A New Admissible Heuristic for the Job Shop Scheduling Problem with Total Flow Time

María R. Sierra
Dept. of Mathematics, Statistics and Computing, University of Cantabria, 39005 Santander (Spain)
e-mail: sierramr@unican.es

Ramiro Varela
Dept. of Computer Science and A.I. Centre, University of Oviedo, 33271 Gijón (Spain)
e-mail: ramiro@uniovi.es

Abstract

In this paper, we confront the Job Shop Scheduling problem with total flow time minimization by means of the A^* algorithm. We propose a new heuristic based on problem relaxation to One Machine Sequencing problem with tardiness minimization. The heuristic is improved by means of the well-known generalized Emmons' constraint propagation rules. Additionally, we use a pruning by dominance method to reduce the effective search space. We report results from an experimental study conducted to evaluate the performance of the proposed heuristic and to compare the A* approach with a classic local search procedure. The results show that the proposed heuristic is efficient as A* is able to reach optimal solutions taking a time lower than the time taken by the local search algorithm.

Introduction

The Job Shop Scheduling Problem (JSSP) is a paradigm of optimization and constraint satisfaction problems that has interested to researches over the last years. Traditionally, the optimization criteria is makespan minimization. For this version of the problem, very efficient exact and approximate methods have been proposed in the literature, most of them relying on the concept of $critical\ path$. Among the exact methods, the most relevant is the branch and bound algorithm proposed by Brucker et al. in (Brucker, Jurisch, & Sievers 1994; Brucker 2004), developed from concepts and techniques proposed by some other authors as Carlier and Pinson (Carlier & Pinson 1989; 1994). Regarding non-exact methods, the most relevant are the local search techniques based on the neighborhood structures proposed firstly by Dell' Amico and Trubian (Dell' Amico & Trubian 1993) that were used and developed afterwards by many other authors, often in conjunction with one or various meta-heuristics such as Genetic Algorithms (Mattfeld 1995; Nowicki & Smutnicki 1996; Yamada & Nakano 1996) Tabu Search or Simulated Annealing (Zhang *et al.* 2008). Unfortunately, all these techniques, relaying on critical paths, are not so useful when the optimization criteria is other than makespan minimization.

In this paper we confront the JSSP problem with total flow time minimization by means of the A^* algorithm (Hart, Nilsson, & Raphael 1968; Nilsson 1980; Pearl 1984). We build on a previous work (Sierra & Varela 2007) where we confronted the JSSP with makespan minimization. The search space is that of active schedules and the branching schema is based on the well-known G&T algorithm (Giffler & Thomson 1960). We propose a new heuristic combining a problem relaxation to the One Machine Sequencing (OMS) problem with the well-known generalized Emmons' constraint propagation rules for the preemptive OMS. In order to reduce the effective search space, we adapted the pruning by dominance method proposed in (Sierra & Varela 2007).

The paper is organized as follows. In section 2 the JSSP is formulated. Section 3 describes the search space of active schedules for the JSSP. Section 4 summarizes the main characteristics of A^* algorithm. In section 5, the heuristic used to guide A^* for the JSSP with total flow time is described. Section 6 describes the generalized Emmons' rules and how they are applied to improve the heuristic. Section 7 reviews the concepts of dominance and describes the rule for testing it for the JSSP with total flow time. Section 8 review the local search procedure proposed in (Kreipl 2000). Section 9 reports results from an experimental study. Finally, section 10 summarizes the main conclusions.

