best-first search

The heuristic knowledge represented by f can be fully exploited if the nodes in OPEN are sorted by non-decreasing f -values. This search strategy is called the *best-first search*. The function f is usually defined such that $f(n)$ is an estimation of $f^*(n)$, which is the cost of the best solution reachable from n , i.e. $f^*(n) = g(n) + h^*(n)$, where $h^*(n)$ is the optimal cost from n to its nearest goal. This way, the evaluation function can be defined as $f(n) = g(n) + h(n)$ for all states n , where $h(n)$ is a heuristic estimation for $h^*(n)$. In this case, we have the A* algorithm proposed by Hart et al. (1968) and very well described by Nilsson (1980) and Pearl (1984).

The A* algorithm has some interesting properties that depend on the function h . When $h(n) \leq h^*(n)$ for all states n , the algorithm is admissible and then it guarantees that the first goal reached represents an optimal solution. Moreover, if we have two admissible heuristics h_1 and h_2 such that $h_1(n) < h_2(n)$ for all non goal states n , h_2 is more informed than h_1 and then the number of nodes expanded by A* to reach a solution with h_2 is not greater than it is with h_1 .

3.1.2 Depth-first search

The main inconvenience of best-first search algorithms is the large amount of space they require for the list OPEN what, in practice, may get the memory of the target machine exhausted before reaching a solution even for small, or medium-size instances. For this reason a common alternative is to use *depth-first search* (DFS). In this case the OPEN list is managed as a LIFO structure, i.e. the successors of the expanded state n are inserted at the beginning of OPEN. If these nodes are locally sorted by their f -values, then the search strategy is termed partially-informed depth-first search (Pearl 1984; Mencía et al. 2010; Mencía et al. 2012). This strategy only requires a linear consumption of memory. It usually finds a suboptimal solution quickly and then a sequence of improving solutions, i.e. it is an any-time search strategy. At the same time, nodes n such that $f(n) \geq UB$, UB being the cost of the best solution found so far, i.e. the incumbent, can be pruned as they do not lead to an improving solution. This pruning is a key component of *branch and bound* algorithms. In the long term DFS can come up with an optimal solution after the whole search space is explored. However, it often gets stuck in low promising areas of the search space due to the local heuristic guidance.

3.2 The hybrid algorithm: A*DFS

Our proposal here is to combine both strategies into a hybrid algorithm, termed A*DFS, to exploit the exploration capability of best-first search and the intensification capability of depth-first search. Algorithm 1 provides a pseudo-code for A*DFS, and Algorithm 2 shows the DFS component. The hybrid algorithm consists of two consecutive phases: The first one starts in the beginning and completes when a memory limit is reached. In this phase, the algorithm alternates between best-first (A*) and depth-first searches. Concretely, it starts in A* mode; then, after every N expanded nodes, a limited depth-first search is issued from the next selected node n just before being expanded by A* (Algorithm 1, lines 15-22). The depth-first search from n finishes after expanding E states and the algorithm turns again into A* mode expanding n . When the memory limit is reached, A*DFS shifts to a second phase, in which an exhaustive depth-first search (without limiting the number of expansions) is carried out from each state that was generated but not expanded by A*. In both phases, the search is guided by f -values and when a state n is generated such that $f(n) \geq UB$, n is pruned.

Combining A* and depth-first search has given promising results for solving shortest-path problems,² in which the search space is a graph with cycles and the solutions are located at unknown and unbounded depths. Stern et al. (2010) propose a combination in which an depth-first search limited to a maximum depth is yielded from each node to be expanded by an A* algorithm. This approach lets the algorithm save a lot of memory, thanks to the depth-first search component and so it can solve more problem instances. However, limiting the depth of the depth-first searches is not appropriate for a search tree where the solutions are located at a known depth. This is the case of many scheduling problems, including the one we face in this paper.

3.2.1 Phase 1. Interleaving A* with limited depth-first searches

In the first phase, the integration of A* with DFS is done in the following way. When running in A* mode, the algorithm keeps an OPEN-A* list with nodes sorted by f -values. However, each time the algorithm turns into the DFS mode, a local OPEN list is created. This list is initiated with n and it is finally erased after DFS completes E expansions. At this time, the algorithm turns into A* mode for the next N expansions, n being the first node expanded in this step as it is the first one in OPEN-A*. This search strategy produces some node reexpansions, but at the same time it helps to prevent the algorithm from running out of memory too early, and therefore it can issue more intensification steps in different regions of the search space.

The parameters N and E are relevant in order to reach a proper balance between exploration and exploitation. A very small value of N may lead the algorithm to sample quite similar regions of the search space, so achieving little diversification. On the contrary, a very large N could lead to a very small number of intensification steps, hence reducing the ability of the algorithm to sample different regions. Clearly, the value of E must be at least the distance from the node n to a leaf node, but it should be larger than this to explore more than one solution in the region under the node n . However, a very large E may require the algorithm to spend too much time in each intensification.

Finally, as it is common in metaheuristics, we try to adapt the intensification effort of the depth-first searches to the promising degree of the search tree under node n . To do that, we restart the counter of expansions at 0 each time the algorithm reaches an improving solution when running in the DFS mode, so keeping the search in this mode for at least E more expansions (**Algorithm 2, lines 9–13**).

3.2.2 Phase 2. Finishing with exhaustive depth-first search

Once the memory limit is reached, A*DFS turns to the second phase. Here, a depth-first search is performed from each node stored in the OPEN-A* list. As this list is sorted by non-decreasing f -values, DFS traverses first the subtrees under the most promising nodes generated in the previous phase in A* mode, so it is more likely to find good solutions and to improve the lower bounds soon. So, DFS operates on the OPEN-A* list and in this way, A*DFS runs always safe from memory getting exhausted.

The algorithm completes when either OPEN-A* gets empty or the minimum f -value in OPEN-A* is not lower than the cost of incumbent solution.

3.2.3 Discussion

The proposed combination of best-first and depth-first search is expected to take profit from the benefits of both strategies and overcome some of their drawbacks.

