Improving heuristic estimations with constraint propagation in searching for optimal schedules [⋆]

Carlos Mencía, María R. Sierra and Ramiro Varela

Department of Computing.
University of Oviedo. Campus of Viesques, 33271 Gijón, Spain
Tel. +34-8-5182508. FAX +34-8-5182125.
{uo156612, sierramaria, ramiro}@uniovi.es
http://www.aic.uniovi.es/Tc

Abstract. We face the Job Shop Scheduling Problem by means of branch and bound and A^* search. Our main contribution is a new method, based on constraint propagation rules, that allows improving the heuristic estimations. We report results from an experimental study across conventional instances with different sizes showing that A^* takes profit from the improved estimations. Both algorithms can reach optimal solutions for medium size instances and, in this case, the branch and bound algorithm is better than A^*. However, for very large instances that remain unsolved in both cases, A^* returns much better lower bounds due to the improved estimation.

Keywords Job Shop Scheduling, Heuristic Search, A^* algorithm, Branch and Bound Constraint propagation

1 Introduction

State space search is a classical technique in the field of artificial intelligence. It often allows to obtain optimal solutions to combinatorial problems up to a moderate size. Usually, the key for success consists in devising a reduced search space, accurate heuristic estimations of lower and upper bounds of optimal solutions and powerful constraint propagation rules to reduce the effective search space. Search algorithms require large amounts of computational resources, i.e. time and space. So, when the main memory of the target machine is scarce, only branch and bound ($B\&B$) algorithms are actually efficient. However, the current technology offers low cost machines with up to 8 or 16 GiB of memory, so as other approaches such as best first search are also efficient for at least moderate sized problems.

In this paper we consider the Job Shop Scheduling Problem (JSSP) and two search algorithms to solve it: the branch and bound algorithm proposed by P. Brucker et al. in [Brucker et al., 1994] and [Brucker, 2004], and an implementation of the Nilsson's A^* algorithm [Nilsson, 1980] designed from Brucker's algorithm. Our main contribution

[⋆] This work has been supported by the Spanish Ministry of Science and Education under research project MEC-FEDER TIN2007-67466-C02-01 and by the Principality of Asturias under grant FICYT-BP09105.

is a method to improve heuristic estimations which is based on some constraint propagation rules. The results of our experimental study show that both algorithms can solve instances with 10 jobs and 10 machines and that A^* reaches better lower bounds for larger instances that remain unsolved with both algorithms, while the branch and bound algorithm computes the best upper bounds.

The remaining of the paper is organized as follows. In section 2 the JSSP is formulated. Section 3 outlines the main characteristics of Brucker's algorithm. In section 4, we describe the main components of the A^* algorithm which is based on ideas taken from Brucker's algorithm. In section 5 we present the method to improve the heuristic estimations. Section 6 shows the results of the experimental study. Finally, in section 7 we summarize the main conclusions and propose some ideas for future research.

2 Problem formulation

The Job Shop Scheduling Problem (JSSP) requires scheduling a set of N jobs $\{J_1, \ldots, J_N\}$ on a set of M resources or machines $\{R_1, \ldots, R_M\}$. Each job J_i consists of a set of tasks or operations $\{\theta_{i1}, \ldots, \theta_{iM}\}$ to be sequentially scheduled. Each task θ_{il} has a single resource requirement $R_{\theta il}$, a fixed duration $p_{\theta il}$ and a start time $st_{\theta il}$ to be determined. The JSSP has three constraints: precedence, capacity and non-preemption. Precedence constraints translate into linear inequalities of the type: $st_{\theta il} + p_{\theta il} \leq st_{\theta i(l+1)}$. Capacity constraints translate into disjunctive constraints of the form: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, if $R_v = R_w$. Non-preemption requires assigning machines to operations without interruption during their whole processing time. The objective is to come up with a feasible schedule such that the completion time, i.e. the *makespan*, is minimized. This problem is denoted as $J||C_{max}$ in the $\alpha|\beta|\gamma$ notation.

