Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 2388–2405
computers & operations research
www.elsevier.com/locate/cor

Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups

Jorge M.S. Valente[a,*], Rui A.F.S. Alves[b]

[a] *LIACC/NIAAD - Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal*
[b] *Faculdade de Economia, Universidade do Porto, Portugal*

Available online 11 January 2007

Abstract

In this paper, we consider the single machine weighted tardiness scheduling problem with sequence-dependent setups. 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. Previous beam search implementations use fixed beam and filter widths. We consider the usual fixed width algorithms, and develop new versions that use variable beam and filter widths.

The computational results show that the beam search versions with a variable width are marginally superior to their fixed value counterparts, even when a lower average number of beam and filter nodes is used. The best results are given by the recovering beam search algorithms. For large problems, however, these procedures require excessive computation times. The priority beam search algorithms are much faster, and can therefore be used for the largest instances.

Scope and purpose

We consider the single machine weighted tardiness scheduling problem with sequence-dependent setups. In the current competitive environment, it is important that companies meet the shipping dates, as failure to do so can result in a significant loss of goodwill. The weighted tardiness criterion is a standard way of measuring compliance with the due dates. Also, the importance of sequence-dependent setups in practical applications has been established in several studies.

In this paper, we present several heuristics based on the beam search technique. In previous beam search implementations, fixed beam and filter widths have been used. We consider the usual fixed width algorithms, and also develop new versions with variable beam and filter widths.

The computational tests show that the beam search versions with a variable width are marginally superior to their fixed value counterparts. The recovering beam search procedures are the heuristic of choice for small and medium size instances, but require excessive computation times for large problems. The priority beam search algorithm is the fastest of the beam search heuristics, and can be used for the largest instances.

Keywords: Scheduling; Weighted tardiness; Sequence-dependent setups; Beam search

1. Introduction

In this paper, we consider the single machine weighted tardiness scheduling problem with sequence-dependent setups. Formally, this problem can be stated as follows. A set of n jobs $\{J_1, J_2, \ldots, J_n\}$ has to be scheduled without

* Corresponding author. Tel.: +351 22 557 11 00; fax: +351 22 550 50 50.
E-mail address: jvalente@fep.up.pt (J.M.S. Valente).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.11.004

preemption on a single machine that can handle at most one job at a time. The machine and the jobs are assumed to be continuously available from time zero onwards. Job J_j, $j = 1, 2, \ldots, n$, requires a processing time p_j, and has a due date d_j and a positive weight or penalty w_j. The weight w_j may represent rush shipping costs that are incurred when an order is overdue, or even a contractual penalty for late delivery. This weight can also be associated with the importance of a specific customer to the company, as well as with a loss of goodwill and future lost sales.

For any given schedule, the tardiness of J_j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of J_j. If job J_j is processed immediately after job J_i, a setup time s_{ij} is incurred. The setup times are therefore sequence-dependent, since they depend on both the job that is to be processed next, and the job that precedes it. If job J_j is processed first, it is assumed that it requires a setup time s_{0j}. The objective is then to find a sequence that minimizes the total weighted tardiness $\sum_{j=1}^{n} w_j T_j$.

In the current competitive environment, it is important that companies meet the shipping dates submitted to their costumers, as failure can result in rush shipping costs, lost sales and a significant loss of goodwill. The weighted tardiness criterion has been a standard way of measuring compliance with the due dates. Several studies have also established the importance of sequence-dependent setups in practical applications. Wilbrecht and Prescott [1] point out that sequence-dependent setups are important when a job shop is operated at or near full capacity. In a survey of industrial managers, Panwalkar et al. [2] found that approximately three quarters of the managers reported that at least some operations required sequence-dependent setup times, while about 15% reported that all the operations they scheduled involved sequence-dependent setups. Wortman [3] emphasized the importance of explicitly considering sequence-dependent setups for an appropriate management of the production capacity.

The single machine weighted tardiness scheduling problem with sequence-dependent setups is strongly NP-hard, since it is a generalization of weighted tardiness scheduling [4]. The problem with sequence-dependent setups has been previously considered by Raman et al. [5] and Lee et al. [6]. Raman et al. presented a dispatching heuristic. Lee et al. proposed a three-phase heuristic solution procedure. In the first phase, some instance statistics are calculated. The second phase uses the Apparent Tardiness Cost with Setups (ATCS) dispatching procedure to schedule the jobs. The parameters required by this dispatching rule are calculated using the results of the first phase. Finally, an improvement procedure is used to improve the schedule obtained in the second phase. The computational results showed that the ATCS heuristic outperformed the dispatching rule proposed by Raman et al. [5]. The ATCS heuristic was adapted for the problem with identical parallel machines by Lee and Pinedo [7].