Firstly, the use of A* in the first phase of A*DFS provides global guidance towards promising regions in the search space and allows the algorithm to improve the lower bounds over time. Then DFS intensifies the search on some of these regions without getting stuck in any of them due to the limited number of expansions; hopefully, in these searches new solutions improving the incumbent may be found. Also, developing exhaustive DFS from the nodes in OPEN-A* in the second phase of the algorithm makes A*DFS able to run for a long time without increasing the memory consumption and to explore the most promising subtrees first, so improving both lower and upper bounds.

Regarding the formal properties of A*DFS, it is clear that when a new solution is reached in the DFS mode, this solution may not be optimal. However, if the heuristic function h is admissible, a solution is optimal if it is reached when running in A* mode (**Algorithm 1, lines 9–13**). This is a consequence of OPEN-A* being sorted by non-decreasing f -values.

Also, at any time, the lowest f -value in OPEN-A* and OPEN lists is a lower bound on the optimal solution. Here, it is worth doing some remarks regarding the computation of lower bounds. As we can observe in Algorithm 2, DFS searches for solutions with cost less than UB in the subtree under the node $state$, and returns a lower bound on the cost of the best solution in that subtree. So, if DFS is able to traverse the whole subtree without finding any improving solution, UB is a lower bound as it is proven that no solution with less cost exists from $state$. On the other hand, if the subtree is completely explored finding an improving solution, its cost $UB_{DFS} < UB$ will be a lower bound, as it is the best solution reachable from $state$. In other cases, the lower bound is defined by the least f -value in OPEN. These cases are shown in lines 19–23. A*DFS takes profit from the lower bounds returned by DFS, as depicted in lines 17 and 33 of Algorithm 1. In these cases, the least f -value in OPEN-A* will be given by the first state, so it can be computed very efficiently.

The expected behaviour of A*DFS is to return a sequence of improving solutions, as it DFS does, but this sequence having a larger rate of improvement than that of the sequence returned by single DFS. Also, it is expected to return lower bounds as accurate as A*, or even better, and to certify more optimal solutions as it can run indefinitely.

Algorithm 1 A*DFS**Require:** $start, N, E, memoryLimit, timeLimit$.**Ensure:** Lower (LB) and upper (UB) bounds on the optimal solution. A solution ($incumbentSolution$) with cost UB (if exists and is reached by $timeLimit$).*Phase 1: Interleaving A* with limited Depth First Search*

```

1:  $LB \leftarrow f(start)$ ;
2:  $UB \leftarrow \infty$ ;
3:  $incumbentSolution \leftarrow null$ ;
4:  $OPEN\_A^* \leftarrow \{start\}$ ;
5:  $\#expandedNodes \leftarrow 0$ ;
6: while not ( $empty(OPEN\_A^*)$ ) and ( $LB < UB$ ) and not ( $memoryLimit$  or  $timeLimit$  reached) do
7:    $n \leftarrow pop(OPEN\_A^*)$ ;
8:    $LB \leftarrow f(n)$ ;
9:   if  $isGoal(n)$  then
10:      $UB \leftarrow f(n)$ ;
11:     Update  $incumbentSolution$  from  $n$ ;
12:     return  $incumbentSolution, LB, UB$ ; {optimal}
13:   end if
14:   {Every  $N$  node expansions a limited DFS is issued}
15:   if ( $\#expandedNodes \bmod N = 0$ ) then
16:     ( $newSolution, LB_{DFS}, UB_{DFS}$ )  $\leftarrow DFS(n, E, UB, timeLimit)$ ;
17:      $LB \leftarrow \min(LB_{DFS}, \min\{f(q) \mid q \in OPEN\_A^*\})$ ;
18:     if  $UB_{DFS} < UB$  then
19:        $incumbentSolution \leftarrow newSolution$ ;
20:        $UB \leftarrow UB_{DFS}$ ;
21:     end if
22:   end if
23:   if  $f(n) < UB$  then
24:      $S \leftarrow SUC(n)$ ;
25:      $OPEN\_A^* \leftarrow merge(S, OPEN\_A^*)$ ; {According to  $f$ -values}
26:      $\#expandedNodes \leftarrow \#expandedNodes + 1$ ;
27:   end if
28: end while
29: Phase 2: Finishing with exhaustive depth-first search
30: if  $memoryLimit$  reached then
31:   while not ( $empty(OPEN\_A^*)$ ) and ( $LB < UB$ ) and not ( $timeLimit$  reached) do
32:      $n \leftarrow pop(OPEN\_A^*)$ ;
33:     ( $newSolution, LB_{DFS}, UB_{DFS}$ )  $\leftarrow DFS(n, \infty, UB, timeLimit)$ ;
34:      $LB \leftarrow \min(LB_{DFS}, \min\{f(q) \mid q \in OPEN\_A^*\})$ ;
35:     if  $UB_{DFS} < UB$  then
36:        $incumbentSolution \leftarrow newSolution$ ;
37:        $UB \leftarrow UB_{DFS}$ ;
38:     end if
39:   end while
40: end if
41: return  $incumbentSolution, LB, UB$ ;

```

Algorithm 2 DFS**Require:** $state, E, UB, timeLimit$.**Ensure:** Lower (LB_{DFS}) and upper (UB_{DFS}) bounds in the subtree rooted with $state$. A solution ($newSolution$) of the subtree if $UB_{DFS} < UB$.

```

1:  $LB_{DFS} \leftarrow f(state)$ ;
2:  $UB_{DFS} \leftarrow \infty$ ;
3:  $newSolution \leftarrow null$ ;
4:  $OPEN \leftarrow \{state\}$ ;
5:  $\#expDFS \leftarrow 0$ ;
6: while ( $not\_empty(OPEN\_A^*)$ ) and ( $\#expDFS < E$ ) and ( $not\ timeLimit\ reached$ ) do
7:    $n \leftarrow pop(OPEN)$ ;
8:   if  $f(n) < min(UB, UB_{DFS})$  then
9:     if  $isGoal(n)$  then
10:       $UB_{DFS} \leftarrow f(n)$ ;
11:      Update  $newSolution$  from  $n$ ;
12:       $\#expDFS \leftarrow 0$ ;
13:     end if
14:      $S \leftarrow SUC(n)$   $\{S\}$  sorted by non-decreasing  $f$ -values;
15:      $OPEN \leftarrow append(S, OPEN)$ ;
16:      $\#expDFS \leftarrow \#expDFS + 1$ ;
17:   end if
18: end while
19: if  $empty(OPEN)$  then
20:    $LB_{DFS} \leftarrow min(UB, UB_{DFS})$ ;
21: else
22:    $LB_{DFS} \leftarrow min\{f(q) \mid q \in OPEN\}$ ;
23: end if
24: return  $newSolution, LB_{DFS}, UB_{DFS}$ ;

```

4. A*DFS for $JSO(n, p)$ with total flow time minimisation

In this section, we set out the main components of the proposed algorithm, namely the search space where the algorithm looks for solutions, the heuristic estimation used to guide the search and also some global pruning rules which help to reduce the effective search space.