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 starting time $st_{\theta_{il}}$ whose value should be determined. The JSSP has three constraints: precedence, capacity and no-preemption. Precedence constraints translate into linear inequalities of the type: $st_{\theta_{il}} + p_{\theta_{il}} \leq st_{\theta_{i(l+1)}}$. Capacity constraints translate into disjunctive constraints of the form: $st_v + p_v \leq st_w \vee st_w + p_w \leq st_v$, if $R_v = R_w$. No-preemption requires that the machine is assigned to an operation without interruption during its whole processing time. The objective is to come up with a feasible schedule such that a given objective function is optimized. As pointed in (Brucker 2004), three objective functions are commonly used: makespan, total flow time and tardiness. In this paper, we focus on minimizing the total flow time, i.e. the sum of completion times of all jobs. The problem is denoted as $J//\sum C_i$ in the conventional $\alpha/\beta/\gamma$ notation used in the literature.

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 $E = \cup_{\{i=1,...,M\}} E_i$. E_i includes an $arc\ (v, w)$ for each pair of operations requiring R_i. The arcs are weighed with the processing time of the operation at the source node. Node *start* is connected to the first operation of each job and the last operation of each job is connected to node *end*.

A feasible schedule is represented by an acyclic subgraph G_s of G, $G_s = (V, A \cup H)$, where $H = \cup_{i=1,...,M} H_i$, H_i being a processing ordering for the operations requiring R_i. The completion time of a job is the cost of a longest path from node *start* to node *end* restricted to pass through the last operation of the job just before the node *end*.

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v, i.e. it is the value of st_v. The *tail* q_v is defined as the processing time of operations after v in the job sequence. PM_v and SM_v denote the predecessor and successor of v respectively on the machine sequence and PJ_v and SJ_v denote the predecessor and successor nodes of v respectively on its job.

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

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

where $P(v)$ denotes the disjunctive predecessors of v, i.e. operations requiring machine R_v which are scheduled before v. Hence, the value $r_v + p_v + q_v$ is a lower bound of the completion time of the job of operation v that can be reached from the partial schedule. So these values allow to obtain a lower bound of the total flow time. This lower bound is not very tight and may be improved as we will see in section .

The Search Space of Active Schedules

A schedule is *active* if to start earlier any operation, at least another one should be delayed. The search space of active schedules is dominant for the $J//\sum C_i$, i.e. it contains at least and optimal schedule. Maybe the most appropriate strategy to obtain active schedules is the *G&T* algorithm proposed in (Giffler & Thomson 1960). This is a greedy algorithm that produces an active schedule in a number of $N * M$ steps. At each step *G&T* makes a non-deterministic choice. Every active schedule can be reached by taking the appropriate sequence of choices. Therefore, by considering all choices, we have a complete search tree for strategies such as branch and bound, backtracking or A^*.

Algorithm 1 SUC(state n). Algorithm to expand a state n. When it is successively applied from the initial state, i.e. an empty schedule, it generates the whole search space of active schedules.

1. $A = \{v \in US(n); PJ_v \in SC(n)\}$;
2. $v = \arg\min\{r_u + p_u; u \in A\}$;
3. $B = \{w \in A; R_w = R_v$ and $r_w < r_v + p_v\}$;

for each $w \in B$ **do**
 4. $SC(n') = SC(n) \cup \{w\}$ and $US(n') = US(n) \backslash \{w\}$;
 5. $G_{n'} = G_n \cup \{w \to v; v \in US(n'), R_v = R_w\}$;
 $\backslash *$ *w gets scheduled at time r_v in state n'* $* \backslash$
 6. $c(n, n') = r_w + p_w - (r_{PJ_w} + p_{PJ_w})$;
 7. Add n' to successors;

end for

8. return successors;

Algorithm 1 shows the expansion operation that generates the full search tree when it is applied successively from the initial state, in which none of the operations are scheduled yet. In the sequel, we will use the following notation.