A survey of machine scheduling problems involving setup considerations is given by Allahverdi et al. [8]. Potts and Kovalyov [9] review the literature on problems that integrate scheduling and batching decisions. This literature review includes some scheduling problems with setup times, more specifically scheduling models with job families. The single machine total weighted tardiness problem with no setups has received considerable attention. A recent survey of the state of the art in weighted and unweighted tardiness scheduling can be found in [10].

In this paper, we present several heuristic algorithms based on the beam search technique. These algorithms include the classic beam search procedures, with both priority and total cost evaluation functions, as well as the more recent filtered and recovering variants. Previous implementations of the beam search approach use fixed beam and filter widths. We consider the usual fixed width algorithms, and also develop new versions that use variable beam and filter widths. In these versions, the number of beam or filter nodes is determined at each level of the search tree, based on the quality of the competing nodes. The proposed algorithms are compared with the three-phase heuristic solution procedure presented by Lee et al. [6].

The remainder of the paper is organized as follows. In Section 2, we describe the beam search approach and its several variations. The fixed and variable width alternatives are discussed in Section 3. The proposed heuristic procedures and their implementation details are presented in Section 4. The computational results are reported in Section 5, and some concluding remarks are given in Section 6.

2. The beam search approach

Beam search is a heuristic method for solving combinatorial optimization problems that consists in an adaptation of the branch-and-bound algorithm (for a description of the branch-and-bound algorithm for scheduling problems, see for instance Brucker [11]). In the beam search method, only the most promising nodes at each level of the search tree are retained for further branching, while the remaining nodes are pruned off permanently. Since only a few nodes are

selected for further branching at each iteration (with the remaining nodes being pruned), the running time of beam search heuristics is polynomial in the problem size.

The beam search approach was first used in the artificial intelligence community for the speech recognition [12] and image understanding [13] problems. Since then, several applications to scheduling problems have appeared in the literature. Fox [14] and Ow and Smith [15] incorporated beam search algorithms in scheduling systems for complex job shop environments. Sabuncuoglu and Bayiz [16] proposed a beam search algorithm for job shop problems with the makespan and mean tardiness objective functions. Ow and Morton [17,18] developed a variation of this technique called filtered beam search, and tested it on the single machine early/tardy problem.

Another variation of the beam search approach, denoted as recovering beam search, was proposed by Della Croce and T'kindt [19] and Della Croce et al. [20]. Della Croce and T'kindt apply this new approach to the single machine completion time problem with release dates, while Della Croce et al. consider both the two-machine total completion time flow shop scheduling problem and the uncapacitated p-median location problem. The recovering beam search approach has since then been applied to several other problems. Valente and Alves [21] presented both recovering and filtered beam search heuristics for the single machine early/tardy problem, while Ghirardi and Potts [22] proposed a recovering beam search algorithm for the problem of scheduling jobs on unrelated parallel machines to minimize the makespan. Recently, Esteve et al. [23] developed a recovering beam search procedure for a just-in-time scheduling problem with multiple criteria. In the next subsections, we describe the classic beam search technique, as well as the more recent filtered and recovering variants.

2.1. Classic beam search

The classic beam search approach consists in a truncated branch-and-bound algorithm where only the most promising β nodes are retained for further branching at each level of the search tree; β is the so-called *beam width*. Since the intent of this technique is to search quickly for a good solution, the other nodes are simply discarded and backtracking is not allowed. This means beam search cannot recover from a wrong decision, and therefore is not guaranteed to find an optimal solution. A wider beam width reduces the risk of eliminating a node that would ultimately lead to the optimal solution, but at the cost of increased computational effort.

The node evaluation process plays a major role in the effectiveness of a beam search algorithm. Two different types of evaluation functions have been used: *priority evaluation functions* and *total cost evaluation functions*. Priority evaluation functions simply calculate a priority or urgency rating, typically by using a dispatching heuristic to calculate a priority index for the last job added to the current partial schedule. Total cost evaluation functions calculate an estimate of the minimum total cost of the best solution that can be obtained from the partial schedule represented by the node. This is usually done by using a dispatching rule to schedule the remaining jobs and complete the existing partial sequence. A priority evaluation function only considers the next decision to be made (i.e., the next job to schedule), and therefore has a local view of the problem. A total cost evaluation function, on the other hand, has a global view of the problem, since it projects from the current partial solution to a complete schedule in order to obtain a cost estimate.