In the sequel, a problem instance will be represented by a directed graph $G = (V, A \cup E)$. Each node in the set V represents an actual operation, with the exception of the dummy nodes *start* and *end*, which represent operations with processing time 0. The arcs of A are called *conjunctive arcs* and represent precedence constraints and the arcs of E are called *disjunctive arcs* and represent capacity constraints. E is partitioned into subsets E_i with $E = \cup_{\{i=1,\ldots,M\}} E_i$. E_i includes an *arc* (v, w) for each pair of operations requiring R_i. The arcs are weighted with the processing time of the operation at the source node. Node *start* is connected to the first operation of each job and the last operation of each job is connected to node *end*.

A feasible schedule S is represented by an acyclic subgraph G_S of G, $G_S = (V, A \cup H)$, where $H = \cup_{i=1,\ldots,M} H_i$, H_i being a processing ordering for the operations requiring R_i. The makespan is the cost of a *critical path* and it is denoted as $L(S)$. A critical path is a longest path from node *start* to node *end*. A critical path contains a set of *critical blocks*. A critical block is a maximal sequence, with length at least two, of consecutive operations in a critical path requiring the same machine. Figure 1 shows a solution graph for an instance with 3 jobs and 3 machines.

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v, i.e. it is the value of st_v. The *tail* q_v is defined so as the value $q_v + p_v$ is the cost of the longest path from v to *end*. Hence, $r_v + p_v + q_v$ is the makespan if v is in a critical path, otherwise, it is a lower bound. PM_v and SM_v denote the predecessor and successor of v respectively on the machine sequence and PJ_v and SJ_v denote the predecessor and successor operations of v respectively on the job sequence.

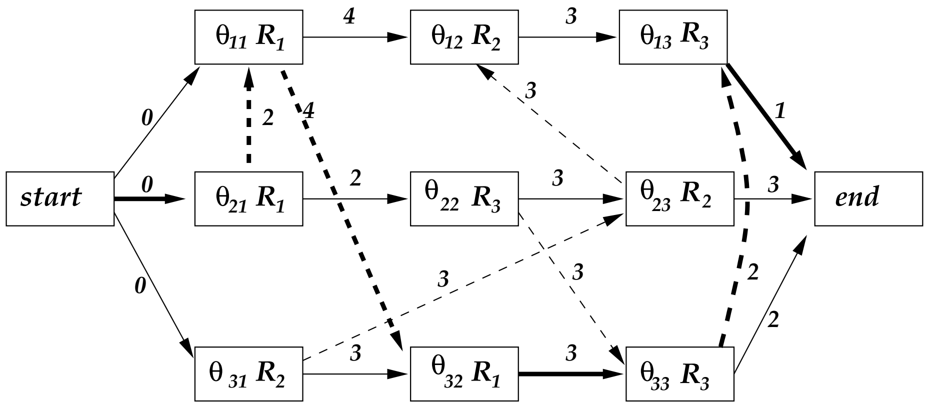

Fig. 1. A feasible schedule to a problem with 3 jobs and 3 machines. Bold face arcs show a critical path whose length (makespan) is 12.

A partial schedule is given by a subgraph of G where some of the disjunctive arcs are not fixed yet. In such a schedule, heads and tails can be estimated as

$$r_v = \max\{ \max_{J \subseteq P(v)} \{\min_{j \in J} r_j + \sum_{j \in J} p_j\}, r_{PJ_v} + p_{PJ_v}\} \tag{1}$$

$$q_v = \max\{ \max_{J \subseteq S(v)} \{\sum_{j \in J} p_j + \min_{j \in J} q_j\}, p_{SJ_v} + q_{SJ_v}\} \tag{2}$$

with $r_{start} = q_{end} = 0$ and where $P(v)$ denotes the disjunctive predecessors of v, i.e. operations requiring machine R_v which are scheduled before than v. Analogously, $S(v)$ denotes the disjunctive successors of v. Hence, the value $r_v + p_v + q_v$ is a lower bound of the best schedule that can be reached from the partial schedule.

