Asia-Pacific Journal of Operational Research
Vol. 26, No. 3 (2009) 319–339
© World Scientific Publishing Co. & Operational Research Society of Singapore

BEAM SEARCH HEURISTICS FOR THE SINGLE MACHINE SCHEDULING PROBLEM WITH LINEAR EARLINESS AND QUADRATIC TARDINESS COSTS

JORGE M. S. VALENTE
LIAAD, Faculdade de Economia
Universidade do Porto, Portugal
jvalente@fep.up.pt

Received 9 October 2007
Accepted 30 May 2008

In this paper, we consider the single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. We present heuristic algorithms based on the beam search technique. These algorithms include classic beam search procedures, as well as the filtered and recovering variants. Several dispatching rules are considered as evaluation functions, to analyze the effect of different rules on the effectiveness of the beam search algorithms.

The computational results show that using better rules improves the performance of the beam search heuristics. The detailed, filtered beam search (FBS) and recovering beam search (RBS) procedures outperform the best existing heuristic. The best results are given by the recovering and detailed variants, which provide objective function values that are quite close to the optimum. For small to medium size instances, either of these procedures can be used. For larger instances, the detailed beam search (DBS) algorithm requires excessive computation times, and the RBS procedure then becomes the heuristic of choice.

Keywords: Scheduling; single machine; linear earliness; quadratic tardiness; beam search; heuristics.

1. Introduction

In this paper, we consider a single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. Formally, the problem can be stated as follows. A set of n independent jobs $\{J_1, J_2, \ldots, J_n\}$ has to be scheduled on a single machine that can handle at most one job at a time. The machine is continuously available from time zero onwards, and preemptions are not allowed. Job $J_j, j = 1, 2, \ldots, n$, requires a processing time p_j and should ideally be completed on its due date d_j. For a given schedule, the earliness and tardiness of J_j are defined as $E_j = \max\{0, d_j - C_j\}$ and $T_j = \max\{0, C_j - d_j\}$, respectively, where C_j is the completion time of J_j. The objective is to find a schedule that minimizes the sum of linear earliness and quadratic tardiness costs $\sum_{j=1}^{n}(E_j + T_j^2)$, subject to the constraint that no machine idle time is allowed.

Single machine scheduling environments actually occur in many practical operations. Moreover, the performance of many production systems is frequently determined by the quality of the schedules for a single bottleneck machine. Also, the analysis of single machine problems provides results and insights that can often be applied to more complex scheduling environments.

Scheduling models with both earliness and tardiness costs have received considerable attention from the scheduling community, due to their practical importance. Indeed, early/tardy scheduling problems are compatible with the concepts of just-in-time production and supply chain management. These production strategies, which have been adopted by many organizations, view both early and tardy deliveries as undesirable.

In this paper, a linear penalty is used for the early jobs, since early completions result in unnecessary inventory, and the costs incurred with this inventory tend to be proportional to the quantity held in stock. Late deliveries, on the other hand, can result in lost sales and loss of goodwill. A quadratic tardiness penalty is used for the tardy jobs, since a customer's dissatisfaction tends to increase quadratically with the tardiness, as proposed in the loss function of Taguchi (1986).

We assume that machine idle time is not allowed. This assumption is appropriate for many production settings. Indeed, idle time should be avoided when the machine has limited capacity or high operating costs, as well as when starting a new production run involves large set-up costs or times.

To the best of our knowledge, the complexity of the considered problem is still open. However, from existing complexity results for related problems, it seems likely that the considered problem is difficult. Indeed, Hall *et al.* (1991) established that the linear earliness and tardiness problem $\sum_{j=1}^{n}(E_j + T_j)$ with a common due date d for all jobs is NP-hard. Moreover, the quadratic earliness and tardiness problem $\sum_{j=1}^{n}(E_j^2 + T_j^2)$ with a common due date d was also shown to be NP-hard by Kubiak (1993).

The linear earliness and quadratic tardiness problem with no idle time has been previously considered by Valente (2008). Valente (2008) proposed a lower bounding procedure based on a relaxation of the job completion times, as well as a branch-and-bound algorithm. In Valente (2007), several dispatching heuristics are presented, and their performance is analyzed on a wide range of instance types. The corresponding problem with inserted idle time has been considered by Schaller (2004). He presented a timetabling procedure to optimally insert idle time in a given sequence, as well as a branch-and-bound algorithm and simple and efficient heuristics.

Some problems with related objective functions have also been considered. The single machine problem with linear earliness and tardiness penalties $\sum_{j=1}^{n}(E_j + T_j)$ has been studied by Garey *et al.* (1988), Kim and Yano (1994) and Schaller (2007). The minimization of the quadratic lateness $\sum_{j=1}^{n} L_j^2$, where the lateness of J_j is defined as $L_j = C_j - d_j$, has also been considered by Gupta and Sen (1983), Su and Chang (1998), Schaller (2002) and Sen *et al.* (1995). Baker and Scudder (1990) and

Hoogeveen (2005) provide excellent surveys of scheduling problems with earliness and tardiness penalties. Kanet and Sridharan (2000) give a review of scheduling models with inserted idle time that complements our focus on a problem with no machine idle time.

In this paper, we present several heuristic algorithms based on the beam search technique. These algorithms include classic beam search procedures, with both priority and total cost evaluation functions, as well as the more recent filtered and recovering variants. Beam search procedures require evaluation functions that are usually provided by a dispatching rule. We consider four dispatching heuristics, to analyze the effect of different rules on the performance of the beam search algorithms. Extensive preliminary computational experiments were performed to determine appropriate values for the parameters required by the beam search procedures. The performance of the four heuristic rules is also analyzed in these initial tests. The best performing versions of the beam search algorithms are then compared with the best existing heuristic, as well as with optimal solutions.

The remainder of this paper is organized as follows. In Sec. 2, we describe the beam search approach and its several variations, and present the proposed heuristic procedures and their implementation details. The computational results are reported in Sec. 3. Finally, some concluding remarks are given in Sec. 4.

2. The Beam Search Heuristics

2.1. *History and review*

Beam search is a heuristic method for solving combinatorial optimization problems. It consists of a truncated branch-and-bound procedure in which only the most promising nodes at each level of the search tree are kept for further branching, while the remaining nodes are pruned off. The classic or traditional beam search algorithm was first used in artificial intelligence problems by Lowerre (1976) and Rubin (1978).

Two variations of the classic beam search approach have since been proposed. Ow and Morton (1988, 1989) developed a variation called FBS. Recently, Della Croce and T'kindt (2002) and Della Croce *et al.* (2004) proposed an approach denoted by RBS.