The priority evaluation functions can pose a slight problem. The priority index of the dispatching rules used to calculate the urgency rating of the last scheduled job depends necessarily on the specific scheduling problem under consideration. Also, this priority index is usually a function of the current partial schedule, namely a function of the current time. Different nodes on the same tree level correspond to different partial sequences, and contain a possibly different set of scheduled jobs. Therefore, the completion time of the last scheduled job in the current partial sequence may be different for two nodes, even when those nodes are on the same level of the search tree. The priorities are then context-dependent, and the priorities calculated for the offspring of a node cannot be legitimately compared with those obtained from the expansion of a different node. This problem can be solved by initially selecting the best β children of the root node (i.e., the node containing an empty sequence). At lower levels of the search tree, only the best descendant of each beam node is kept for the next iteration. Total cost evaluation functions are not affected by this problem, since cost estimates are context-independent and can be compared.

We now present the main steps of both priority beam search and detailed beam search algorithms. Priority beam search uses a priority evaluation function, while detailed beam search uses a total cost evaluation function. In the following, B is the set of nodes retained at each level of the search tree for further branching, C is a set of offspring nodes and n_0 is the root node.

Priority beam search:

Step 1: Initialization:
Set $B = \emptyset$, $C = \emptyset$.
Branch n_0 generating the corresponding children.
Calculate the priority of the last scheduled job for each child node.
Select the best β 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 $C = \emptyset$.

Step 3: Stopping condition:
If the nodes in B are leaf (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.

Detailed beam search:

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

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 β child nodes and add them to C.
Set $B = \emptyset$.

Step 3: Node selection:
Select the best β nodes in C and add them to B.
Set $C = \emptyset$.

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.

2.2. Filtered and recovering beam search

The two alternative types of evaluation function used in the classic beam approach have different advantages and weaknesses. The priority evaluation functions are computationally cheap, but are potentially inaccurate, and may lead to the elimination of good nodes. The total cost evaluation functions, on the other hand, provide a more accurate evaluation, but require a much higher computational effort. The filtered and recovering beam search variants use a filtering step and a two-stage approach that takes advantage of both crude and accurate evaluations. These two methods thus try to provide a node evaluation process that gives high quality evaluations without requiring excessive computation times.

The filtered and recovering beam search algorithms first apply a computationally inexpensive filtering procedure in order to select only some of the offspring of each beam node for a more detailed evaluation. The chosen nodes are then evaluated more accurately using a total cost function, and the best β nodes are kept for further branching.

Two different types of filtering procedure have been previously used. Ow and Morton [17,18] proposed a filtering procedure that uses a priority evaluation function to calculate an urgency rating for each offspring. The best α children of each beam node are then chosen for the detailed evaluation step, where α is the so-called *filter width*. Recently,

Della Croce and T'kindt [19] and Della Croce et al. [20] introduced a new type of filtering procedure in conjunction with the development of the recovering beam search algorithm. This new procedure uses problem-specific properties to determine, with a reduced computational effort, the nodes that advance to the detailed evaluation step. More specifically, problem dependent dominance conditions, denoted as valid dominance conditions, when available for the problem under consideration, are applied together with so-called pseudo-dominance conditions, which hold in a heuristic context only. Whenever a valid dominance condition or a pseudo-dominance condition applies for a given node, that node is then eliminated from further consideration.

The priority function approach was then originally applied in filtered beam search algorithms, while the dominance conditions filtering procedure was developed for the recovering beam search heuristic. Nevertheless, either of these two filtering procedures can be used in both filtered and recovering algorithms. Therefore, it is possible to apply a dominance conditions procedure in a filtered beam search approach, and the priority function filter can also be applied in recovering beam search algorithms.

The recovering beam search procedure differs from the filtered beam search approach in two major ways. First, in filtered beam search the accurate evaluation step uses an upper bound on the total cost of the best solution that can be obtained from the partial sequence represented by the node. In the recovering variant, on the other hand, the accurate node evaluation process uses both lower and upper bounds. These bounds are used in an evaluation function $V = (1-\gamma)LB + \gamma UB$, where $0 \leqslant \gamma \leqslant 1$ is a user-defined parameter. The evaluation function V is then a weighted sum of the lower and upper bounds, with the weight of the upper bound being given by the parameter γ.