3 Brucker's algorithm

The best exact algorithm proposed so far to cope with the $J||C_{max}$ problem is undoubtedly the branch and bound algorithm due to P. Brucker et al. ([Brucker et al., 1994]). This algorithm starts from the constraint graph for a given problem instance and proceeds to fix disjunctive arcs subsequently. The key feature of the algorithm is a smart branching schema that relies on fixing arcs either in the same or in the opposite direction as they appear in a critical block of a feasible solution. Moreover, the algorithm exploits powerful methods to obtain accurate lower and upper bounds. Lower bound calculation is based on preemptive one machine sequencing problem relaxations. The optimal solution to the simplified problem is given by the so-called Jackson's Preemptive Schedule (JPS), which is obtained in polynomial time by the algorithm proposed in [Carlier, 1982]. Upper bounds are obtained by means of the greedy $G\&T$ algorithm proposed in [Giffler and Thomson, 1960]. Finally, Brucker's algorithm exploits a constraint propagation method termed immediate selection [Carlier and Pinson, 1989]. This method allows fixing additional disjuntive arcs so as the effective search tree is dramatically reduced. Brucker's algorithm can easily solve almost any instance up to 10 jobs and 10 machines as well as many larger instances. In fact, the famous set of instances selected by D. Applegate and W. Cook in [Applegate and Cook, 1991] are considered very hard to solve due to the fact that they are not solved by this algorithm (with only one exception, the FT10 instance).

4 A^* algorithm for the JSSP

We start reviewing the A^* algorithm, then we describe the state space, the initial heuristic estimation, the way to obtain upper bounds, and the immediate selection procedure that allows reducing the effective search space. As we have commented above, all these components are adapted from Brucker's algorithm.

4.1 A^* algorithm

For best-first search we have chosen the Nilsson's A^* algorithm [Nilsson, 1980]. A^* starts from an initial state s, a set of goal nodes and a transition operator SUC such that for each node n of the search space, $SUC(n)$ returns the set of successor states of n. Each transition from n to n' has a non-negative cost $c(n,n')$. The algorithm searches for a path from s to the goals. The set of candidate nodes to be expanded are maintained in the $OPEN$ list. The next node to be expanded is that with the lowest value of the evaluation function f, defined as $f(n) = g(n) + h(n)$; where $g(n)$ is the minimal cost known so far from s to n and $h(n)$ is a heuristic estimation of the minimal distance from n to the nearest goal. If the heuristic function underestimates the actual minimal cost, $h^*(n)$, from n to the goals, i.e. $h(n) \leq h^*(n)$, for every node n, the algorithm is admissible, i.e. it returns an optimal solution. The heuristic function $h(n)$ represents knowledge about the problem domain, therefore as long as h approximates h^* the algorithm is more and more efficient as it needs to expand a lower number of states to reach an optimal solution.

4.2 The search space

A search state n is given by a partial solution graph $G_n = (V, A \cup FD_n)$, where FD_n denotes the fixed disjunctive arcs in n. In the initial state $FD_n = \oslash$. Heads and tails in a state n are calculated by expressions (1) and (2). The cost of the best path from the initial state to a state n is given by the largest cost path between nodes *start* and *end* in G_n; with only one exception for the initial state whose value is 0.

4.3 Heuristic estimation

The heuristic estimation is based on problem splitting and constraint relaxations. Let $\mathbf{O}$ be the set of operations requiring the machine m. Scheduling operations in $\mathbf{O}$ accordingly to their heads and tails in a state n so as the value $max_{v \in \mathbf{O}}(st_v + p_v + q_v)$ is minimized is known as the One Machine Sequencing Problem (OMSP). The optimal solution to this problem for a state n is clearly a lower bound of $f^*(n) = g^*(n) + h^*(n)$, where $g^*(n)$ is the optimal cost from *start* to n. However, the OMS problem is still NP-hard. So, a new relaxation is required in order to obtain a polynomially solvable problem. To do that, it is common relaxing the non-preemption constraint. An optimal