Beam search algorithms have been applied to several combinatorial optimization problems, particularly in the scheduling field. Some recent applications of beam search heuristics to scheduling problems include Della Croce and T'kindt (2002), Della Croce *et al.* (2004), Valente and Alves (2005), Ghirardi and Potts (2005) and Esteve *et al.* (2006).

In the following subsections, we first present the classic beam search technique. Then, the more recent filtered and recovering approaches are described. The proposed beam search procedures, and some implementation details, are then presented. Finally, we provide a numerical example to illustrate one of the proposed procedures.

2.2. *Classic beam search*

The classic beam search method consists of a truncated branch-and-bound algorithm in which only the most promising β nodes at each level of the search tree are selected as nodes to branch from; β is the so-called *beam width*. The remaining nodes are ignored, and backtracking is not allowed. Therefore, the node evaluation process is crucial for the effectiveness of a beam search procedure. Two different types of evaluation functions have been used in classic beam search procedures: *priority evaluation functions* and *total cost evaluation functions*.

Priority evaluation functions calculate an urgency rating for the last job added to the current partial sequence, typically by using the priority index of a dispatching rule. Total cost evaluation functions calculate an estimate of the minimum total cost of the best solution that can be reached from the current node. This is usually done by using a dispatching rule to schedule the remaining jobs, to complete the existing partial sequence. Priority evaluation functions have a local view of the problem (they only consider the next decision to be made), while total cost evaluation functions have a global view (they project from the current partial solution to a complete schedule).

The priority evaluation functions can pose a slight problem. The priority index that is used to calculate the urgency rating of the last scheduled job usually depends on the current partial schedule, particularly on the current time. Therefore, the priority values are context-dependent, meaning that the priorities calculated for the offspring of one node cannot be legitimately compared with those obtained from the branching of another node. This problem, however, can be solved by initially selecting the best β children of the root node. Then, at lower levels of the search tree, only the best descendant of each beam node is kept for further branching, so the number of beam nodes is kept at β. The total cost evaluation functions are not affected by this problem, since total cost estimates can be compared.

The main steps of priority (PBS) and DBS algorithms are now presented. The priority (detailed) beam search procedure uses a priority (total cost) evaluation function. In the following, B is the set of beam nodes, C is a set of offspring nodes, and n_0 is the root node.

```
PBS:
Step 1. Initialization:
        Set $B = \varnothing$, $C = \varnothing$.
        Branch $n_0$, generating the corresponding children.
        Calculate the priority of the last scheduled job for each child node.
        Select the best $\beta$ child nodes and add them to $B$.
Step 2. Node selection:
        For each node in $B$:
          (a) Branch the node, generating the corresponding children.
          (b) Calculate the priority of the last scheduled job for each child node.
          (c) Select the best child node and add it to $C$.
        Set $B = C$ and $C = \varnothing$.
```

```
Step 3. Stopping condition:
        If the nodes in $B$ are leaf (i.e., they hold a complete sequence), select the
        node with the lowest total cost as the best sequence found and stop.
        Otherwise, go to Step 2.
```

DBS:

```
Step 1. Initialization:
        Set $B = \{n_0\}$ and $C = \varnothing$.
Step 2. Branching:
        For each node in $B$:
          (a) Branch the node, generating the corresponding children.
          (b) Calculate an upper bound on the optimal solution value for each child
              node.
          (c) Select the best $\beta$ child nodes and add them to $C$.
        Set $B = \varnothing$.
Step 3. Node selection:
        Select the best $\beta$ nodes in $C$ and add them to $B$.
        Set $C = \varnothing$.
Step 4. Stopping condition:
        If the nodes in $B$ are leaf, select the node with the lowest total cost as the
        best sequence found and stop.
        Otherwise, go to Step 2.
```

2.3. *Filtered and recovering beam search*

The priority evaluation functions are quick, but are rather crude and potentially inaccurate, and may result in discarding good solutions. Total cost evaluation functions are more accurate, but much more time consuming. The FBS and RBS algorithms try to combine crude and accurate evaluations, to provide a high quality evaluation within a reasonable computation time. This is achieved by using a two-stage approach. First, a computationally inexpensive *filtering step* is applied. In this step, a crude evaluation is performed, to select only a reduced number of the offspring of each beam node. The selected nodes are then accurately evaluated, and the best β nodes are kept for further branching.

Two different types of filtering step have been proposed. In the approach developed by Ow and Morton (1988, 1989), a priority evaluation function is used to calculate an urgency rating for each offspring. Then, the best α children of each beam node are selected for accurate evaluation, where α is the so-called *filter width*. The second type of filtering phase has been recently introduced by Della Croce and T'kindt (2002) and Della Croce *et al.* (2004). In this approach, problem-dependent dominance conditions (when available) are applied together with the so-called pseudo-dominance conditions (which hold in a heuristic context only). Whenever one of these conditions holds for a given node, that node is pruned.

The RBS approach differs from the FBS algorithm in two major ways. First, the accurate evaluation in the FBS procedure relies on an upper bound on the total cost of the best solution that can be reached from the current node. In the RBS approach, on the other hand, each node is evaluated by both lower and upper bounds. More specifically, each node is evaluated by the function $V = (1-\gamma)LB + \gamma UB$, where $0 \leq \gamma \leq 1$ is a user-defined upper bound weight parameter, and LB and UB are the lower and upper bound values, respectively.

Second, the RBS algorithm includes a so-called *recovering phase*. In this phase, the nodes that passed the filtering step are considered in non-decreasing order of their evaluation function. For each node, the recovering step checks whether the current partial solution σ is dominated by another partial solution $\overline{\sigma}$ with the same search tree level; this is typically done by applying neighborhood operators. If a better partial solution does exist, then σ is replaced by $\overline{\sigma}$. If the possibly modified node is not already in the set of beam nodes, then the node is added to B. This process is repeated until either β nodes have been chosen, or no additional candidate nodes remain.

Classic and FBS algorithms cannot recover from wrong decisions: if a node leading to the optimal solution is pruned, there is no way to reach that solution afterwards. The recovering step tries to overcome this problem, and often allows the RBS procedure to correct previous incorrect decisions. We now present the main steps of both FBS and RBS algorithms. In the RBS algorithm, let n_{best} and UB_{best} denote the current best node and the current best upper bound, respectively.

FBS:

Step 1. Initialization:
Set $B = \{n_0\}$ and $C = \varnothing$.

Step 2. Filtering step:
For each node in B:

(a) Branch the node, generating the corresponding children.
(b) Add to C all the child nodes that are not eliminated by the filtering procedure.