Second, an additional *recovering step* is present in the recovering beam search approach. This step is applied on the best β nodes retained after the detailed evaluation step at each level of the search tree. The recovering step typically consists in applying neighbourhood operators to check whether a current partial solution σ is dominated by another partial solution $\overline{\sigma}$ having the same level of the search tree. If a better partial solution $\overline{\sigma}$ does exist, then $\overline{\sigma}$ becomes the new current partial solution.

The recovering step often allows the recovering beam search procedure to recover from previous wrong decisions. This feature is not available in the classic and filtered beam search variants, which cannot recover from a wrong decision: if a branch leading to the optimal solution is pruned, there is no way to reach afterwards that solution. The recovering step in the recovering beam search approach seeks to overcome this issue by searching for improved partial solutions dominating those selected by the beam. Since the recovering step only replaces a partial solution with another partial solution with the same depth of the search tree, the total number of explored nodes is still polynomial in the problem size. We now present the main steps of both filtered and recovering beam search algorithms.

Filtered beam search:

Step 1: Initialization:
Set $C = \emptyset$ and $B = \{n_0\}$.
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 $B = \emptyset$.
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 $C = \emptyset$.
Step 4: Stopping condition:
If the nodes in B are leaf, select the node with the lowest total cost as the best sequencefound and stop.
Otherwise, go to step 2.

Recovering beam search:

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

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 $B = \emptyset$.

Step 3: Node selection:
For all nodes in C:
(a) Calculate a lower bound LB and an upper bound UB on the optimal solution value of that node.
(b) Compute the evaluation function $V = (1 - \gamma)LB + \gamma UB$.
Select the best β nodes in C and add them to B.
Set $C = \emptyset$.

Step 4: Recovering step:
For each node in B:
(a) Let σ represent the partial solution associated with the current node.
(b) Search for a partial solution $\overline{\sigma}$ that dominates σ by means of neighbourhood operators.
(c) If $\overline{\sigma}$ is found, set $\sigma = \overline{\sigma}$.

Step 5: 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.

3. Beam search with a variable width

3.1. Motivation

In previous implementations of beam search procedures, fixed beam and filter widths have been used. Usually, preliminary tests are first performed in order to determine appropriate values for these parameters. In fact, there is a trade-off between solution quality and computation time, since increasing the beam or filter width usually improves the objective function value (ofv), but at the cost of an increased computational effort. The computation time usually increases linearly with the beam and filter width, while the solution quality improves, but with diminishing returns (i.e., the improvement in the solution quality becomes smaller as the parameters increase). Therefore, computational tests are first performed to determine beam and filter width values that provide a good balance between solution quality and computational effort.

The beam and filter width values determined in these preliminary tests are then used throughout the beam search procedure. This means that the chosen parameter values are used for all instances. More importantly, for each specific instance the same beam and filter width values are used at all levels of the search tree.

In this paper, we propose a new implementation for the beam search procedure that can be used in the classic versions, as well as in the filtered and recovering variants. This new implementation uses variable beam and filter widths. Therefore, the number of beam or filter nodes is not necessarily constant in these new versions, and may change from one iteration to another.

The new implementation was motivated by the fact that different beam or filter width values may be appropriate at different levels of the search tree. In fact, a smaller or a larger width may be preferred at each iteration, according to the quality of the currently competing nodes, measured by their evaluation values. At each level of the search tree, the new versions then select the nodes that are close to the best, that is, the nodes whose evaluation values are close to the evaluation value of the best node.

In the previous implementations, on the other hand, a fixed number of nodes is always selected. Therefore, it is possible that some of the selected nodes are quite inferior to the best node. Also, high quality nodes may be discarded, since only a fixed number of nodes is chosen, even when there are additional nodes that are rather close in quality to the best node. As an example, assume a fixed beam width of 3 was selected, and consider the two scenarios given in Table 1. This table gives hypothetical total cost evaluation values (i.e., upper bounds) for the five best nodes, along with the worst, at a given tree level. In scenario A, the upper bound for the 3rd best node is clearly much higher than the values obtained for the two best nodes, so a beam width of 2 would likely be sufficient. On the contrary, a beam

Table 1
Variable width example

Scenario	Best	2nd	3rd	4th	5th	…	Worst
A	100	102	150	160	165		200
B	100	102	105	106	150		200

width of 4 might be a better choice in scenario B. In this scenario, the upper bound for the 3rd and 4th best nodes are nearly identical, so the 4th node could indeed lead to better solutions than the 3rd best.