Algorithm 1 UB(state n). Calculates a heuristic solution S from a state n. O denotes the set of all operations and SC the set of scheduled operations

```
0. $SC = \{start\}$;
while ($SC \neq O$) do
  1. $A = \{v \in O \setminus SC;\ P(v) \cup \{PJ_v\} \subset SC\}$;
  2. $v^* = \arg\min\{r_u + p_u;\ u \in A\}$;
  3. $B = \{v \in A;\ R_v = R_{v^*}$ and $r_v < r_{v^*} + p_{v^*}\}$;
  4. $C = \{v \in O \setminus SC;\ R_v = R_{v^*}\}$;
  5. $w^* = \arg\min\{$makespan of JPS$(C \setminus \{w\})$ after schedule $w$; $w \in B\}$;
  6. Schedule $w^*$ in $S$ at a time $r_{w^*}$;
  7. Add $w^*$ to $SC$ and update heads and tails of operations not in $SC$;
end while
8. return $S$ and its makespan;
```

solution to the preemptive OMS problem is given by the Jackson's Preemptive Schedule (JPS) [Carlier and Pinson, 1989,Carlier and Pinson, 1994]. The JPS is calculated by the following algorithm: at any time t given by a head or the completion of an operation, from the minimum r_v until all operations are completely scheduled, schedule the ready operation with the largest tail on machine m. Carlier and Pinson proved that calculating the JPS has a complexity of $O(k \times \log k)$, where k is the number of operations. Finally, $f(n)$ is taken as the largest JPS over all machines. So, $h(n) = f(n) - g(n)$.

4.4 Upper bounds

As Brucker's algorithm does, we introduce upper bound calculations in the A^* counterpart. To do that, a variant of the well-known *G&T* algorithm proposed in [Giffler and Thomson, 1960] is used. The algorithm is issued from each expanded node so as it builds a schedule that includes all disjunctive arcs fixed in that state. Here it is important to remark that the disjunctive arcs fixed to obtain the schedule do not remain fixed in that node. *G&T* is a greedy algorithm that produces a schedule in a number of $N * M$ steps. Algorithm 1 shows the *G&T* algorithm adapted to obtain upper bounds from a search state n. Remember that $P(v)$ denotes the disjunctive predecessors of operation v in state n. In each iteration, the algorithm considers the set A comprising all operations v that can be scheduled next, i.e. all operations such that $P(v)$ and PJ_v are already scheduled (initially only operation *start* is scheduled). The operation v^* in A with the earliest completion time if it is scheduled next is calculated, and a new set B is obtained with all operations v in A requiring the same machine as v^* that can start at a time lower than the completion time of v^*. Any of the operations in B can be scheduled next and the selected one is w^* if it produces the least cost JPS for the remaining unscheduled operations on the same machine. Finally, the algorithm returns the built schedule S and the value of its makespan UB.

The algorithm maintains the best upper bound computed so far and finishes when $f(n) \geq UB$ for a node n selected to be expanded.

4.5 Expansion mechanism

The expansion mechanism is based on the following theorem [Brucker et al., 1994].

Theorem 1. *Let S and S' be two schedules. If $L(S') < L(S)$, then one of the two following conditions holds:*

1. *at least one operation v in a critical block B in G_S, different from the first operation of B, is processed in S' before all operations of B.*
2. *at least one operation v in a critical block B in G_S, different from the last operation of B, is processed in S' after all operations of B.*

Now, let us consider a feasible schedule S being compatible with the disjunctive arcs fixed in state n. Of course, S might be the schedule calculated by Algorithm 1. The solution graph G_S has a critical path with critical blocks $B_1, \dots, B_k$. For block $B_j = (u^j_1, \dots, u^j_{mj})$ the sets of operations

$$E^B_j = B_j \setminus \{u^j_1\} \text{ and } E^A_j = B_j \setminus \{u^j_{mj}\}$$

are called the *before-candidates* and *after-candidates* respectively. For each before-candidate (after-candidate) a successor s of n is generated by moving the candidate before (after) the corresponding block. An operation $l \in E^B_j$ is moved before B_j by fixing the arcs $\{l \to i; i \in B_j \setminus \{l\}\}$. Similarly, $l \in E^A_j$ is moved after B_j by fixing the arcs $\{i \to l; i \in B_j \setminus \{l\}\}$.