4.1 The search space: OG&T schedule generation scheme

We define a search space which is derived from the *OG&T* algorithm proposed by Sierra et al. (2013). 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 (1960) for the classic 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 , $1 \leq i \leq p$.

In order to select the candidate operations to be scheduled next, we start considering a naive strategy to establish an initial set of candidates which is subsequently restricted by means of some local pruning rules. 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

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

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

$$r_u = \max \left\{ r_{PJ_u} + p_{PJ_u}, r_v + p_v, \min_{1 \leq i \leq p} t_i \right\} \quad (2)$$

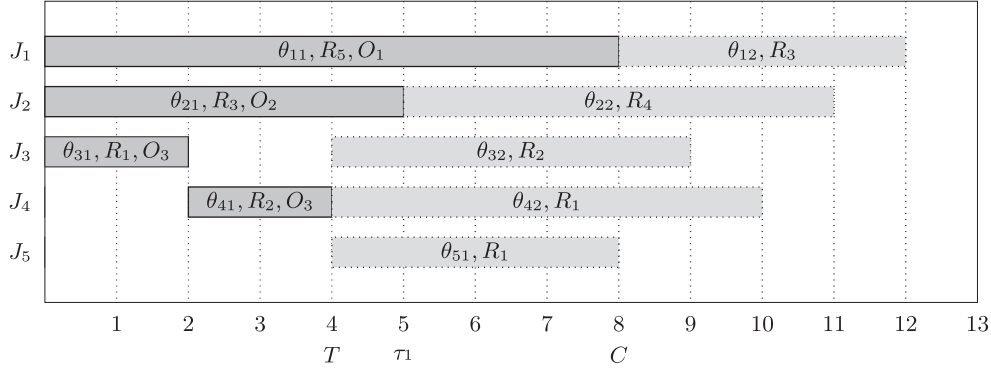


Figure 3. A partial schedule to a problem with 5 jobs, 5 machines and 3 operators. The scheduled operations are $\theta_{11}, \theta_{21}, \theta_{31}, \theta_{41}$.

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.

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

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.

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

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

$$A' = \{u \in A; r_u < r_{v^*} + p_{v^*}\} \quad (5)$$

Moreover, the set of choices can be further restricted in the following way. Let $\tau_0 < \dots < \tau_k$ 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 that $JSO(n, p)$ becomes $J||\Sigma C_i$, it is equivalent to the $G\&T$ algorithm. Sierra et al. (2013) give a full description of the $OG\&T$ algorithm together with a formal proof of its dominance property, i.e. the search space contains at least one optimal solution.

So, in accordance with the above, a state is a partial schedule. In the initial state, all the operations are unscheduled and the remaining states correspond to each one of the situations that may be generated by the algorithm $OG\&T$ after one of the operations of the set B is scheduled. Figure 3 shows a partial schedule that represents a feasible state to a problem with five jobs, five machines and three operators. In accordance with the $OG\&T$ algorithm, in this situation $A = \{\theta_{12}, \theta_{22}, \theta_{32}, \theta_{42}, \theta_{51}\}$, $A' = \{\theta_{22}, \theta_{32}, \theta_{42}, \theta_{51}\}$ and $B = \{\theta_{32}, \theta_{42}, \theta_{51}\}$ (if R_2 and R_1 are chosen from the interval $[\tau_1, C)$). Therefore, this state has three successors as a result of scheduling the operations θ_{32} , θ_{42} and θ_{51} , respectively. For a state n , $g(n)$ is the total flow time of the partial schedule, i.e. the summation of the completion times of the last operation scheduled in each job, and consequently the cost $c(n, n')$ from n to a successor n' is the variation of this value from n to n' . So, for the state n in Figure 3, $g(n) = 19$ and if n' is the result of scheduling θ_{32} from n , then $c(n, n') = 7$. The goals are those states having all the operations scheduled.

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, termed h_{PS} and h_{OP} , respectively and finally we take $h(s) = \max(h_{PS}(s), h_{OP}(s))$.

4.2.1 Heuristic h_{PS}

The first one, termed h_{PS} , is borrowed from Sierra and Varela (2010) where the problem $J||\sum C_i$ is considered: it relies on relaxing non-pre-emption 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 $h_{PS}(s) = f_{PS}(s) - g(s)$.

4.2.2 Heuristic h_{OP}

The second heuristic is obtained from relaxing constraints other than operator's. We start relaxing the heads and the capacity constraints of the machines for the unscheduled operations, with the only exception of the head of the first unscheduled operation in each job. 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 join into one only operation whose release time is the head of the first unscheduled operation of the job. This problem is denoted $(P, NC_{inc}|r_i|\sum C_i)$. A simplification of this problem where all release times are equal, denoted $(P, NC_{inc}||\sum C_i)$, can be optimally solved applying the SPT (Shortest Processing Time) rule (Adiri et al. 1989; Schmidt 2000). At the same time, the one machine-sequencing problem with release times (heads) and total flow time minimisation, denoted $(1|r_i|\sum C_i)$, is NP -hard (Graham et al. 1979) and its pre-emptive version $(1|r_i, pmtn|\sum C_i)$ can be solved in polynomial time by an extension of the SPT rule. However, the problem $(P, |r_i, pmtn|\sum C_i)$ is NP -complete as proved by Baptiste et al. (2007). Therefore, the problem $(P, NC_{inc}|r_i, pmtn|\sum C_i)$ is NP -complete as well.

From the results above, we decided relaxing the release times as well to obtain a polynomially solvable problem, $(P, NC_{inc}||\sum C_i)$. Algorithm 3 shows the calculation of heuristic h_{OP} for a state s .