In the sequel, we will use the following notation. Let O denote the set of operations of a problem instance, and n_1 and n_2 be two search states. In n_1, O can be decomposed into the disjoint union $SC(n_1) \cup US(n_1)$, where $SC(n_1)$ denotes the set of operations scheduled in n_1 and $US(n_1)$ denotes the unscheduled ones. $D(n_1) = |SC(n_1)|$ is the depth of node n_1 in the search space. Given $O' \subseteq O$, $\mathbf{r}_{n_1}(O')$ is the vector of heads of operations O' in state n_1. $\mathbf{r}_{n_1}(O') \leq \mathbf{r}_{n_2}(O')$ iff for each operation $v \in O'$, $r_v(n_1) \leq r_v(n_2)$, $r_v(n_1)$ and $r_v(n_2)$ being the head of operation v in states n_1 and n_2 respectively. Analogously, $\mathbf{q}_{n_1}(O')$ is the vector of tails of operations O' in state n_1; and $\mathbf{p}_{n_1}(O')$ denotes the vector of processing times. We also denote

$$J(n) = (US(n),\ \mathbf{r}_n(US(n)),\ \mathbf{p}_n(US(n)),\ \mathbf{q}_n(US(n))) \tag{2}$$

the residual problem of state n and consider $J = J_m \cup J_{\overline{m}}$, where J_m denotes the set of jobs with any operation in $US(n)$ requiring machine m and $J_{\overline{m}}$ denotes the subset of jobs with operations in $US(n)$ none of them requiring machine m. To obtain relaxed models, we firstly split the original problem $J(n)$ into subproblems $J(n)|_m$ and $J(n)|_{\overline{m}}$ and consider these problems independent from each other. Then we devise relaxations for each of them and calculate the lower bound of the original problem by summing up the optimal costs, or a lower bound, of both relaxed problems. Problem $J(n)|_m$ is given by unscheduled operations of jobs in J_m and problem $J(n)|_{\overline{m}}$ is given by unscheduled operations of jobs in $J_{\overline{m}}$.

A^* Nilsson's Algorithm

For best-first search we have chosen the A^* Nilsson's algorithm (Nilsson 1980). A^* starts from an initial state s, a set of goal nodes Γ and a transition operator SUC such that for each node n of the search space, $SUC(n)$ returns the set of successor states of n. Each transition from n to n' has a positive cost $c(n, n')$. $P^*_{s\text{-}n}$ denotes the minimum cost path from node s to node n. The algorithm searches for a path $P^*_{s\text{-}o}$ with $o = \arg\min\{P^*_{s\text{-}o'} | o' \in \Gamma\}$.

The set of candidate nodes to be expanded are maintained in an ordered list $OPEN$. The next node to be expanded is that with the lowest value of the evaluation function f, defined as $f(n) = g(n)+h(n)$; where $g(n)$ is the minimal cost known so far from s to n, (of course if the search space is a tree, the value of $g(n)$ does not change as there is one only path from s to n, otherwise this value has to be updated as long as the search progresses) and $h(n)$ is a heuristic positive estimation of the minimal distance from n to the nearest goal. If the node selected for expansion is an objective, the algorithm stops and returns the solution reached.

If the heuristic function underestimates the actual minimal cost, $h^*(n)$, from n to the goals, i.e. $h(n) \leq h^*(n)$, for every node n, the algorithm is admissible, i.e. it returns an optimal solution. Moreover, if $h(n_1) \leq h(n_2) + c(n_1, n_2)$ for every pair of states n_1, n_2 of the search graph, h is consistent. Two of the properties of consistent heuristics are that they are admissible and that the sequence of values $f(n)$ of the expanded nodes is non-decreasing. The heuristic function $h(n)$ represents knowledge about the problem domain. As long as h approximates h^* the algorithm gets more and more efficient as it needs to expand a lower number of states to reach the optimal solution.