3.2. Implementation details

We now present the implementation details of the beam search versions with a variable width. First, we formally describe the procedure used to select beam nodes. Let *best* and *worst* denote the evaluation values of the best and worst nodes, respectively. Also, let $0 \leqslant b_{\text{dev}} \leqslant 1$ be the (user-defined) beam relative deviation parameter. The beam threshold evaluation value T_{b} is then calculated as:

$$T_{\text{b}} = best + b_{\text{dev}} \times (worst - best).$$

The nodes with an evaluation value not larger than T_{b} are then selected. However, due to both computational efficiency and solution quality concerns, we also impose user-defined minimum and maximum limits on the number of selected nodes. Let $b_{\min}$ and $b_{\max}$ denote the minimum and maximum beam width values, respectively. Then, if the number of nodes that satisfies the beam threshold value T_{b} is lower than $b_{\min}$, the best $b_{\min}$ nodes are selected. Similarly, if the number of nodes within the threshold exceeds $b_{\max}$, only the best $b_{\max}$ nodes are chosen. The previous discussion, as well as the expression given for T_{b}, assumes that we are dealing with a minimization problem and a total cost evaluation function. In this case, the best nodes are those with a low evaluation value (i.e., a low cost). The adaptation to maximization problems, and/or priority evaluation functions, is straightforward.

The procedure used to select filter nodes is quite similar. Again, let *best* and *worst* denote the evaluation values of the best and worst nodes, respectively. Also, let $0 \leqslant f_{\text{dev}} \leqslant 1$ be the (user-defined) filter relative deviation parameter. The filter threshold evaluation value T_f is then calculated as follows:

$$T_f = best - f_{\text{dev}} \times (best - worst).$$

The nodes with an evaluation value not lower than T_f are then selected. Once more, we impose user-defined minimum ($f_{\min}$) and maximum ($f_{\max}$) limits on the number of selected nodes. Therefore, the number of chosen nodes is never lower than $f_{\min}$, or greater than $f_{\max}$. The previous discussion, as well as the expression given for T_f, assumes that the evaluation function assigns larger evaluation values to the more urgent jobs, as is usually the case with priority evaluation functions. The adaptation to other evaluation functions is straightforward.

Beam search versions with a variable width have a significantly larger set of available choices in the trade-off between solution quality and computation time. For instance, if a fixed beam width of 3 is currently used, and a larger beam is being considered, the closest choice is a beam width of 4. If a variable width is used instead, a wide number of intermediate choices is available by appropriately setting the relative deviation parameter. In fact, the average beam width (over all search tree levels) can be changed smoothly between three and four by increasing the maximum allowed relative deviation.

4. The proposed heuristic procedures

In this section, we describe the implementation details concerning the application of the beam search algorithms to the specific single-machine weighted tardiness problem with sequence-dependent setups. We first present the lower bounding method that is used in the recovering beam search algorithms. The improvement procedures that were considered for the recovering step in the recovering beam search heuristic are then described. Finally, we provide additional implementation details concerning the proposed beam search algorithms.

4.1. The lower bounding procedure

The detailed evaluation step in the recovering beam search procedure requires a lower bound on the total cost of the best solution that can be obtained from the partial schedule represented by the node. We now describe the procedure that is used to calculate a lower bound for the unscheduled jobs at each node. This procedure relaxes the sequence-dependent setups and modifies the jobs' processing times, converting the problem into a total weighted tardiness problem with no setups. A lower bounding procedure for the problem with no setup can then be applied to the modified data.

Let Q represent the current partial sequence of a given node at tree level l, and let $Q(k)$ be the kth job in this sequence. The modified processing times of the unscheduled jobs are then given by

$$p'_j = \min\left\{\min_{i \notin Q}\{s_{ij}\}, s_{Q(l)j}\right\} + p_j.$$

Since the minimum possible setup time is assumed for each job, any of the lower bounding procedures available for the problem with no setup can then be applied to the modified data, in order to obtain a lower bound for the original problem. The lower bounding procedure proposed by Potts and Van Wassenhove [24] was chosen, since it provides adequate results and is also computationally efficient.

4.2. Improvement procedures

The recovering step in the recovering beam search algorithm requires an improvement or neighbourhood search procedure. We considered three simple improvement procedures: adjacent pairwise interchange (API), 3-swaps (3SWAP) and the insertion method (INS) proposed by Lee et al. [6].

The API procedure, at each iteration, considers in succession all adjacent job positions. A pair of adjacent jobs is then swapped if such an interchange improves the ofv. This process is repeated until no improvement is found in a complete iteration (i.e., until the sequence is locally optimal and cannot be further improved by adjacent swaps).