This expansion strategy is complete as it guaranties that at least one optimal solution is contained in the search graph. However, this strategy can be improved by fixing additional arcs so as the search space is a complete tree. Let us consider a permutation $(E_1, \dots, E_{2k})$ of all sets E^B_j and E^A_j. This permutation defines an ordering for successors generation. When a successor is created from a candidate E_t, we can assume that all solutions reachable from n by fixing the arcs corresponding to the candidates $E_1, \dots, E_{t-1}$ will be explored from the successors associated to these candidates. So, for the successor state s generated from E_t the following sets of disjunctive arcs can be fixed: $F_j = \{u^j_1 \to i; i = u^j_2, \dots, u^j_{mj}\}$, for each $E^B_j < E_t$ and $L_j = \{i \to u^j_{mj}; i = u^j_1, \dots, u^j_{mj-1}\}$, for each $E^A_j < E_t$ in the permutation above. So the successors of a search tree node n generated from the permutation $(E_1, \dots, E_{2k})$ are defined as follows. For each operation $l \in E^B_j$ generate a search tree node s by fixing the arcs $FD_s = FD_n \cup S^B_j$, provided that the resulting partial solution graph has no cycles, with

$$S^B_j = \bigcup_{E^B_i < E^B_j} F_i \cup \bigcup_{E^A_i < E^B_j} L_i \cup \{l \to i : i \in B_j \backslash \{l\}\}. \tag{3}$$

And for each operation $l \in E^A_j$ generate a search tree node s by fixing the arcs $FD_s = FD_n \cup S^A_j$ with

$$S^A_j = \bigcup_{E^B_i < E^A_j} F_i \cup \bigcup_{E^A_i < E^A_j} L_i \cup \{i \to l : i \in B_j \backslash \{l\}\}. \tag{4}$$

4.6 Fixing additional arcs by constraint propagation

After adjusting heads and tails, new disjunctive arcs can be fixed by the constraint propagation method due to Carlier and Pinson [Carlier and Pinson, 1989], termed immediate selection. In the sequel, UB denotes an upper bound of the optimal solutions, I denotes the set of operations requiring a given machine and n a search state. For each operation $j \in I$, r_j and q_j denote the head and tail respectively of the operation j in the state n.

Algorithm 2 PROCEDURE Select

```
for all $c, j \in I, c \neq j$ do
  if $r_c + p_c + p_j + q_j \geq UB$ then
    fix the arc $(j \to c)$;
  end if
end for
```

Theorem 2. *Let $c, j \in I, c \neq j$. If*

$$r_c + p_c + p_j + q_j \geq UB,$$

j has to be processed before c in every solution reachable from state n that improves UB.

The arc $(j \to c)$ is called *direct arc* and the procedure Select given in Algorithm 2 calculates all direct arcs for the state n in a time of order $O(|I|^2)$.

The procedure Select can be combined with the method due to Carlier and Pinson that allows improving heads and tails. This method is based on the following result.

Theorem 3. *Let $c \in I$ and $J \subseteq I \backslash \{c\}$.*

(1) If

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB, \tag{5}$$

then in all solutions reachable from state n improving UB, the operation c has to be processed after all operations in J.

(2) If

$$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J \cup \{c\}} q_j \geq UB, \tag{6}$$

then in all solutions reachable from state n improving UB, the operation c has to be processed before all operations in J.

If condition (1) of the theorem above holds, then the arcs $\{j \to c; j \in J\}$ can be fixed. These arcs are called *primal arcs* and the pair (J, c) is called *primal pair*. Similarly, if condition (2) holds, the *dual arcs* $\{c \to j; j \in J\}$ can be fixed and (c, J) is a *dual pair*.