New Heuristic for the JSSP with Total Flow Time

In order to design the heuristic, we consider the following problem relaxation. For jobs in J_m, every capacity constraint involving two operations of $US(n)$ is relaxed, except those involving operations requiring machine m; while capacity constraints involving an operation of $SC(n)$ and an operation of $US(n)$ are maintained in the relaxed model. In this way, the relaxed problem is equivalent to the OMS problem with heads, due dates and total tardiness minimization. The tardiness of an operation θ_{ij} is defined as $T_{ij} = max(0, C_{ij} - d_{\theta ij})$, where C_{ij} is the completion of operation θ_{ij} and d_{ij} the due date. We denote this problem as

$$J'(n)|_m = (US(n)|_m,\ \mathbf{r}_n(US(n)|_m), \mathbf{p}_n(US(n)|_m),\ \mathbf{d}_n(US(n)|_m)), \tag{3}$$

where $\mathbf{d}_n(US(n)|_m)$ denotes the due dates of operations given by $d_{\theta ij} = r_{\theta iM} + p_{\theta iM} - q_{\theta ij}$ for each θ_{ij} in $US(n)|_m$.

The problem $J'(n)|_m$ with total tardiness minimization is NP-hard in the strong sense (Rinnooy 1976). Moreover, even if preemption is allowed, the resulting problem is still NP-hard. In (Baptiste, Carlier, & Jouglet 2004), P. Baptiste el al. introduce a new lower bound for the preemptive case which is obtained from due date relaxations and it is based on the following two results. In order to simplify notation, let $(\theta_1, ..., \theta_k)$ denote the operations to be scheduled on the same machine and r_i, p_i and d_i their heads, processing times and due dates respectively.

Proposition 1 *Let θ_u and θ_v be two operations such that $r_u \leq r_v$, $p_u \leq p_v$ and $d_u \leq d_v$, then there exist and optimal schedule in which θ_v starts after the end of θ_u.*

Proposition 2 *Let θ_u and θ_v be two operations such that $r_u \leq r_v$, $p_u \leq p_v$ and $d_u > d_v$. Exchanging d_u and d_v does not increase the optimal total tardiness.*

These propositions allow to compute a lower bound by means of the following algorithm. Starting at time t given by the minimum of the heads of all operations, the set $D = \{\theta_u / r_u \leq t \wedge p'_u > 0\}$ of operations available but not completed at t is considered (p'_u denotes the remaining processing time of operation θ_u at time t). Let θ_u be the operation with the shortest remaining processing time, and θ_v the operation with the smallest due date. If $d_u = d_v$, according to Proposition 1 it is optimal to schedule the operation θ_u from t until a time t' given by the minimum of the completion time of θ_u, i.e. $t + p'_u$, and the time when a new operation is available. If it is not the case, according to Proposition 2, the due dates of θ_u and θ_v are exchanged. θ_u has now the smallest remaining processing time and the smallest due date, and the new instance has an optimal tardiness equal or lower than the original one. According to Proposition 1, for the new instance, it is optimal to schedule θ_u from time t up to t' obtained as before. The value of t is increased up to t' and the iteration continues until all operations are completed. This algorithm runs in O(k log k).

From the preemptive schedule calculated by this algorithm for problem $J'(n)|_m$, we actually obtain a lower bound of $f^*(n)$ as

$$F(J(n)) = \max_{m \in R}\{\textstyle\sum_{J_i \in J_m}(T_{\theta_i{}^m} + r_{\theta_{iM}} + p_{\theta_{iM}}) + \sum_{J_i \in J_{\overline{m}}}(r_{\theta_{iM}} + p_{\theta_{iM}})\} \tag{4}$$

where ${\theta_i}^m$ denotes the operation of job J_i requiring machine m and $T_{\theta_i{}^m}$ its tardiness in the peemptive schedule. So, to obtain the value of the heuristic estimation $h(n)$, the value of $g(n)$ should be discounted and so the heuristic is calculated as

$$h(n) = F(J(n)) - g(n). \tag{5}$$

As h is not obtained from an optimal solution of a relaxed problem but from a lower bound, it is admissible but it might not be consistent.