Algorithm 3 Calculating the heuristic h_{OP} for a state s

Require: A state s .

Ensure: The heuristic estimation $h_{OP}(s)$.

Build a $(P, NC_{inc}||\sum C_i)$ instance \mathbf{P} relaxing the problem represented by the state s as it is indicated in the text;

while not all operations in \mathbf{P} are scheduled **do**

 Let m be the first machine (operator) available in the current partial schedule;

 Let t be the time at which m gets available;

 Select the unscheduled operation θ in \mathbf{P} with the shortest processing time;

 Schedule the operation θ at time t on the machine m ;

end while

return The total flow time of the built schedule for \mathbf{P} - $g(s)$;

4.3 Dominance relations

The effective search tree may be reduced by means of dominance relations among states similar to that exploited by Sierra and Varela (2010) for the classic job shop scheduling problem. This relation is defined in accordance with the formal definition given in Ibaraki (1977) for dominance relations that guarantee a single optimal solution. 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 J_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 $t_1 \leq \dots \leq t_p$ 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 $t'_1 \leq \dots \leq t'_p$, where $t'_i = \max(r_{u^*}, t_i)$. So, the availability of operators in state s is defined as the ordered vector $av(s) = (t'_1, \dots, t'_p)$. At the same time, 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 4.1 *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 . \square

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 4.1. 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 3 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.

In order to save memory, the proposed A*DFS algorithm only stores nodes in the CLOSED list when it is running in A* mode. In DFS mode, it only checks if the state to be expanded is dominated by a state in CLOSED.

5. Computational results

In this section, we report the results from an experimental study carried out to evaluate the components of A*DFS and to compare it with other methods. We have experimented across a number of instances adapted from the classic JSP considering different numbers of operators. The target machine was Intel Xeon (2,26 GHz), 24 GB RAM. and all the algorithms were implemented in C++. In these experiments, we have set A*DFS parameters $N = 100$ and E at twice the distance from n to the leaves. The algorithms were evaluated at two time limits, 300 s and 3600 s. Also, A*, DFS and A*DFS have been given a memory limit of 4 GB.

5.1 Evaluating the heuristics and the pruning method

We start with some experiments to analyse the behavior of the heuristic estimations h_{PS} and h_{OP} separately and in combination and the efficiency of the pruning method by dominance rules. To do that, we chose a small instance such as FT06, of size $n \times m = 6 \times 6$, which can be solved to optimality for almost any combination of number of operators, heuristic and pruning option. Figure 4 shows the number of expanded states (in logarithmic scale) required to certify the optimal solution in each case. We established a time limit of 3600 s. Even though most of the instances were solved in much less time, two of them were not solved: the instances with 1 and 2 operators using h_{PS} and no pruning. Moreover, these instances were not solved by 24 hours after expanding about 350 million states. It is remarkable that these instances get solved by 36s and 487s, respectively, with the same heuristic and pruning. In these two cases, the number of expanded states after 3600 s was considered.

We can observe that h_{OP} is the best option for small values of p while h_{PS} is the best one for large values, independently of the pruning option. Clearly, the combined heuristic is the best option overall. At the same time, it is clear that the pruning method is quite effective. In almost all the cases, the number of expanded states is much lower with this option. In particular, for the instances with 3 and 4 operators, which seem to be the most difficult to solve, and the combined heuristic, the number

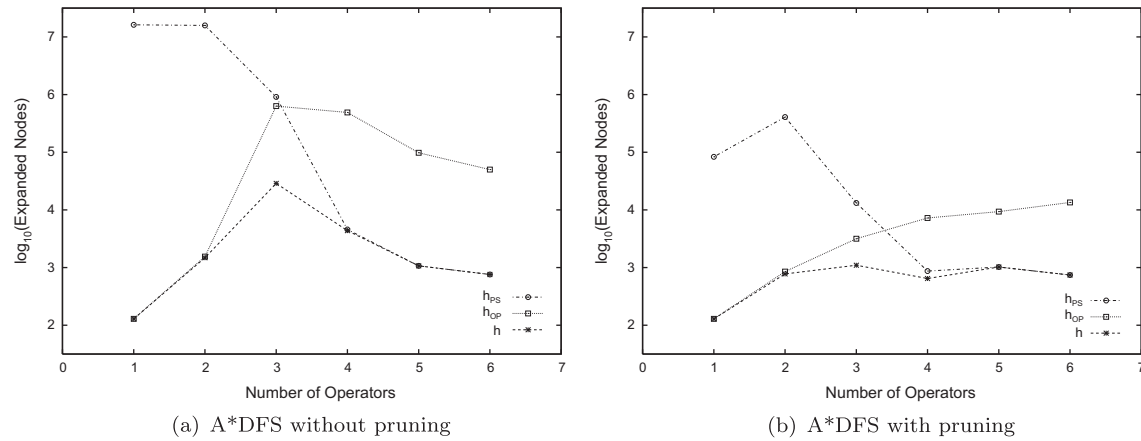


Figure 4. Summary of results from A*DFS on FT06 (6×6) instance for different combinations of heuristic, number of operators and pruning option. The time limit is 3600 s.

of expanded states decreases in one order of magnitude thanks to the pruning method. From these results, we considered the combined heuristic and the pruning method in the remaining experiments.

5.2 Comparison with other algorithms

As, to our knowledge, there is no other approach to the $JSO(n, p)$ problem with total flow time minimisation, we opted to compare A*DFS with its two component algorithms A* and DFS,³ and also with an implementation on a current solver specialised in scheduling as it is IBM ILOG CPLEX CP Optimiser (IBM 2009). When DFS is used alone, we exploit the pruning rule in the same way as in A*, i.e. storing the expanded nodes in CLOSED as long as the memory limit is not reached. In the CP implementation, $JSO(n, p)$ is modeled like a classic job shop scheduling problem where the p operators are naturally modeled as a nonrenewable cumulative resource of capacity p . 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).

In the experiments, we considered a large benchmark defined from the well-known LA01-40 instances (Beasley 1990) with $n \times m \in \{10 \times 5, 15 \times 5, 20 \times 5, 10 \times 10, 15 \times 10, 20 \times 10, 30 \times 10, 15 \times 15\}$; each subset having five instances. For each one of these instances, the number of operators p varies from 1 to $\min(n, m)$, so we have 350 instances in all. Every instance was solved with each method taking two different time limits: 300 s and 3600 s respectively. The lower bound and the solution reached (upper bound) was recorded in each run.