The 3SWAP 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 analysed, and the best configuration is selected. Once more, the procedure is applied repeatedly until no improvement is possible.

The INS method selects at each iteration the job with the largest weighted tardiness. The selected job is then inserted after the nearest $\lceil n/3 \rceil$ jobs. The best insertion in then performed if it improves the ofv. This process is repeated until no improving move is found.

4.3. Beam search implementation details

The implementation details of the proposed beam search procedures are presented in this section. Both priority and detailed classic beam search algorithms were considered, as well as filtered and recovering beam search procedures. For each of these algorithms, we developed versions with fixed and variable widths. From now on, the priority and detailed classic beam search algorithms will be identified as PBS and DBS. Similarly, the filtered and recovering procedures will be denoted as FBS and RBS. The versions with fixed and variable widths will be identified by appending _F and _V, respectively, to the beam search procedures identifiers.

In order to apply these algorithms to the specific weighted tardiness problem with sequence-dependent setups, it is necessary to specify their main components, such as branching scheme, evaluation function, filtering procedure and recovering step. In the following, we first describe the branching scheme, which is common to all the algorithms. Then, we provide the remaining implementations details for each type of algorithm.

Branching scheme: The same branching procedure is used for all algorithms. The branching scheme consists in the usual n-ary forward branching: 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.

Priority beam search: The PBS algorithms require a priority evaluation function. This function is provided by the priority index of the ATCS dispatching rule. The evaluation value of a node is therefore obtained by calculating the ATCS priority index of the last scheduled job in that node.

Detailed beam search: The detailed beam search algorithms require a total cost evaluation function, i.e., an upper bounding procedure. This upper bounding procedure is provided by the ATCS dispatching heuristic. For a given node, the ATCS rule is used to sequence the remaining unscheduled jobs, therefore completing the existing partial schedule. The evaluation value of the node is then equal to the cost of the complete schedule obtained with the ATCS heuristic.

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

Recovering beam search: The RBS algorithms require a filtering procedure, upper and lower bounding procedures for the detailed 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 bound procedure is provided by the method presented in Section 4.1. For a given node, this procedure is used to calculate a lower bound for the remaining 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 calculated for the remaining unscheduled jobs. For the recovering step, we considered the API, 3SWAP and INS procedures described in Section 4.2. We performed preliminary computational tests to determine an adequate choice for the improvement procedure. The API method was chosen, since it provided the best balance between computation time and solution quality.

Early termination: The DBS, FBS and RBS algorithms use an upper bounding procedure, as mentioned above. For a given node, this procedure provides an upper bound on the total cost of the best solution that can be reached from that node. Therefore, if the upper bound of a node is equal to 0, it is possible to immediately terminate the algorithm. In fact, the current partial schedule of the node with an upper bound of 0 can be completed by scheduling the remaining unscheduled jobs with the upper bounding procedure. This yields a solution with a cost equal to 0, and therefore optimal.

Improvement step: The beam search algorithms are compared with the three-phase heuristic solution procedure presented by Lee et al. [6]. This procedure initially calculates some instance statistics, and then uses the ATCS dispatching heuristic to schedule the jobs. Finally, the INS improvement procedure is applied to improve the schedule obtained by the ATCS rule. For simplicity, this three-phase procedure will be simply denoted as ATCS. The INS improvement procedure was also applied, as an improvement step, to the beam search algorithms. Therefore, once the beam search heuristics have generated a solution, the INS procedure in then used to improve that solution.

5. Computational results

In this section, we first describe the set of test problems and the preliminary experiments that were performed to determine adequate values for the parameters used by the several algorithms. Then, we present the computational results and compare the proposed beam search procedures. Throughout this section, and in order to avoid excessively large tables, we will sometimes present results only for some representative cases.

5.1. Experimental design

The computational tests were performed on a set of problems with 15, 20, 25, 30, 40, 50, 75, 100, 250 and 500 jobs. These problems were randomly generated using a scheme similar to the one adopted by Lee et al. [6]. For each job J_j, an integer processing time p_j and an integer penalty w_j were generated from the uniform distributions [50, 150] and [1, 10], respectively. The integer setup times s_{ij} were obtained from the uniform distribution $[0, 2\overline{s}]$, where $\overline{s}$ is the average setup time. The average setup time is set at $\eta\overline{p}$, where η is the setup time severity factor, set at 0.50, 1.00, 1.50 and 2.00, and $\overline{p}$ is the average processing time.