Improving the Heuristic with the Generalized Emmons Rules

Heuristic h may be improved by exploiting a set of dominance rules proposed in (Baptiste, Carlier, & Jouglet 2004) that generalize the well-known Emmons rules proposed in (Emmons 1969). The original Emmons rules allow to deduce some precedence relations for the special case where the release dates are equal and preemption is not allowed. The generalization to arbitrary release dates is possible if the non-preemption constraint is relaxed. In this case the resulting generalized Emmons rules can be used to tighten the lower bound of the preemptive problem.

The application of these rules requires a deadline δ_u for each operation θ_u. The value of δ_u may be initiated to the completion time of an active schedule. It is easy to see that all active schedules have the same completion time C_{max}. To compute this value, a schedule can be built where jobs are scheduled in non-decreasing order of release dates. Then the value of δ_u can be tighten since each operation θ_u cannot be completed after $C_{max} - \Sigma_{\theta_u \in A_u} p_u$, where A_u is a set of operations that have determined to start after θ_u completes. Analogously, the release date r'_u of operation θ_u can be initiated as r_i and then adjusted to $\max(r'_u, C_{max}(B_u))$, where B_u is a set of operations that have to be completed before operation θ_u can start and $C_{max}(B_u)$ is the completion time of an active schedule of operations of B_u. The generalized Emmons rules are given in the following three propositions.

Proposition 3 *(Generalized Emmons rule 1) Let S be a schedule and θ_u and θ_v two operations such that $r_u \leq r_v$, $p_u \leq p_v$ and $d_u \leq d_v + p_v$, then there exist a schedule S' in which θ_v starts after the end of θ_u and the tardiness of S' is lower than or equal to the tardiness of S.*

Proposition 4 *(Generalized Emmons rule 2) Let S be a schedule and θ_u and θ_v two operations such that $r_u \leq r_v$, $d_u \leq v_v$ and $\delta_u \leq max(r_v + p_v, d_v)$, then there exist a schedule S' in which θ_v starts after the end of θ_u and the tardiness of S' is lower than or equal to the tardiness of S.*

Proposition 5 *(Generalized Emmons rule 3) Let S be a schedule and θ_u and θ_v two operations such that $r_u \leq r_v$, $\delta_u \leq p_v$, then there exist a schedule S' in which θ_v starts after the end of θ_u and the tardiness of S' is lower than or equal to the tardiness of S.*

These rules may be applied at each step of the algorithm described in previous section to compute the heuristic h to discard some operations among the candidates to be scheduled next. For example, if at time t there are two ready operations θ_u and θ_v, i.e. their release dates are $r'_u = r'_v = t$, such that $p'_u \leq p'_v$ and $d_u \leq d_v + p'_v$, then as a consequence of the generalized Emmons rule 1, the remaining of θ_u can be scheduled before starting operation θ_v and both the deadline of θ_u and the release date of θ_v may be adjusted accordingly. Analogous reasoning may be followed from rules 2 and 3. As pointed in (Baptiste, Carlier, & Jouglet 2004) this improved lower bound can be computed in $O(k^4)$. Although, in practice, the propagation of the generalized Emmons rules is computed in a reasonable amount of time.

Dominance Properties

Given two states n_1 and n_2, we say that n_1 dominates n_2 if and only if the best solution reachable from n_1 is better, or at least of the same quality, than the best solution reachable from n_2. In some situations this fact can be detected and then the dominated state can be early pruned. Let us consider a small example.

Figure 1 shows the Gantt charts of two partial schedules, with three operations scheduled, corresponding to search states for a problem with 2 jobs and 3 machines. If the second operation of job J_1 requires R_2 and the third operation of J_2 requires R_3, it is easy to see that the best solution reachable from the state of Figure 1a can not be better than the best solution reachable from the state of Figure 1b. This is due to the fact that the residual problem comprises the same set of operations in both states and in the first state the heads of all operations are larger or at least equal than the heads in the second state. So, the state of Figure 1a may be pruned if both states are stored in memory at the same time. Of course, a good heuristic will lead the search to explore first the state of Figure 1b if both of them are in $OPEN$ at the same time. However, at a later time, the state of Figure 1a and a number of its descendants might also be expanded. Consequently, early pruning of this state can reduce the space and, if the matching of states to test dominance is efficient, the search time as well.