Set $B = \varnothing$.

Step 3. Node selection:
Calculate an upper bound on the optimal solution value for all nodes in C.
Select the best β nodes in C and add them to B.
Set $C = \varnothing$.

Step 4. Stopping condition:
If the nodes in B are leaf, select the node with the lowest total cost as the best sequence found and stop.
Otherwise, go to Step 2.

RBS:

```
Step 1. Initialization:
        Set $B = \{n_0\}$, $C = \varnothing$, $n_{best} = \varnothing$ and $UB_{best} = \infty$.
Step 2. Filtering step:
        For each node in $B$:
          (a) Branch the node, generating the corresponding children.
          (b) Add to $C$ all the child nodes that are not eliminated by the filtering
              procedure.
        Set $B = \varnothing$.
Step 3. Accurate evaluation:
        For all nodes $n_k, k = 1, \ldots, |C|$ in $C$:
          (a) Calculate a lower bound $LB_k$ and an upper bound $UB_k$ on the optimal
              solution value of node $n_k$.
          (b) Compute the evaluation function $V = (1 - \gamma)LB_k + \gamma UB_k$.
          (c) If $UB_k < UB_{best}$, set $n_{best} = n_k$ and $UB_{best} = UB_k$.
Step 4. Recovering step:
        Sort all nodes in $C$ in non-decreasing order of the evaluation function
        value.
        Set $k = 1$.
        While $|B| < \beta$ and $k \leq |C|$:
          (a) Let $\sigma$ represent the partial solution associated with the current node $n_k$.
          (b) Search for a partial solution $\overline{\sigma}$ that dominates $\sigma$ by means of neigh-
              borhood operators.
          (c) If $\overline{\sigma}$ is found, set $\sigma = \overline{\sigma}$.
          (d) If $n_k \notin B$
                i. Set $B = B \cup \{n_k\}$.
               ii. If $UB_k < UB_{best}$, set $n_{best} = n_k$ and $UB_{best} = UB_k$.
          (e) Set $k = k + 1$.
Step 5. Stopping condition:
        If the nodes in $B$ are leaf, stop with $n_{best}$ and $UB_{best}$ as the best node
        and lowest total cost found, respectively.
        Otherwise, go to Step 2.
```

2.4. *Implementation details*

The implementations details of the beam search heuristics will now be presented. We considered both PBS and DBS classic beam search algorithms, as well as FBS and RBS procedures. To apply these algorithms to the linear early/quadratic tardy problem, it is necessary to specify their main components, such as branching scheme, evaluation functions, filtering procedure, and recovering step. In the following, we first describe the branching scheme. Then, we present the dispatching rules that were used as evaluation functions. Finally, some additional implementation details are provided.

Branching scheme

The branching procedure is identical for all algorithms. A forward branching scheme is used: the sequence is constructed by adding one job at a time starting from the first position. Therefore, a branch at level l of the search tree indicates the job scheduled in position l.

Dispatching rules

A dispatching rule is required by the several beam search variants, to provide a priority evaluation function and/or to calculate an upper bound. Four dispatching heuristics were considered, with the purpose of analyzing the effect of different rules on the performance of the beam search procedures. More specifically, we used the EDD, SPT_s_j, CS_AS, and EQTP_EXP dispatching rules presented in Valente (2007).

The EDD rule sequences the jobs in non-decreasing order of their due dates. The SPT_s_j rule is derived from a local optimality condition for tardy jobs. This dispatching rule calculates a priority index for each remaining job every time the machine becomes available, and the job with the highest priority is selected to be processed next. Let $I_j(t)$ denote the priority index of job J_j at the current time t. The SPT_s_j rule uses the priority index $I_j(t) = (1/p_j)[\overline{p} + 2\max(t + p_j - d_j, 0)]$, where $\overline{p}$ is the average processing time of the remaining unscheduled jobs.

The EDD (SPT_s_j) heuristic performed well for instances where most jobs are early (tardy). The CS_AS procedure tries to take advantage of the strengths of the EDD and SPT_s_j rules. At each iteration, the CS_AS heuristic uses one of these rules to choose the next job, according to the characteristics of the current workload. More specifically, the CS_AS procedure first calculates a critical slack value $crit\_slack = slack\_prop * n_U * \overline{p}$, where n_U is the number of unscheduled jobs, and $0 \leq slack\_prop < 1$ is a user-defined parameter. Next, the average slack $\overline{s}$ of the remaining unscheduled jobs is determined (the slack of job J_j is defined as $s_j = d_j - t - p_j$). The EDD (SPT_s_j) rule is then selected if $\overline{s} > crit\_slack$ ($\overline{s} \leq crit\_slack$).

The EQTP_EXP dispatching rule was the best performing of the heuristics considered in Valente (2007). This procedure calculates a priority index for each unscheduled job at each iteration, and selects the job with the largest priority. More precisely, this heuristic uses the following priority index $I_j(t)$:

$$I_j(t) = \begin{cases} (1/p_j)[\overline{p} + 2(t + p_j - d_j)] & \text{if } s_j \leq 0 \\ (\overline{p}/p_j)\exp[-(\overline{p}+1)s_j/k\overline{p}] & \text{if } 0 < s_j < [\overline{p}/(\overline{p}+1)]k\overline{p} \\ (1/p_j)^{-2}[(\overline{p}/p_j) - (1/p_j)(\overline{p}+1)s_j/k\overline{p}]^3 & \text{if } [\overline{p}/(\overline{p}+1)]k\overline{p} \leq s_j < k\overline{p} \\ -(1/p_j) & \text{otherwise,} \end{cases}$$

where k is a lookahead parameter.

At each iteration, the lookahead parameter k is set equal to the number of critical jobs (i.e., jobs that are not yet tardy, but are about to become tardy). First, a critical slack value *crit_slack* is calculated, just as previously described for the CS_AS heuristic. Then, each job is classified as critical if $0 < s_j \leq crit\_slack$. Following the recommendations given in Valente (2007), the parameter *slack_prop* is set at 0.15 and 0.6 for the CS_AS and EQTP_EXP heuristics, respectively.

For each type of beam search procedure, we therefore considered four versions, corresponding to these four dispatching rules. In the following, the CS_AS and EQTP_EXP rules will be denoted simply as CS and EQTP.

Priority beam search

PBS algorithms require a priority evaluation function. For each of the four versions of the PBS procedure, the priority function is provided by the priority index of the appropriate dispatching rule (EDD, SPT_s_j, CS or EQTP). Therefore, the evaluation value of a node is obtained by calculating the appropriate priority index of the last scheduled job.