The integer due dates d_j were generated from the uniform distribution $[\widehat{C}_{\max}(1-\tau-R/2), \widehat{C}_{\max}(1-\tau+R/2)]$, where $\widehat{C}_{\max}$ is a makespan estimate, τ is the tardiness factor and R is the due date range factor. Both τ and R were set at 0.1, 0.3, 0.5, 0.7 and 0.9. The makespan clearly depends on the schedule, due to the existence of sequence-dependent setup times. Therefore, the makespan can only be calculated once a specific sequence is determined. Consequently, a makespan estimate $\widehat{C}_{\max}$ is required for the uniform distribution that is used to generate the due dates. Following [6,7], the makespan was estimated as $\widehat{C}_{\max} = n(\overline{p} + \delta\overline{s})$, with $\delta \leqslant 1$. The makespan, as well as its estimate, consists of two

parts: the processing time of the jobs and the setup times between the jobs. Since every job must be processed once, $n\overline{p}$ represents the processing time component of the makespan. By setting $\delta \leqslant 1$, we assume that the setup time component of the makespan is usually much smaller than $n\overline{s}$, since during the generation of a schedule a job with a smaller setup time is more likely to be selected. The value of δ is calculated using the function $\delta = 0.4 + 10/n^2 - \eta/7$, once more following [6,7].

Different problem sets were used for the preliminary experiments and for obtaining the computational results. The set used for comparing the procedures included 50 instances for each parameter combination, yielding 5000 instances for each problem size. A smaller set, with five instances for each combination, was used in the parameter adjustment tests. Due to the large computation times that would be required, the DBS algorithms were only used to solve instances with up to 100 jobs, while the FBS and RBS procedures were not applied to the 500 job instances. All the algorithms were coded in Visual C + + 6.0 and executed on a Athlon 64 3000+ personal computer.

5.2. Preliminary tests

Extensive computational tests were first performed to determine appropriate values for the parameters used by several algorithms. As mentioned before, there is a trade-off between solution quality and computation time, since increasing the parameter values usually improves the ofv, at the cost of increased computational effort (the upper bound weight parameter γ is an exception). For the fixed width versions, the following values were considered for the several parameters:

$$\alpha = \{1, 2, \ldots, 10\},$$
$$\beta = \{1, 2, \ldots, 8\},$$
$$\gamma = \{0.1, 0.2, \ldots, 0.9\}.$$

The fixed width algorithms were applied to the test instances for all combinations of the relevant parameter values. The mean ofv's and runtimes were then calculated and plotted. A thorough analysis of these charts, as well as the underlying data, revealed the usual behaviour: the computation time increased linearly with the beam and filter width, while the solution quality improved, but with diminishing returns. The parameter values that seemed to provide the best trade-off between solution quality and computation time were then chosen after a detailed analysis of these results. The beam and filter width were set at 3 for all fixed width algorithms, while a value of 0.5 was chosen for the upper bound weight in the RBS procedures.

Preliminary tests were also performed to set the parameters used in the variable width versions. We considered values of 1 and 2 for the minimum beam and filter widths, while for the maximum width we tested the values 4, 5 and 6. For the beam and filter width relative deviation parameters, we considered a different set of values for each instance size. These values were selected in order to generate a range of average beam and filter widths that encompassed the chosen fixed value of 3. The mean ofv's and runtimes, as well as the average number of nodes, were calculated and analysed. We then selected the values that not only yielded an average width usually quite close to (and sometimes even lower than) the fixed values of 3, but also provided a good balance between the solution quality and computation time. This would allow a better first comparison between fixed and variable width versions, since similar average widths are usually used in both, while at the same time taking into account the trade-off between ofv and runtime. The minimum and maximum beam widths were set at 2 and 4, respectively, for all algorithms. The filter width minimum and maximum values were set at 1 and 5 for both the FBS and RBS procedures, while a value of 0.5 was chosen for the upper bound weight in the RBS algorithms. The values chosen for the beam and filter relative deviation parameters were different for each instance size, and are given in Table 2.