5.2.1 Upper bounds

The errors in percentage terms (%Error) of each solution w.r.t. the best lower bound obtained in our experiments are summarised in Figures 5 and 6 and Tables 1 and 2. Figures 5 and 6 show the results from the four algorithms for each group of instances with the same size at 300 s and 3600 s, respectively, averaged for all instances with the same number of operators. The first remarkable result of these experiments is that for all combinations of n , m and p , A*DFS is always better than both A* and DFS alone independently of the time limit. Also, A*DFS is better than CP in most of the cases.

Tables 1 and 2 summarise the results for each group of instances with the same size and time limits 300 s and 3600 s, respectively, and show the number of instances for which the algorithms are able to certify the optimality of the solution (#Sol.). As we can observe, the results of A* are the same at both time limits as it consumes the whole memory before 300 s, while the results from DFS are slightly better at 3600 s than they are at 300 s. Nevertheless, A*DFS is able to improve substantially the quality of the solutions and also to certify the optimality of more of them when it is given more time. CP is the algorithm that layouts the largest difference in the quality of solutions from 300 s to 3600 s, but even at 3600 s CP has larger average error than A*DFS at 300 s. Moreover, even at 3600 s CP is unable to certify the optimality of any solution.

In order to enhance the conclusions of the experimental comparison between A*DFS and CP, we have conducted some statistical analysis following García et al. (2010). Particularly, we have used paired Wilcoxon tests samples. We have used as alternative hypothesis that the errors from a method M1 at a time T1 are smaller than those from a method M2 at time T2.

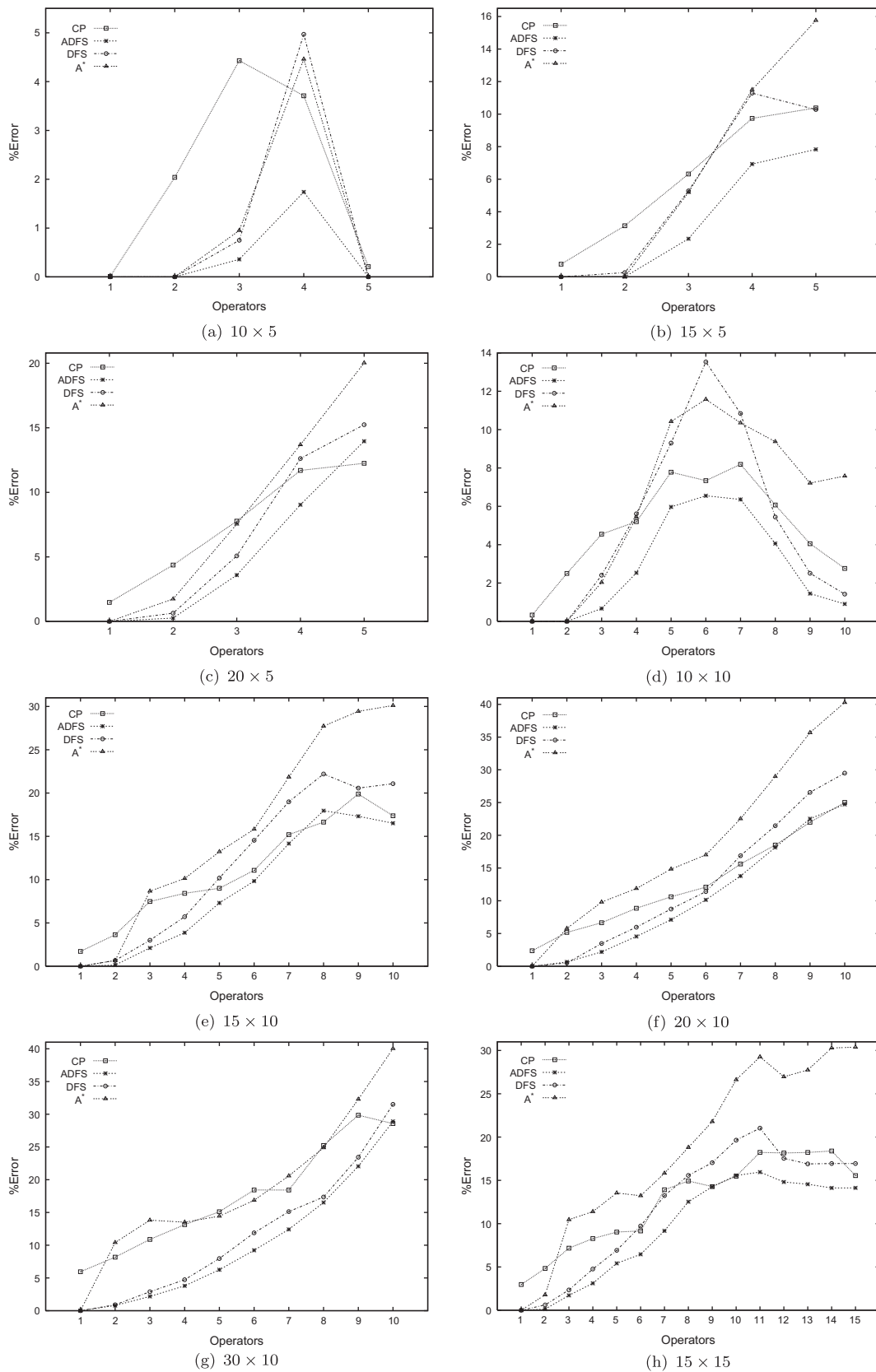


Figure 5. Errors in percentage from CP, A*DFS, DFS and A* across the LA set averaged for instances with the same size and number of operators. The time limit is 300 s.