Detailed beam search

DBS algorithms require a total cost evaluation function, i.e., an upper bounding procedure. For each DBS version, this upper bounding procedure is provided by the appropriate dispatching heuristic. Therefore, and for a given node, the appropriate rule is used to sequence the remaining unscheduled jobs, thereby completing the existing partial schedule. The evaluation value of the node is then equal to the cost of the complete schedule.

Filtered beam search

FBS algorithms require a filtering procedure and an upper bounding procedure. The upper bounding procedure is provided by the relevant dispatching rule, just as previously described for the DBS algorithms. The filtering step uses a priority evaluation function filter, so a priority evaluation function is used to calculate an urgency rating for each offspring of a given node. The best α children are then chosen for the detailed evaluation step. The priority evaluation function is given by the priority index of the appropriate dispatching heuristic, just as previously described for the PBS algorithms.

Recovering beam search

RBS algorithms require a filtering procedure, upper and lower bounding procedures for the accurate evaluation step, and an improvement procedure for the recovering step. The filtering and upper bounding procedures are identical to those used in the FBS algorithms. The lower bounding procedure is provided by the method proposed in Valente (2008). For a given node, this procedure is used to calculate a lower bound for the unscheduled jobs. The lower bound of the node is then equal to the sum of the cost of the existing partial schedule and the lower bound for the unscheduled jobs.

We considered three simple improvement procedures for the recovering step: adjacent pair-wise interchange (API), 3-swaps (3SW), and largest cost insertion (LCI). The API procedure considers in succession all adjacent job positions. A pair of adjacent jobs is then swapped if such an interchange improves the objective function value. This process is repeated until no improvement is found. The 3SW procedure is similar, but it considers three consecutive job positions instead of an adjacent pair of jobs. All possible permutations of these three jobs are then analyzed, and the best configuration is selected. The LCI method selects the job with the largest objective function value. This job is then removed from its position i in the schedule, and inserted at position j, for all $j \neq i$. The best insertion is then performed if it improves the objective function value. This is repeated until no improving move is found.

2.5. *Numerical example*

In this section, a numerical example is used to illustrate one of the proposed beam search procedures. Consider an instance with four jobs, with processing times 8, 10, 3 and 5, and due dates 15, 10, 5 and 18, respectively. Let ( ) represent the root node n_0. Also, let $(x, y, \ldots)$ represent the node corresponding to the partial sequence x-y-.... In the following, we describe the application of the EDD version of the FBS procedure to this instance. A value of 2 is considered for both the beam and filter width parameters.

In step 1, the root node () is assigned to the set of beam nodes B, while set C is initialized as an empty set. In step 2, the only node in B is the root node. This node is then branched in step 2(a), generating the four offspring nodes (1), (2), (3), and (4). In the filtering step, a priority filter is used, and the two best nodes are then added to set C in step 2(b). The EDD rule sequences jobs in non-decreasing order of their due dates, i.e., it assigns a larger urgency rating to jobs with smaller due dates. Therefore, the two best nodes that are added to set C are nodes (3) and (2). In step 3, an upper bound is calculated for each of these nodes. This upper bound is obtained by using the EDD rule to sequence the remaining unscheduled jobs. For node (3), the sequence 3-2-1-4 is obtained, giving an upper bound value of 111. For node (2), the upper bound is equal to 164. The nodes (3) and (2) are then selected for further branching, and are assigned to set B.

Since the nodes in B do not correspond to a complete sequence, we return to step 2. Node (3) is first expanded, and the children nodes (3,1), (3,2) and (3,4) are obtained. The EDD filtering step then adds the nodes (3,2) and (3,1) to set C. Similarly, node (2) is branched, and its two best offspring (2,3) and (2,1) are added to C. In step 3, the EDD rule is used to calculate an upper bound for these nodes.

We then obtain upper bound values of 111, 191, 164, and 329 for nodes (3,2), (3,1), (2,3) and (2,1), respectively. Nodes (3,2) and (2,3) have the lowest upper bounds, and are assigned to set B. Repeating steps 2 and 3, we would obtain the two leaf nodes (3-2-1-4) and (3-2-4-1), with objective function values 111 and 132, respectively. The procedure then terminates with 3-2-1-4 as the best sequence found.

3. Computational Results

In this section, we first present the set of test problems used in the computational tests. The preliminary computational experiments are then described. These experiments were performed, on the one hand, to select appropriate values for the parameters required by the several beam search heuristics. On the other hand, these preliminary tests were also used to analyze the performance of the beam search procedures under the alternative rules that were considered (EDD, SPT_s_j, CS, and EQTP), to select the best performing rule. Finally, we present the computational results. The beam search procedures are first compared with the best existing dispatching heuristic, and the heuristic results are then evaluated against the optimum objective function values for the smaller instance sizes. The instances used in the computational tests are available online at http://www.fep.up.pt/docentes/jvalente/benchmarks.html. The objective function value provided by the EQTP, DBS, and RBS heuristics, as well as the optimum objective function value (when available), can also be obtained at this address.

3.1. *Experimental design*

The computational tests were performed on a set of problems with 10, 15, 20, 25, 30, 40, 50, 75, 100, 250, 500, and 750 jobs. These problems were randomly generated as follows. For each job J_j, an integer processing time p_j was generated from one of the two uniform distributions $[45, 55]$ and $[1, 100]$, to obtain low (L) and high (H) variability, respectively, for the processing time values. For each job J_j, an integer due date d_j was generated from the uniform distribution $[P(1 - T - R/2), P(1 - T + R/2)]$, where P is the sum of the processing times of all jobs, T is the tardiness factor, set at 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0, and R is the range of due dates, set at 0.2, 0.4, 0.6, and 0.8.

For each combination of problem size n, processing time variability (var), T and R, 50 instances were randomly generated. Therefore, a total of 1200 instances were generated for each combination of problem size and processing time variability. All the algorithms were coded in Visual C++ 6.0, and executed on a Pentium IV — 2.8 GHz personal computer. Due to the large computational times that would be required, the detailed beam search algorithm was only used on instances with up to 100 jobs, while the filtered and recovering procedures were not applied to the 750 job instances.

3.2. *Preliminary tests*

In this section, we describe the preliminary computational experiments. These experiments were used to determine adequate values for the various parameters required by the beam search algorithms. Moreover, these tests were also used to analyze the performance of the four heuristic rules that were considered for each beam search procedure, to select the best performing rule. A separate problem set was used to conduct these preliminary experiments. This test set included instances with 25, 50, 75, and 100 jobs, and contained five instances for each combination of instance size, processing time variability, T and R. The instances in this smaller test set were generated randomly just as previously described for the full problem set.