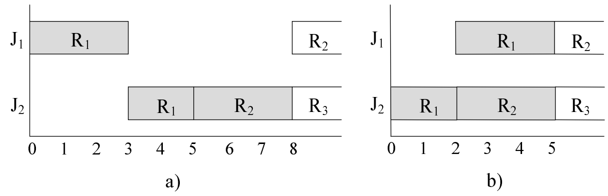

Figure 1: Partial schedules of two search states, state b) dominates state a)

We define dominance among states as follows.

Definition 1 *Given two states n_1 and n_2, such that $n_1 \notin P^*_{s\text{-}n_2}$ and $n_2 \notin P^*_{s\text{-}n_1}$, n_1 dominates n_2 if and only if $f^*(n_1) \leq f^*(n_2)$.*

Of course, establishing dominance among any two states is problem dependent and it is not easy in general. The next result gives a sufficient condition for a node n_1 dominates a node n_2. This condition can be efficiently evaluated as each expanded node have to be compared only with the nodes previously expanded having the same subset of operations scheduled.

Theorem 1 *Let n_1 and n_2 be two states such that $SC(n_1) = SC(n_2)$. Let us denote $SC = SC(n_1) = SC(n_2)$ and $US = US(n_1) = US(n_2)$ and $r_v(n)$ and $q_v(n)$ the head and tail respectively of operation v in state n. If*

$$\boldsymbol{r}_{n_1}(US) \leq \boldsymbol{r}_{n_2}(US) \tag{6}$$

and

$$\sum_{1 \leq i \leq N, \theta_{iM} \in SC} r_{\theta_{iM}}(n_1) \leq \sum_{1 \leq i \leq N, \theta_{iM} \in SC} r_{\theta_{iM}}(n_2) \tag{7}$$

then n_1 dominates n_2.

Proof 1 *(Sketch of the proof) As the heads of all unscheduled operations are lower in n_1 than they are in n_2, every solution to the remaining problem of n_2 is also a solution to the remaining problem of n_1.*

Local Search Algorithm

For the purpose of comparison with our A^* algorithm, in this paper we consider the local search algorithm proposed in (Kreipl 2000) termed $large\ step\ random\ walk$. In principle, this local search algorithm is devoted to the JSSP with weighted tardiness minimization. However, this problem is equivalent to the JSSP with total flow time if we consider all jobs having the same weight and all due dates having value 0. In the experimental study, we have used the implementation of this heuristic for the JSSP with total flow time included in the LEKIN tool downloaded from (http://www.stern.nyu.edu/om/software/lekin/index.htm).

The idea behind the large step random walk is the successive iteration of intensification phases, called small steps, and diversification phases, or large steps. The large step uses a metropolis algorithm and so it accepts worsening solutions in order to escape from local optima, whilst the small step uses a random descent method so as it always reaches a local optimum.

The neighborhood structure proposed in (Kreipl 2000) is similar to the structures defined in (Taillard 1994) and (Dell' Amico & Trubian 1993) for makespan minimization. It is also based on exchanges on critical paths. However, at difference from the case of makespan minimization where there is only one critical path, for total flow time there are as many critical paths as the number of jobs. So, the number of neighbors is in general very high and in practice the method is not so efficient as it is for makespan minimization. For further details of the large step random walk method we refer to the interested reader to the paper of S. Kreipl.