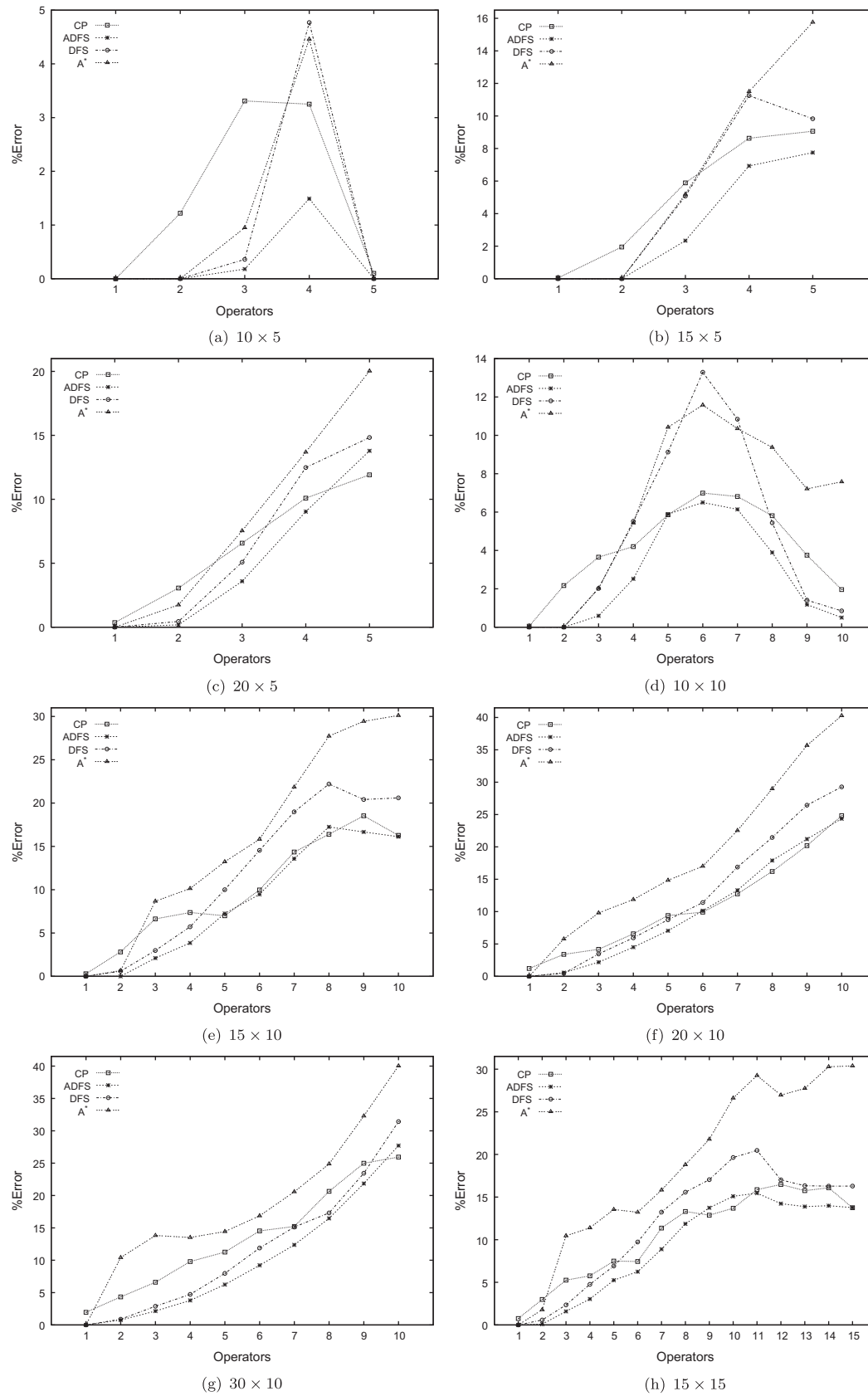


Figure 6. Errors in percentage from CP, A*DFS, DFS and A* across the LA set averaged for instances with the same size and number of operators. The time limit is 3600 s.

Table 1. Errors in percentage of the solutions reached by the four algorithms averaged for each group of instances with the same size ($n \times m$). The time limit is 300 s.

Size	CP		A*DFS		DFS		A*	
	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.
10×5	0	2,08	16	0,42	15	1,14	17	1,08
15×5	0	6,07	10	3,42	6	5,42	10	6,49
20×5	0	7,51	5	5,37	5	6,71	5	8,60
10×10	0	4,88	10	2,85	10	5,11	10	6,40
15×10	0	11,04	5	8,92	5	11,70	7	15,76
20×10	0	12,69	5	10,38	5	12,45	5	18,68
30×10	0	17,38	5	10,21	5	11,58	5	18,68
15×15	0	12,57	6	9,46	5	11,95	7	18,53
Avg. %Err.		9,28		6,62		8,26		11,78
Sum. #Sol.	0		62		56		66	

Table 2. Errors in percentage of the solutions reached by the four algorithms averaged for each group of instances with the same size ($n \times m$). The time limit is 3600 s.

Size	CP		A*DFS		DFS		A*	
	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.	#Sol.	%Err.
10×5	0	1,58	21	0,33	16	1,03	17	1,08
15×5	0	5,12	10	3,40	10	5,23	10	6,49
20×5	0	6,40	6	5,32	6	6,57	5	8,60
10×10	0	4,12	18	2,72	11	4,85	10	6,40
15×10	0	9,96	10	8,63	5	11,60	7	15,76
20×10	0	10,86	5	10,12	5	12,41	5	18,68
30×10	0	13,53	5	10,05	5	11,57	5	18,68
15×15	0	10,59	6	9,14	5	11,75	7	18,53
Avg. %Err.		7,77		6,22		8,13		11,78
Sum. #Sol.	0		81		63		66	

Table 3. p -Values from Wilcoxon paired tests of the results from A*DFS and CP by 300 s (5 min) and 3600 s (1 h) 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 at a time T1 are smaller than those from M2 at time T2.

Size	M1 (T1)	A*DFS (5 min)	A*DFS (1 h)	A*DFS (5 min)
	M2 (T2)	CP (5 min)	CP (1 h)	CP (1 h)
10×5		1,07E-04	1,61E-04	1,60E-04
15×5		9,84E-07	1,87E-04	2,09E-04
20×5		5,14E-04	1,47E-02	1,79E-02
10×10		1,21E-08	4,94E-07	2,11E-06
15×10		2,85E-06	1,87E-04	4,20E-03
20×10		2,06E-07	1,05E-02	3,95E-02
30×10		7,13E-10	1,77E-08	1,41E-07
15×15		1,34E-12	1,47E-07	1,13E-05

So, we have analysed several combinations of methods (A*DFS and CP) and time limits (300 s and 3600 s). The results of these tests are summarised in Table 3 where the p -values are indicated when each group of instances with the same size is taken as population sample. As we can observe, the null hypothesis is rejected at a high level of significance in all cases. So, these analysis confirm that A*DFS outperforms CP in this benchmark at both time limits. Furthermore, A*DFS at 300 s is better than CP at 3600 s.