Extensive tests were first conducted to determine adequate values for the beam width, filter width, and upper bound weight parameters. A trade-off exists between solution quality and computation time, since increasing the beam or filter width parameters usually improves the objective function value, at the cost of increased computational effort. The following values were considered for these parameters: $\alpha = \{1, 2, \ldots, 10\}$, $\beta = \{1, 2, \ldots, 8\}$ and $\gamma = \{0.1, 0.2, \ldots, 0.9\}$. The preliminary tests were also used to select an adequate improvement procedure for the recovering step in the RBS algorithm. As mentioned before, we considered API, 3SW, and LCI procedures.

The several versions of the beam search algorithms were applied to the test instances for all combinations of the relevant parameter values (and improvement procedures, for the RBS heuristics). We then calculated and plotted the mean objective function values and runtimes. A thorough analysis of these data showed a usual behavior: the computation time increased linearly with the beam and filter width parameters, while the solution quality improved, but with diminishing returns. We then selected the parameter values and the improvement procedure that seemed to provide the best trade-off between solution quality and computation time. For all beam search versions, a value of 3 was chosen for both the beam and the filter width parameters. In the four RBS versions, the upper bound weight was set at 0.8, and the API procedure was chosen for the recovering step.

The performance of the alternative dispatching rules that were considered was also analyzed in the preliminary computational tests, to determine the best performing rule. Table 1 presents, for each beam search algorithm, the average of the relative improvements in objective function value over the EDD rule (%imp), as well as the percentage number of times a rule achieves the best objective function value found when compared with the other rules (%best). More precisely, the relative improvement over the EDD rule is calculated as (edd_ofv − rule_ofv)/edd_ofv × 100, where edd_ofv and rule_ofv are the objective function values obtained by the EDD rule and the appropriate rule (SPT_s_j, CS or EQTP), respectively. These values are omitted for the EDD rule, since they would all be necessarily equal to 0.

Table 1. Preliminary results.

var	heur	rule	$n = 25$		$n = 50$		$n = 75$		$n = 100$	
			%imp	%best	%imp	%best	%imp	%best	%imp	%best
L	PBS	EDD	—	5.00	—	5.83	—	5.00	—	1.67
		SPT_s_j	−172.54	30.00	−110.67	30.00	−451.52	31.67	−154.00	29.17
		CS	−0.53	40.83	0.14	41.67	0.26	36.67	0.38	37.50
		EQTP	−0.10	94.17	−0.03	94.17	0.25	95.83	−0.33	95.00
	DBS	EDD	—	23.33	—	7.50	—	6.67	—	5.00
		SPT_s_j	−2.08	35.00	−3.29	33.33	−8.77	35.00	−1.83	30.83
		CS	0.10	57.50	0.30	53.33	0.34	50.00	0.42	49.17
		EQTP	−0.95	90.83	−0.43	92.50	−0.62	93.33	−0.48	93.33
	FBS	EDD	—	20.83	—	5.00	—	5.00	—	4.17
		SPT_s_j	−101.75	30.00	−90.30	30.00	−314.23	31.67	−123.62	29.17
		CS	0.01	45.83	0.17	48.33	0.13	45.00	0.21	42.50
		EQTP	−0.38	94.17	0.16	94.17	−0.07	95.83	−0.11	95.00
	RBS	EDD	—	60.83	—	59.17	—	60.00	—	58.33
		SPT_s_j	−23.42	44.17	−14.05	38.33	−65.02	34.17	−17.85	34.17
		CS	0.07	66.67	0.07	66.67	0.09	59.17	0.08	60.83
		EQTP	−0.21	90.83	−0.07	87.50	−0.65	94.17	−0.75	93.33
H	PBS	EDD	—	3.33	—	1.67	—	3.33	—	2.50
		SPT_s_j	−137.62	17.50	−169.23	13.33	−123.77	12.50	−125.97	18.33
		CS	16.02	33.33	18.47	26.67	19.58	23.33	20.42	26.67
		EQTP	19.07	79.17	23.47	82.50	23.83	87.50	26.12	90.83
	DBS	EDD	—	14.17	—	5.83	—	4.17	—	3.33
		SPT_s_j	−4.21	33.33	−0.28	29.17	0.37	22.50	−0.85	25.00
		CS	5.08	53.33	5.96	36.67	7.28	34.17	7.67	36.67
		EQTP	4.19	60.00	4.01	72.50	5.53	80.00	6.10	79.17
	FBS	EDD	—	6.67	—	4.17	—	0.00	—	0.83
		SPT_s_j	−121.82	21.67	−179.08	21.67	−129.20	20.00	−133.34	21.67
		CS	5.64	30.00	10.15	35.83	12.73	29.17	14.62	30.83
		EQTP	9.99	85.00	14.51	78.33	16.47	87.50	19.86	89.17
	RBS	EDD	—	48.33	—	38.33	—	43.33	—	36.67
		SPT_s_j	−58.00	25.83	−110.13	13.33	−102.29	9.17	−109.15	10.00
		CS	0.70	40.83	1.01	25.00	1.25	16.67	1.41	15.00
		EQTP	2.35	85.83	2.57	67.50	2.56	63.33	2.89	65.00

The SPT_s_j rule provides the best objective function value for a larger percentage of instances than the EDD rule. However, the relative improvement values are quite negative, so the SPT_s_j rule gives, on average, an objective function value that is much larger than the one achieved by the EDD heuristic. The quite negative relative improvement values are essentially due to the inferior performance of the SPT_s_j heuristic for instances with a low tardiness factor (i.e., instances where most jobs will be completed early). Indeed, the SPT_s_j heuristic actually provides better results than the EDD rule for instances with a high tardiness factor, but this is more than offset by a quite poor performance for the low tardiness factor instances. This is to be expected since, as we mentioned earlier, the EDD rule performs better for instances with a larger number of early jobs, while the SPT_s_j heuristic is instead suited to instances where most jobs will be tardy.

For instances with low processing time variability, the objective function values provided by the EDD, CS, and EQTP rules are quite close. Nevertheless, the CS and EQTP rules provide the best results for a larger number of instances. This is particularly clear for the EQTP rule, which provides the best results for over 90% of the test instances. The CS and (especially) the EQTP rules clearly outperform the EDD rule for the high variability instances. In fact, these rules provide a quite significant relative improvement, and also give the best results for a much larger number of instances.