In [Brucker et al., 1994], an efficient method is derived to calculate all primal and dual arcs. This method is based on the following ideas. If (J, c) is primal pair, the operation c cannot start at a time lower than

$$r_J = \max_{J' \subseteq J} \{\min_{j \in J'} r_j + \sum_{j \in J'} p_j\}. \tag{7}$$

So, if $r_c < r_J$, we can set $r_c = r_J$ and then the procedure Select fixes all primal arcs $\{j \to c; j \in J\}$.

This fact leads to the following problem.

Definition 1 (Primal problem). *Let $c \in I$. Does there exist a primal pair (J, c) such that $r_c < r_J$? If it exists, find*

$$r_{J^*} = \max\{r_J; (J, c)\, is\, a\, primal\, pair\}. \tag{8}$$

In [Carlier and Pinson, 1989], Carlier and Pinson propose an algorithm which is also based on JPS calculations to solve the primal problem in a time $O(k \log k)$, with $k = |I|$. So, all primal pairs can be computed in $O(k^2 \log k)$. This algorithm is fairly complicated so as it cannot be explained here for the lack of space. We refer the interested reader to Carlier and Pinson's or Brucker et al.'s papers.

Analogously, if (c, J) is dual pair, q_c cannot be lower than

$$q_J = \max_{J' \subseteq J} \{ \sum_{j \in J'} p_j + \min_{j \in J'} q_j \}. \quad (9)$$

So, if $q_c < q_J$, we can set $q_c = q_J$ and then the procedure Select fixes all dual arcs $\{c \rightarrow j; j \in J\}$. All dual pairs can be obtained similarly.

Finally, the algorithm used to fix additional disjunctive arcs proceeds as follows:

(1) calculation of all primal arcs for all machines,

(2) calculation of new heads and tails,

(3) calculation of all dual arcs for all machines,

(4) calculation of new heads and tails.

As new heads and tails are computed in steps 2 and 4 due to the additional arcs fixed in steps 1 and 3, steps 1-4 should be repeated as long as new disjunctive arcs are fixed.

5 Improving the heuristic using constraint propagation

The underlying idea used here to improve the heuristic estimation given by $f(n)$ is the following. Should we can prove that a solution with cost $f(n)$ may not be reached from state n, then the heuristic estimation could be increased up to $f(n) + 1$. To do that, we propose using the immediate selection procedure described above, but fixing the value of the upper bound to $f(n) + 1$. This process may be repeated if the heuristic estimation improves, i.e. the resulting partial solution graph contains any cycle. This suggest an iterative search over the sequence $f(n) + 1, f(n) + 2, \ldots$. To reduce the consumed time, we have opted for a binary search in the interval $[f(n) - 1 \ldots UB]$, instead. In the experimental study, we also consider a simplification of this method where the algorithm to improve heads and tails is not used. In this case, the time taken is expected to be lower at the cost of a lower improvement as well. The resulting heuristic is clearly admissible and it can be proved to be consistent.

6 Experimental study

In this experimental study we analyze the behavior of both algorithms, $B\&B$ and A^*, when they use simple and improved heuristic estimations in two different conditions: when they are able to solve the instances and when they cannot reach a solution within a given time. In this later case, both algorithms return a lower bound and an upper bound of the optimal solutions. To do that, we have considered two different sets of problems taken from the OR-library. Firstly, a set of instances FT10, LA16-20, ORB01-10 and ABZ5-6 of size 10×10 (10 jobs and 10 machines), as this is the threshold size for square instances to be considered interesting and both algorithms can solve them in

Table 1. Summary of results across 10×10 instances.

Average	$B\&B$	A^*	A^* I	A^* IR
Expanded	6396	19879	7542	8460
Generated	62428	195209	86192	95848
Inserted in OPEN	6445	23283	21805	22982
Time(s)	15,28	46,78	94,44	77,28

at most a few minutes. Then, we considered a set of large instances with sizes 15×10 (LA21,24,25), 20×10 (LA27,29) and 20×15 (LA38,40), which are beyond the size of problems that both algorithms can solve in 30 minutes. The target machine was Linux (Ubuntu V.8.04) on Intel Core 2 Duo (2,13 GHz, 7 GiB RAM).