Table 4. Errors in percentage of the lower bounds reached by A*, DFS and A*DFS by 300 s and 3600 s. The values are averaged for each group of instances with the same size ($n \times m$). A* always finishes before 300 s with either an optimal solution or the memory getting exhausted.

Size	A*	300 s		3600 s	
		A*DFS	DFS	A*DFS	DFS
10 \times 5	0,15	0,17	2,43	0,00	2,28
15 \times 5	0,07	0,07	1,89	0,00	1,79
20 \times 5	0,02	0,06	0,95	0,00	0,93
10 \times 10	0,48	0,83	4,72	0,00	4,62
15 \times 10	0,06	0,22	1,66	0,00	1,66
20 \times 10	0,06	0,11	0,71	0,00	0,71
30 \times 10	0,01	0,04	0,22	0,00	0,22
15 \times 15	0,03	0,25	1,19	0,00	1,19
Avg. Err.	0,11	0,22	1,72	0,00	1,68

5.2.2 Lower bounds

Table 4 reports the results of the lower bounds reached by A*, DFS and A*DFS at both time limits, 300 s and 3600 s. The values are errors in percentage terms, w.r.t. the best lower bounds reached in all the experiments, averaged for each group of instances with the same size. As we can observe, at 300 s it is A* the algorithm that computes the best lower bounds in average. However, A*DFS at 3600 s is the best one as it returns the best lower bound for every instance. So, we can conclude that not only A*DFS outperforms its two components A* and DFS in reaching upper bounds, but it also outperforms both A* and DFS in reaching lower bounds in the long term.⁴

6. Conclusions

We have seen that by combining different search strategies as best-first search and depth-first search, it is possible to obtain a hybrid search algorithm that outperforms both strategies when each of them works separately. More concretely, we have proposed a new hybrid search strategy, termed A*DFS, that combines best-first and depth-first search in an original way. Firstly, it yields depth-first searches limited by a number of expansions from some of the states to be expanded by A*, hence diversifying the search and sampling different promising regions of the search space. Then, when a memory limit is reached, A*DFS yields exhaustive depth-first searches from the nodes generated but not yet expanded in the previous phase, which allows the algorithm to improve both lower and upper bounds without requiring additional memory resources.

We have applied this algorithm to solve the job shop scheduling problem with operators and total flow time minimisation, which is very hard to solve and has interesting practical applications. To do so, we have exploited the schedule generation scheme proposed in (Sierra et al. 2013), termed *OG&T*, defining a reduced search space. This algorithm also uses effectively some global pruning rules which rely on dominance relations among states and two heuristic estimations for search guidance and pruning.

The results from a comprehensive experimental study over a large benchmark set defined from well-known instances of the job shop scheduling problem shows that the hybrid algorithm is able to compute high quality solutions and lower bounds, and it clearly outperforms its two components: best-first and depth-first search. Furthermore, a comparison with IBM ILOG CPLEX CP Optimiser, a commercial solver specialised in scheduling problems, shows that the proposed algorithm is able to compute better solutions in 300 s than CP Optimiser is in 3600 s.

As future work, we plan to improve A*DFS in various ways, for example by introducing restarts and randomisation (Gomes et al. 1998) or other variants of depth-first such a limited discrepancy search (LDS) (Harvey and Ginsberg 1995) or the recent intensified iterative deepening A* (Mencía et al. 2013), in order to get better upper bounds.

Finally, we plan to refine the heuristics and the global pruning rules for the same problem and to apply a similar approach to other scheduling problems; in particular to scheduling problems with operators having different skills, as the one faced in Guyon et al. (2012).

Acknowledgements

We are grateful to the anonymous reviewers for their useful comments and suggestions. This research has been supported by the Spanish Government under project FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP09105.

Notes

1. Without loss of generality, we consider minimisation problems in this section.
2. Some examples are puzzles or planning problems.
3. Some preliminary results from A* and DFS have been discussed in Sierra et al. (2011) and Mencía et al. (2012), respectively.
4. Detailed results from all these experiments, including the best lower and upper bounds for each instance are provided in the supplementary material (www.di.uniovi.es/iscop, link Repository/Detailed Results from Papers).