For the high variability instances, the improvement provided by the CS and EQTP rules over the EDD heuristic is higher for the PBS procedure, which relies only on a priority evaluation. Therefore, it certainly seems that a high quality priority function should be used in beam search algorithms. Even though the relative improvement is smaller for the FBS procedure (which uses both priority and detailed evaluations), and also for the DBS algorithm (which uses only a detailed evaluation), the more sophisticated CS and EQTP rules nevertheless still provide a significant improvement. Hence, a good rule should also be used to obtain an upper bound estimate in beam search procedures. The objective function values provided by the several rules are closer for the RBS procedure. This is to be expected, since the RBS algorithm uses a recovering step that corrects previous wrong decisions. Therefore, incorrect choices made previously by an inferior rule can later be corrected, and the several rules then provide results that are much closer.

The EQTP rule is then selected, since it provides the best performance. In fact, this rule not only achieves the best results for a quite large percentage of the test instances, but it also provides a large relative improvement over the EDD heuristic for the instances with a high processing time variability. In the following sections, we will therefore present results only for the beam search versions that use the EQTP rule.

3.3. *Heuristic results*

In this section, we compare the beam search algorithms with the best existing heuristic, namely the EQTP dispatching rule. Table 2 gives the average of the relative improvements in objective function value over the EQTP procedure (%imp), as well as the percentage number of times a heuristic achieves the best result when compared with the other heuristics (%best). The relative improvement over the EQTP heuristic is calculated as $(\text{eqtp\_ofv} - \text{heur\_ofv})/\text{eqtp\_ofv} \times 100$, where heur_ofv and eqtp_ofv are the objective function values of the appropriate heuristic and the EQTP dispatching rule, respectively. The relative improvement values are omitted for the EQTP heuristic, since they are necessarily equal to 0.

The PBS algorithm fails to improve on the EQTP dispatching rule. Indeed, both the objective function values and the percentage of best results are quite close for

Table 2. Heuristic results.

var	heur	$n = 25$		$n = 50$		$n = 100$		$n = 500$	
		%imp	%best	%imp	%best	%imp	%best	%imp	%best
L	EQTP	—	24.42	—	14.83	—	16.33	—	45.25
	PBS	−0.39	24.58	−0.30	14.50	−0.26	16.08	−0.21	43.75
	DBS	0.56	81.50	0.73	88.25	0.57	91.50	—	—
	FBS	0.35	55.33	0.38	48.67	0.08	42.08	−0.05	47.42
	RBS	1.12	73.33	0.99	56.42	0.58	48.50	0.25	99.67
H	EQTP	—	7.17	—	0.75	—	0.33	—	13.67
	PBS	0.04	7.25	−0.07	0.83	−0.05	0.33	−0.04	13.67
	DBS	4.33	44.33	3.73	54.17	3.49	65.58	—	—
	FBS	3.95	35.58	2.24	32.17	1.22	32.83	0.33	42.50
	RBS	5.51	77.42	4.00	42.92	3.00	24.17	2.06	72.33

these two heuristics. The negative average relative improvement values for some instance sizes may seem surprising, since it appears that the PBS algorithm should generate the EQTP sequence, and therefore could not provide results inferior to those of the EQTP dispatching rule. However, the PBS algorithm is only guaranteed to generate the EQTP sequence if there are no ties in the selection of jobs during the various iterations, or if those ties are resolved in the same way. Due to the nature of both the instance data and the EQTP priority index, ties may indeed occur when a job is to be selected for the next position. Also, for computational efficiency concerns, these ties are not guaranteed to be solved identically in the EQTP and PBS heuristics. Therefore, it is possible for the PBS heuristic not to generate the EQTP sequence, and consequently to provide an inferior result.

The DBS, FBS, and RBS procedures provide an improvement over the EQTP dispatching heuristic, particularly for the high variability instances. The best results are given by the RBS and DBS algorithms. The RBS procedure usually provides a higher relative improvement, while the DBS heuristic generally achieves the best results for a larger number of instances.

The FBS algorithm is also superior to the EQTP dispatching rule, but is outperformed by the DBS and RBS procedures. The DBS algorithm performs a thorough evaluation for all nodes, which can explain its superior performance, since the FBS procedure only calculates an upper bound estimate for the nodes that are not eliminated by the filtering step. The RBS heuristic, on the other hand, not only uses a weighted average of both upper and lower bounds in the detailed evaluation step, but also benefits from the recovering step, which uses local search to correct previous wrong decisions.

The relative improvement given by the RBS, DBS, and FBS algorithms is much larger for the high variability instances. Indeed, the improvement provided by the RBS and DBS procedures is about 3–4% for instances with high variability. For the low variability instances, however, the relative improvement is usually below 1%.

Table 3. Relative improvement over the EQTP heuristic, for instances with 50 jobs.

heur	T	Low var				High var			
		$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$
PBS	0.0	0.00	0.00	0.00	−0.01	−0.01	−0.02	−0.01	−0.01
	0.2	−2.98	−3.91	−0.05	−0.07	−1.99	0.09	0.00	0.01
	0.4	0.00	0.00	0.00	−0.21	0.18	0.00	0.02	0.12
	0.6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	1.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DBS	0.0	0.00	0.02	0.05	0.09	0.25	0.56	0.86	1.58
	0.2	4.83	5.26	1.29	1.51	13.54	17.03	12.53	13.77
	0.4	0.03	0.06	0.18	4.12	1.29	1.06	2.70	22.73
	0.6	0.00	0.00	0.00	0.00	0.65	0.20	0.14	0.21
	0.8	0.00	0.00	0.00	0.00	0.34	0.05	0.03	0.03
	1.0	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01
FBS	0.0	0.00	0.01	0.04	0.07	0.29	0.55	0.85	1.28
	0.2	2.47	3.33	0.67	0.66	6.32	10.87	6.41	6.57
	0.4	0.01	0.02	0.07	1.77	1.04	0.80	1.85	15.33
	0.6	0.00	0.00	0.00	0.00	0.56	0.24	0.17	0.27
	0.8	0.00	0.00	0.00	0.00	0.20	0.07	0.03	0.03
	1.0	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01
RBS	0.0	0.00	0.02	0.05	0.09	0.33	0.67	0.98	1.59
	0.2	10.60	8.16	1.01	1.11	24.65	18.10	10.20	11.11
	0.4	0.02	0.03	0.09	2.66	1.36	1.02	2.56	21.42
	0.6	0.00	0.00	0.00	0.00	0.77	0.18	0.11	0.27
	0.8	0.00	0.00	0.00	0.00	0.52	0.07	0.02	0.01
	1.0	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00

In Table 3, we present the effect of the T and R parameters on the relative improvement over the EQTP dispatching rule, for instances with 50 jobs. The improvement given by the beam search algorithms is minor when a larger number of jobs are completed after their dates ($T \geq 0.6$). In fact, when most jobs are tardy ($T = 1.0$), the objective function values are quite close for all the heuristics. The improvement provided by the RBS, DBS, and FBS algorithms, however, is quite significant for instances with $T \leq 0.4$, where a larger proportion of jobs are completed early. Indeed, the relative improvement given by the RBS and DBS algorithms is about 10–20% for some of the parameter combinations with $T = 0.2$ or $T = 0.4$.