Table 1 shows the average values of expanded and generated nodes, and the time taken in each case across the 10×10 instances. In addition to $B\&B$, A^* and A^* with the improved heuristic described in Section 5 (A^* I), a new version (A^* IR) is included, where the improvement procedure is simplified by skipping the operation of improving heads and tails from the solution of the primal and dual problems.

As we can observe, $B\&B$ is the best in all three measures taken. However, these results show that A^* is also a suitable approach as it can solve all the instances. In the first version, the time taken by A^* is about 3 times the time taken by $B\&B$. Moreover, with the improved versions of the heuristic estimation, the time is even larger, even though the number of expanded and generated nodes decreases in more than 50 percent. However, the number of nodes that are finally inserted in the OPEN list is quite similar with all versions of A^* and, in any case, is about 3,5 times the number of nodes inserted by $B\&B$. We have also experimented with $B\&B$ and the improved versions of the heuristic estimation and the results were not good: the time taken was larger and the number of expanded nodes was very similar.

Table 2 shows the results across the larger and more difficult instances. As we can see none of the four algorithms reaches the optimal solution in a time of 30 minutes (even A^* runs out of memory for all instances before this time). So we report the UB and LB reached in any case ($B.K.$ denotes the best know values). Clearly $B\&B$ gets the best UB's, while the best LB's are reached by the improved versions of A^*, in particular by A^* IR. This is a real advantage of these last two versions with respect to $B\&B$ and A^*. Clearly, this advantage is due to the improved heuristic estimation.

Table 2. Summary of results across the larger instances.

	$B.K.$		$B\&B$		A^*		A^* I		A^* IR	
Instances	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB
$LA21$	1046	1046	1069	995	1073	995	1099	1004	1098	1007
$LA24$	927	927	957	881	958	889	962	922	962	923
$LA25$	977	977	978	894	998	924	999	948	999	949
$LA27$	1235	1235	1314	1235	1311	1235	1309	1235	1309	1235
$LA29$	1153	1130	1218	1114	1239	1114	1241	1114	1225	1114
$LA38$	1196	1196	1228	1077	1254	1129	1255	1160	1255	1163
$LA40$	1222	1222	1262	1170	1277	1170	1264	1178	1264	1180

7 Conclusions

We have proposed a method to improve heuristic estimations for the Job Shop Scheduling Problem with makespan minimization. We have exploited this method with Brucker's $B\&B$ algorithm and with an A^* version of this algorithm as well. The first relevant conclusion is that A^* can solve instances up to a size of at least 10×10, which is the threshold size for non-trivial instances. For instances that both algorithms can solve optimally, the $B\&B$ algorithm is more efficient in a factor of about 3 or 4 times. We have observed that the $B\&B$ algorithm usually obtains better upper bounds earlier than A^*, so this is clearly one of the reasons for it to perform better. However, for instances that remain unsolved with both algorithms, A^* produces much better lower bounds thanks to the proposed heuristic improvement, while $B\&B$ still produces the best upper bounds. So, we can conclude that both approaches are of real interest as they complement each other. These results suggest us designing new approaches based on both algorithms. For example, they could be run in parallel so as they share the best current upper and lower bounds. In this case, we expect that at least A^* will improve. Furthermore, a partially informed depth first search will probably take profit from the improved heuristic, at difference of $B\&B$, so as it could perform better. Besides, some other constraint propagation rules, such as those proposed in [Dorndorf et al., 2000], could also be used to further improve heuristic estimations. Finally, the A^* algorithm could also improve using pruning by dominance techniques, such as the one proposed in [Sierra and Varela, 2008].

References

[Applegate and Cook, 1991] Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 3:149–156.

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

[Brucker et al., 1994] Brucker, P., Jurisch, B., and Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127.

[Carlier, 1982] Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47.

[Carlier and Pinson, 1989] Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

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

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

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

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

[Sierra and Varela, 2008] Sierra, M. and Varela, R. (2008). Pruning by dominance in best-first search for the job shop scheduling problem with total flow time. *Journal of Intelligent Manufacturing, DOI 10.1007/s10845-008-0167-4*, 1:1–2.