References

- Adiri, I., J. Bruno, E. Frostig, E. and A. H. G. Rinnooy Kan. 1989. "Single Machine Flow-Time Scheduling with a Single Breakdown." *Acta Informatica* 26 : 679–696.
- Agnetis, A., M. Flamini, G. Nicosia and A. Pacifici. 2011. "A Job-shop Problem with One Additional Resource Type." *Journal of Scheduling* 14 (3): 225–237.
- Artigues, C., M. Gendreau, and L. Rousseau, and A. Vergnaud. 2009. "Solving an Integrated Employee Timetabling and Job-shop Scheduling Problem Via Hybrid Branch-and-bound." *Computers & Operations Research* 36 (8): 2330–2340.
- Baptiste, P., P. Brucker, M. Chrobak and C. Dürr. 2007. "The Complexity of Mean Flow Time Scheduling Problems with Release Times." *Journal of Scheduling* 10 : 139–146.
- Beasley, J. E. 1990. "OR-Library: Distributing Test Problems by Electronic Mail." *Journal of the Operational Research Society* 41 (11): 1069–1072.
- Brucker, P., and S. Knust. 2006. *Complex Scheduling*. New York: Springer-Verlag.
- Driebel, R., and L. Monch. 2012. "An Integrated Scheduling and Material-handling Approach for Complex Job Shops: a computational study." *International Journal of Production Research* 50 (20): 5966–5985.
- Framinan, J., and R. Leisten. 2003. "An Efficient Constructive Heuristic for Flowtime Minimisation in Permutation Flow Shops." *Omega, International Journal of Management Science* 31 (4): 311–317.
- Gacias, B., C. Artigues, and P. Lopeza. 2010. "Parallel Machine Scheduling with Precedence Constraints and Setup Times." *Computers and Operations Research* 37 : 2141–2151.
- Ganggang, N., S. Sun, P. Lafon, Y. Zhang and J. Wang. 2012. "Two Decompositions for the Bicriteria Job-shop Scheduling Problem with Discretely Controllable Processing Times." *International Journal of Production Research* 50 (24): 7415–7427.
- García, S., A. Fernández, J. Luengo and F. Herrera. 2010. "Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power." *Information Sciences* 180 : 2044–2064.
- Giffler, B., and G. L. Thompson. 1960. "Algorithms for Solving Production Scheduling Problems." *Operations Research* 8 : 487–503.
- Glover, F. and M. Laguna. 1993. "Tabu Search." In *Modern Heuristic Techniques for Combinatorial Problems*, edited by C. Reeves, 70–150. Oxford: Blackwell.
- Gomes, C. P., B. Selman, and H. A. Kautz. 1998. "Boosting Combinatorial Search Through Randomization." In *AAAI/IAAI*, 431–437.
- González, M. A., C. R. Vela, I. González-Rodríguez and R. Varela. 2012. "An Efficient Hybrid Evolutionary Algorithm for Scheduling with Setup times and Weighted Tardiness Minimization." *Soft Computing* 16 (12): 2097–2113.
- González, M. A., C. R. Vela, M. R. Sierra and R. Varela. 2010. "Tabu Search and Genetic Algorithm for Scheduling with Total Flow Time Minimization." In *COPLAS 2010: ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 33–41.
- Graham, R. L., E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan. 1979. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey." *Annals of Discrete Mathematics* 5 : 287–326.
- Guyon, O., P. Lemaire, E. Pinson and D. Rivreau. 2012. "Solving an Integrated Job-shop Problem with Human Resource Constraints." *Annals of Operations Research*, doi:10.1007/s10479-012-1132-3.
- Hart, P., N. Nilsson, and B. Raphael. 1968. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." *IEEE Transactions on Systems Science and Cybernetics* 4 (2): 100–107.
- Harvey, W. D., and M. L. Ginsberg. 1995. "Limited Discrepancy Search." *Proceedings of IJCAI* 1995 1 : 607–615.
- Ibaraki, T. 1977. "The Power of Dominance Relations in Branch-and-Bound Algorithms." *Journal of the Association for Computing Machinery* 24 (2): 264–279.
- IBM, 2009. Modeling with IBM ILOG CP Optimizer – Practical Scheduling Examples. <ftp://public.dhe.ibm.com/common/ssi/ecm/en/WSW14076usen/WSW14076USEN.PDF>.
- Laborie, P., 2009. "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)*, 148–162.

- Lin, S. W., C. Y. Huang, C. C. Lu and K. C. Ying. 2012. "Minimizing Total Flow Time in Permutation Flowshop Environment." *International Journal of Innovative Computing, Information and Control* 8 (10A): 6599–6612.
- Liu, C. H., L. S. Chen, and P. S. Lin. 2012. "Lot Streaming Multiple Jobs with Values Exponentially Deteriorating over Time in a Job-shop Environment." *International Journal of Production Research*. 51 (1): 202–214.
- Meeran, S., and M. Morshed. 2012. "A Hybrid Genetic Tabu Search Algorithm for Solving Job Shop Scheduling Problems: A Case Study." *Journal of Intelligent Manufacturing* 23 (4): 1063–1078.
- Mencía, C., M. R. Sierra, M. A. Salido, J. Escamilla and R. Varela. 2012. "Combining Global Pruning Rules with Depth-first Search for the Job Shop Scheduling Problem with Operators." In *Proceedings of the 19th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*.
- Mencía, C., Sierra, M. R., and Varela, R. 2010. "Partially Informed Depth First Search for the Job Shop Problem." In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 113–120. Toronto: AAAI Press.
- Mencía, C., M. R. Sierra, and R. Varela. 2012. "Depth-first Heuristic Search for the Job Shop Scheduling Problem." *Annals of Operations Research* 206: 265–296.
- Mencía, C., M. R. Sierra, and R. Varela. 2013. "Intensified Iterative Deepening A* with Application to Job Shop Scheduling." *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-012-0726-6.
- Nilsson, N. 1980. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga.
- Pearl, J. 1984. *Heuristics: Intelligent Search strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley.
- Rabiee, M., M. Zandieh, and P. Ramezani. 2012. "Bi-objective Partial Flexible Job Shop Scheduling Problem: NSGA-II, NPGA, MOGA and PAES Approaches." *International Journal of Production Research* 50 (24): 7327–7342.
- Ruiz-Torres, A. J., J. H. Ablanedo-Rosas, and L. D. Otero. 2012. "Scheduling with Multiple Tasks Per Job – The Case of Quality Control Laboratories in the Pharmaceutical Industry." *International Journal of Production Research* 50 (3): 691–705.
- Schmidt, G. 2000. "Scheduling with Limited Machine Availability." *European Journal of Operational Research* 5: 1–15.
- Sierra, M. R., C. Mencía, and R. Varela. 2011. "Advances in Artificial Intelligence." *Lecture Notes in Computer Science* 7023: 193–202.
- Sierra, M. R., C. Mencía, and R. Varela. 2013. "New Schedule Generation Schemes for the Job-shop Problem with Operators." *Journal of Intelligence Manufacturing*. doi: 10.1007/s10845-013-0810-6.
- Sierra, M. R., and R. Varela. 2010. "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.
- Stern, R., T. Kulberis, A. Felner, and R. Holte. 2010. "Using Lookaheads with Optimal Best-First Search." In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 185–190. Atlanta, Georgia, July 11–15.
- Talbi, E. 2009. "Metaheuristics: From Design to Implementation." In *Wiley Series on Parallel and Distributed Computing*. New York: JohnWiley & Sons.
- Vela, C. R., R. Varela, and M. A. González. 2010. "Local Search and Genetic Algorithm for the Job Shop Scheduling Problem with Sequence Dependent Setup Times." *Journal of Heuristics* 16: 139–165.
- Zouba, M., P. Baptiste, and D. Rebaine. 2009. "Scheduling Identical Parallel Machines and Operators within a Period Based Changing Mode." *Computers and Operations Research* 36 (12): 3231–3239.