The heuristic runtimes (in seconds) are presented in Table 4. The DBS procedure is computationally demanding, and therefore can only be used for small to medium size instances. The FBS and RBS algorithms are faster, and can be applied to larger instances. The PBS procedure is the fastest of the beam search procedures. However, the EQTP dispatching rule is more computationally efficient, and provides results of similar quality.

The RBS or DBS procedures are then recommended for small to medium size instances. For somewhat larger instances, the RBS heuristic is the procedure of

Table 4. Heuristic runtimes (in seconds).

var	heur	$n = 25$	$n = 50$	$n = 100$	$n = 250$	$n = 500$	$n = 750$
L	EQTP	0.000	0.000	0.001	0.004	0.015	0.034
	PBS	0.002	0.007	0.036	0.509	5.408	32.398
	DBS	0.021	0.295	4.553	—	—	—
	FBS	0.004	0.029	0.206	3.369	28.250	—
	RBS	0.006	0.033	0.227	3.473	28.567	—
H	EQTP	0.000	0.000	0.001	0.004	0.016	0.036
	PBS	0.002	0.008	0.047	0.536	5.430	32.747
	DBS	0.022	0.316	4.938	—	—	—
	FBS	0.005	0.031	0.232	3.613	29.943	—
	RBS	0.006	0.035	0.250	3.711	30.282	—

choice, since it provides much better results than the FBS algorithm, and is only slightly more computationally intensive. For quite large instance sizes, the EQTP dispatching rule is the only procedure that can provide results in a reasonable computation time.

3.4. *Comparison with optimum results*

In this section, the heuristic results are compared with the optimum objective function values, for instances with up to 20 jobs. In Table 5, we present the average of the relative deviations from the optimum (%dev), calculated as $(H - O)/O \times 100$, where H and O are the heuristic and the optimum objective function values, respectively. The percentage number of times each heuristic generates an optimum schedule (%opt) is also given.

From Table 5, it can be seen that the heuristics are quite close to the optimum when the variability is low. The RBS procedure, in particular, performs extremely well. In fact, this heuristic provides objective function values that are less than 0.2% above the optimum, and also generates an optimum solution for a quite large

Table 5. Comparison with optimum objective function values.

var	heur	$n = 10$		$n = 15$		$n = 20$	
		%dev	%opt	%dev	%opt	%dev	%opt
L	EQTP	1.78	45.58	2.14	34.50	1.83	28.17
	PBS	1.44	50.33	2.51	35.83	2.33	29.25
	DBS	0.10	89.50	0.45	76.08	0.69	68.08
	FBS	0.22	83.67	0.63	64.67	1.10	60.00
	RBS	0.02	97.00	0.03	83.17	0.13	73.25
H	EQTP	22.14	22.25	16.45	11.92	11.96	8.67
	PBS	17.99	24.08	15.39	12.67	12.20	9.08
	DBS	3.13	52.75	3.54	38.58	3.22	33.17
	FBS	2.73	51.92	2.91	38.08	3.79	32.58
	RBS	0.46	88.83	0.89	75.83	0.81	56.83

number of instances. The DBS and FBS procedures also perform quite well. These heuristics provide an optimum solution for a large number of instances, and their average deviation from the optimum is usually less than 1%. Even the simpler PBS and EQTP procedures perform well, providing results that are about 1–2% above the optimum.

The performance of the heuristics, however, deteriorates when the variability of the processing times increases, particularly for the simpler EQTP and PBS procedures. The RBS procedure still performs quite well for instances with high variability, since its average deviation from the optimum is less than 1%, and it provides an optimum solution for over half of the test instances. The performance of the DBS and FBS procedures is also quite good. Indeed, these procedures give results that are about 3% above the optimum. The EQTP and PBS heuristics, however, perform poorly for the high variability instances, since they are 10–20% above the optimum.

These results are in line with those presented in the previous section for the relative improvement provided by the beam search heuristics. In fact, the relative improvement over the EQTP dispatching heuristic was lower (higher) for the instances with a low (high) variability. We can now see that there was indeed little room for improvement in the low variability instances. When the variability is high, however, the EQTP heuristic performs poorly, and therefore it is possible to obtain a larger relative improvement.

The effect of the T and R parameters on the relative deviation from the optimum is presented in Table 6, for instances with 20 jobs. The heuristics are much closer to the optimum when a larger number of jobs is tardy ($T \geq 0.6$). Actually, when most of the jobs complete after their due dates ($T = 1.0$), the heuristic procedures are usually optimal or nearly optimal. The relative deviation from the optimum is higher for instances with a larger proportion of early jobs (particularly instances with $T = 0.2$ or $T = 0.4$).

4. Conclusion

In this paper, we considered the single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. Several heuristics based on the beam search approach were presented. These algorithms included classic beam search procedures, as well as the filtered and recovering variants. Beam search algorithms require evaluation functions, which are typically provided by dispatching rules. Four dispatching heuristics were considered, so as to analyze the effect of different rules on the performance of the beam search algorithms.

We performed extensive preliminary experiments, to determine adequate values for the parameters required by the several beam search procedures. The performance of the alternative dispatching rules was also analyzed in these initial tests. The results show that using better rules improves the performance of the beam search heuristics.