Experimental Study

The goal of this study is to assess the performance of the proposed A^* algorithm and also to compare it with other approach taken from the literature. As we have commented above, to do that we have chosen the local search procedure proposed in (Kreipl 2000). The target machine was Linux (Ubuntu V.8.04) on Intel Core 2 Duo (2,13 GHz., 7,5 Gb. RAM). We have considered the sets of problems $LA01$-05 with 10 jobs and 5 machines each and the $ORB1$-10 instances with 10 jobs and 10 machines each; all instances are taken from the $OR\text{-}library$. Instances $LA01$-05 are solved to optimality by A^*, whilst instances $ORB1$-10 are harder to solve so as the optimal solution is not known for all of them.

Table 1 shows the results obtained by A^* combining pruning by dominance, heuristic h_3 and the Emmons' rules across LA instances. It is clear that the pruning method allows reducing both the effective search space and the running time in about one order of magnitude. Regarding the improvement obtained from the Emmons' rules, when pruning is not applied they allow solving one more instance than the number of instances solved when these rules are not applied. When pruning is exploited, the Emmons' rules allow reducing both the number of expanded nodes and the time taken in about 30%.

Table 2 shows the results obtained by the local search procedure for instances $LA01$-05 with a time limit of 30s. This is the mean time taken by A^* to solve these instances, as we can see in Table 1. As we can observe, only one of the 5 instances gets solved to optimality. If the time limit is augmented to 64s. (the maximum time taken for A^* for these instances), the local search procedure solves 3 of the 5 instances.

Regarding instances $ORB1$-10, they are in the threshold of problem size that A^* is able to solve; only 2 of the 10 instances get solved, for 3 instances the algorithm stopped after $3600s$. without solution and for the remaining 5 the memory is exhausted before this time. We have experimented with instances of size 9×9 obtained from instances $ORB1$-10 by eliminating the last machine and the last job. The results of this experiment are reported in Table 3. All these instances get solved taking a time of $142s$. in average. Now, we experiment with the local search across these 9×9 instances leaving this time limit. As we can see in Table 4, only 5 of the 10 instances get solved. For the remaining 5 instances, we have also leaved the algorithm running $300s$. and in this case only one more instance gets solved, as we can see in Table 5.

Table 1: Summary of results from combining heuristic h_3, Emmons' rules and pruning by dominance over instances $LA01 - LA05$. Time limit 3600s. (Values in **bold** indicate that memory limit was exhausted without reach a solution.)

No pruning + h_3			
Instance	Generated	Expanded	Time(s)
$LA01$	1109048	471492	137
$LA02$	**3627888**	**1453887**	**441**
$LA03$	390090	153589	48
$LA04$	844107	339748	102
$LA05$	**3569995**	**1399554**	**436**
Pruning + h_3			
Instance	Generated	Expanded	Time(s)
$LA01$	246576	106896	37
$LA02$	516490	215975	84
$LA03$	79239	31832	11
$LA04$	135262	56746	21
$LA05$	410076	173443	70
No pruning + h_3 + Emmons			
Instance	Generated	Expanded	Time(s)
$LA01$	413499	175824	57
$LA02$	5184833	2156489	1501
$LA03$	181531	72591	25
$LA04$	605888	245079	79
$LA05$	**3622363**	**1451037**	**535**
Pruning + h_3 + Emmons			
Instance	Generated	Expanded	Time(s)
$LA01$	133391	57437	20
$LA02$	398687	166206	63
$LA03$	51467	20642	8
$LA04$	109852	45996	17
$LA05$	277781	116830	45

Table 2: Summary of results with Local Search over instances $LA01 - LA05$.

Instance	LA01	LA02	LA03	LA04	LA05
Best Sol.	4832	4459	4151	4259	4072
(30s.)	4833	4498	4151	4271	4131
(64s.)	4833	4459	4151	4271	4072

Table 3: Summary of results from combining heuristic h_3, Emmons' rules and pruning by dominance over instances $ORB\_9 \times 9$.