Table 6. Relative deviation from the optimum for instances with 20 jobs.

heur		Low var				High var			
	T	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$
EQTP	0.0	0.19	0.09	0.08	0.11	0.66	1.64	2.65	2.64
	0.2	17.05	13.56	3.98	2.23	60.07	91.36	26.80	20.54
	0.4	0.10	0.13	0.42	5.92	6.34	4.57	13.49	44.53
	0.6	0.02	0.02	0.01	0.01	3.97	1.38	1.23	1.88
	0.8	0.01	0.01	0.00	0.00	1.68	0.82	0.30	0.25
	1.0	0.00	0.00	0.00	0.00	0.03	0.05	0.05	0.06
PBS	0.0	0.28	0.10	0.11	0.12	0.67	1.64	2.65	2.64
	0.2	20.10	21.02	4.74	2.21	60.38	87.69	39.06	19.12
	0.4	0.10	0.12	0.41	6.62	6.33	4.45	13.39	43.84
	0.6	0.02	0.02	0.01	0.01	3.97	1.38	1.19	1.79
	0.8	0.01	0.01	0.00	0.00	1.12	0.80	0.23	0.25
	1.0	0.00	0.00	0.00	0.00	0.03	0.05	0.05	0.06
DBS	0.0	0.05	0.01	0.01	0.02	0.38	0.87	1.28	1.11
	0.2	8.18	6.48	1.55	0.09	24.13	14.95	5.70	4.13
	0.4	0.03	0.01	0.03	0.18	2.32	0.95	3.54	14.43
	0.6	0.01	0.00	0.00	0.00	1.72	0.26	0.30	0.49
	0.8	0.00	0.00	0.00	0.00	0.28	0.27	0.06	0.08
	1.0	0.00	0.00	0.00	0.00	0.01	0.02	0.01	0.01
FBS	0.0	0.01	0.00	0.01	0.02	0.15	0.26	0.51	0.52
	0.2	11.19	7.91	2.07	0.66	30.45	28.12	5.79	6.56
	0.4	0.02	0.04	0.20	4.23	3.14	1.18	2.39	7.78
	0.6	0.01	0.00	0.00	0.00	2.43	0.27	0.24	0.52
	0.8	0.00	0.00	0.00	0.00	0.43	0.18	0.04	0.07
	1.0	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01
RBS	0.0	0.00	0.00	0.00	0.01	0.04	0.11	0.28	0.15
	0.2	1.86	0.21	0.23	0.27	3.97	4.52	2.31	1.42
	0.4	0.01	0.03	0.08	0.50	0.68	0.34	0.72	3.79
	0.6	0.00	0.00	0.00	0.00	0.67	0.13	0.06	0.04
	0.8	0.00	0.00	0.00	0.00	0.04	0.06	0.00	0.01
	1.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The best performing versions of the beam search algorithms were then compared with the best existing heuristic, as well as with optimal solutions. The best results are given by the RBS and DBS algorithms, and the FBS procedure also provides an improvement over the best existing heuristic. The relative improvement given by the RBS, DBS, and FBS algorithms is much larger for the high variability instances. The several heuristic procedures, particularly the RBS procedure, were quite close to the optimum for instances with low variability. The RBS and DBS algorithms still performed quite well for the high variability instances, giving results that are about 1% and 3% above the optimum, respectively. The EQTP and PBS heuristics, however, perform poorly for these instances.

The RBS or DBS procedures are recommended for small to medium size instances. For somewhat larger instance sizes, the DBS heuristic requires excessive computation times, and the RBS procedure is then the heuristic of choice.

For extremely large instances, however, dispatching rules are the only procedure that can provide results within reasonable computation times.

Acknowledgments

The author would like to thank the anonymous referees and an associate editor for several, and most useful, comments and suggestions that were used to improve this paper.

References

Baker, KR and GD Scudder (1990). Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38, 22–36.

Della Croce, F, M Ghirardi and R Tadei (2004). Recovering beam search: Enhancing the beam search approach for combinatorial problems. *Journal of Heuristics*, 10, 89–104.

Della Croce, F and V T'kindt (2002). A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53, 1275–1280.

Esteve, B, C Aubijoux, A Chartier and V T'kindt (2006). A recovering beam search algorithm for the single machine just-in-time scheduling problem. *European Journal of Operational Research*, 172, 798–813.

Garey, MR, RE Tarjan and GT Wilfong (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, 13, 330–348.

Ghirardi, M and CN Potts (2005). Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165, 457–467.

Gupta, SK and T Sen (1983). Minimizing a quadratic function of job lateness on a single machine. *Engineering Costs and Production Economics*, 7, 187–194.

Hall, N, W Kubiak and S Sethi (1991). Earliness-tardiness scheduling problems, {II}: Deviation of completion times about a restrictive common due date. *Operations Research*, 39, 847–856.

Hoogeveen, H (2005). Multicriteria scheduling. *European Journal of Operational Research*, 167, 592–623.

Kanet, JJ and V Sridharan (2000). Scheduling with inserted idle time: Problem taxonomy and literature review. *Operations Research*, 48, 99–110.

Kim, YD and CA Yano (1994). Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics*, 41, 913–933.

Kubiak, W (1993). Completion time variance minimization on single machine is difficult. *Operations Research Letters*, 14, 49–59.

Lowerre, BT (1976). *The HARPY Speech Recognition System*. PhD Thesis, Carnegie-Mellon University, USA.

Ow, PS and TE Morton (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 26, 35–62.

Ow, PS and TE Morton (1989). The single machine early/tardy problem. *Management Science*, 35, 177–191.

Rubin, S (1978). *The ARGOS Image Understanding System*. PhD Thesis, Carnegie-Mellon University, USA.

Schaller, J (2002). Minimizing the sum of squares lateness on a single machine. *European Journal of Operational Research*, 143, 64–79.

Schaller, J (2004). Single machine scheduling with early and quadratic tardy penalties. *Computers & Industrial Engineering*, 46, 511–532.

Schaller, J (2007). A comparison of lower bounds for the single-machine early/tardy problem. *Computers & Operations Research*, 34, 2279–2292.

Sen, T, P Dileepan and MR Lind (1995). Minimizing a weighted quadratic function of job lateness in the single machine system. *International Journal of Production Economics*, 42, 237–243.

Su, L-H and P-C Chang (1998). A heuristic to minimize a quadratic function of job lateness on a single machine. *International Journal of Production Economics*, 55, 169–175.

Taguchi, G (1986). *Introduction to Quality Engineering*. Asian Productivity Organization, Tokyo, Japan.

Valente, JMS (2007). Heuristics for the single machine scheduling problem with early and quadratic tardy penalties. *European Journal of Industrial Engineering*, 1, 431–448.

Valente, JMS (2008). An exact approach for the single machine scheduling problem with linear early and quadratic tardy penalties. *Asia-Pacific Journal of Operational Research*, 25, 169–186.

Valente, JMS and RAFS Alves (2005). Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. *Computers & Industrial Engineering*, 48, 363–375.

Jorge M. S. Valente is Assistant Professor in the Management Department of the Faculty of Economics, University of Porto (Portugal). He holds a PhD in Management Science from the University of Porto. He has published in *Asia-Pacific Journal of Operational Research*, *Computers & Industrial Engineering*, *Computers & Operations Research*, *International Journal of Production Economics*, *Journal of Manufacturing Systems* and in the *Journal of the Operational Research Society*, among others. His current research interests include production scheduling, heuristic techniques, and agent-based computational economics.