	Pruning + h_3 + Emmons		
Instance	Generated	Expanded	Time(s)
$ORB01\_9 \times 9$	547687	310954	131
$ORB02\_9 \times 9$	366068	209260	81
$ORB03\_9 \times 9$	378658	199359	95
$ORB04\_9 \times 9$	914526	533722	229
$ORB05\_9 \times 9$	74543	43514	16
$ORB06\_9 \times 9$	720815	390476	185
$ORB07\_9 \times 9$	481491	280638	124
$ORB08\_9 \times 9$	1458504	749257	431
$ORB09\_9 \times 9$	134458	78979	31
$ORB10\_9 \times 9$	368718	209593	92

Table 4: Summary of results with Local Search over instances $ORB\_9 \times 9$. Time limit 142s (Average).

Instance	Best Sol.	Sol. Reach.
$ORB01\_9 \times 9$	6367	6383
$ORB02\_9 \times 9$	5867	5868
$ORB03\_9 \times 9$	6310	6310
$ORB04\_9 \times 9$	6661	6661
$ORB05\_9 \times 9$	5605	5605
$ORB06\_9 \times 9$	6106	6106
$ORB07\_9 \times 9$	2668	2668
$ORB08\_9 \times 9$	5668	5717
$ORB09\_9 \times 9$	6013	6026
$ORB10\_9 \times 9$	6328	6333

Table 5: Summary of results with Local Search over instances $ORB\_9 \times 9$. Time limit 300s.

Instance	Best Sol.	Sol. Reach.
$ORB01\_9 \times 9$	6367	6383
$ORB02\_9 \times 9$	5867	5868
$ORB08\_9 \times 9$	5668	5693
$ORB09\_9 \times 9$	6013	6013
$ORB10\_9 \times 9$	6328	6333

Conclusions

In this paper we considered an A^* approach to the JSSP with total flow time minimization. We propose a new heuristic based on problem relaxation to the OMS problem with tardiness minimization. The A^* algorithm is enhanced with a pruning by dominance rule that allows reducing the effective search space. We report results from an experimental study showing that the proposed A^* algorithm is quite competitive with the local search procedure proposed in (Kreipl 2000).

As future work, we will consider other objective functions such as the weighted tardiness. Also, we plan to design similar approaches to other problems, such as the JSSP with sequence dependent setup times and the cutting stock problem.

Acknowledgments

This work has been supported by the Spanish Ministry of Science and Education under research project MEC-FEDER TIN2007-67466-C02-01.

References

Baptiste, P.; Carlier, J.; and Jouglet, A. 2004. A brach-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research* 158:595–608.

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.

Brucker, P. 2004. *Scheduling Algorithms*. Springer, 4th edition.

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

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

Dell' Amico, M., and Trubian, M. 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research* 41:231–252.

Emmons, H. 1969. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research* 17:701–715.

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

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Science and Cybernetics* 4(2):100–107.

Kreipl, S. 2000. A large step rendom walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling* 3:125–138.

Mattfeld, D. C. 1995. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Nowicki, E., and Smutnicki, C. 1996. A fast taboo search algorithm for the job shop scheduling problem. *Management Science* 42:797–813.

Pearl, J. 1984. Parallel machine scheduling models with fuzzy processing times. *Information Sciences* 166:49–66.

Rinnooy, A. H. G. 1976. Machine sequencing problem: Classification, complexity and computation. *Nijhoff, The Hague*.

Sierra, M., and Varela, R. 2007. Pruning by dominance in best-first search. In *Proceedings of CAEPIA'2007*, volume 2, 289–298.

Taillard, E. D. 1994. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing* 6(2):108–117.

Yamada, T., and Nakano, R. 1996. Scheduling by genetic local search with multi-step crossover. In *Proceedings of Fourth International Conference On Parallel Problem Solving from Nature (PPSN IV 1996)*, 960–969.

Zhang, C. Y.; Li, P.; Rao, Y.; and Guan, Z. 2008. A very fast ts/sa algorithm for the job shop scheduling problem. *Computers and Operations Research* 35:282–294.