5.3. Beam search results

We now present the computational results for the heuristic procedures. In Table 3, we present the average ofv for each heuristic, as well as the percentage number of times a heuristic produces the best result when compared with the other heuristics (%best). We also give the relative improvement in the ofv over the ATCS procedure (%imp), calculated as $(\text{ATCS} - \text{Heur})/\text{ATCS} \times 100$, where ATCS and Heur are the average ofv's of the ATCS rule and the appropriate heuristic, respectively. The number of times the variable width algorithms perform better (<), equal (=) or worse (>)

Table 2
Beam and filter relative deviation parameter values

n	Beam				Filter	
	DBS	FBS	PBS	RBS	FBS	RBS
15	0.115	0.3	0.91	0.3	0.96	0.95
20	0.105	0.3	0.90	0.3	0.86	0.90
25	0.095	0.3	0.85	0.3	0.83	0.82
30	0.095	0.3	0.83	0.3	0.81	0.80
40	0.080	0.3	0.82	0.3	0.75	0.75
50	0.080	0.3	0.75	0.3	0.73	0.72
75	0.070	0.3	0.70	0.3	0.68	0.67
100	0.065	0.3	0.73	0.3	0.62	0.60
250	–	0.3	0.63	0.3	0.50	0.53
500	–	–	0.58	–	–	–

Table 3
Heuristic results

Heur.		$n = 15$	$n = 25$	$n = 50$	$n = 75$	$n = 100$	$n = 250$
ATCS	ofv	22798	50233	155141	313169	520814	2841684
	%best	8.20	9.92	17.70	22.28	24.68	28.92
DBS_F	ofv	19584	43968	140642	289026	487173	–
	%best	51.66	35.68	32.46	32.96	33.00	–
	%imp	14.10	12.47	9.35	7.71	6.46	–
DBS_V	ofv	19546	43790	140053	288184	485792	–
	%best	54.90	40.04	35.86	35.48	35.22	–
	%imp	14.26	12.83	9.73	7.98	6.72	–
FBS_F	ofv	19559	43831	140090	287366	483626	2734793
	%best	52.24	38.64	35.14	36.38	37.94	40.04
	%imp	14.21	12.75	9.70	8.24	7.14	3.76
FBS_V	ofv	19520	43691	139250	286186	481770	2725027
	%best	51.72	37.46	39.16	40.24	41.36	43.00
	%imp	14.38	13.02	10.24	8.62	7.50	4.11
PBS_F	ofv	21589	48091	151034	306371	511447	2812278
	%best	11.12	11.48	19.04	23.16	25.38	29.78
	%imp	5.30	4.27	2.65	2.17	1.80	1.03
PBS_V	ofv	21550	47993	150913	306260	511018	2811313
	%best	10.96	11.18	18.88	22.98	25.28	29.78
	%imp	5.47	4.46	2.73	2.21	1.88	1.07
RBS_F	ofv	19537	43689	139448	286394	481140	2711569
	%best	53.24	42.38	39.28	39.94	42.14	48.58
	%imp	14.30	13.03	10.12	8.55	7.62	4.58
RBS_V	ofv	19486	43566	139149	285532	481156	2711348
	%best	55.38	43.94	42.94	45.90	47.76	56.38
	%imp	14.53	13.27	10.31	8.82	7.61	4.59

than the corresponding fixed width versions is given in Table 4. In Table 5, we present the average beam and filter width values for the beam search versions with a variable width. We also performed a test to determine if the difference between the fixed and variable width versions is statistically significant. Given that the heuristics were used on exactly the same problems, a paired-samples test is appropriate. Since not all the hypothesis of the paired-samples t-test were met, the non-parametric Wilcoxon test was selected. The significance values of this test, i.e., the confidence level values

Table 4
Comparison of beam search objective function values

n	DBS			FBS			PBS			RBS		
	<	=	>	<	=	>	<	=	>	<	=	>
15	16.4	73.5	10.0	27.6	45.2	27.1	7.4	87.2	5.4	27.0	48.7	24.3
20	22.7	64.1	13.3	33.3	33.4	33.3	8.5	85.9	5.7	33.0	39.0	28.1
25	27.1	56.5	16.4	37.0	28.9	34.1	9.1	84.6	6.3	35.6	31.3	33.1
30	29.8	50.6	19.5	39.5	26.5	34.0	9.4	84.8	5.9	39.1	27.4	33.5
40	34.6	40.3	25.1	41.7	23.9	34.4	9.1	85.0	5.9	40.0	24.5	35.6
50	38.2	35.4	26.3	43.4	24.1	32.5	8.7	84.8	6.5	40.5	24.4	35.0
75	39.7	29.1	31.2	41.3	25.5	33.2	7.9	85.6	6.5	41.0	25.3	33.7
100	40.7	28.1	31.1	40.3	27.0	32.7	9.5	85.5	5.1	38.4	27.0	34.6
250	–	–	–	40.6	29.6	29.8	9.7	83.4	6.9	37.5	29.6	32.9
500	–	–	–	–	–	–	11.4	79.3	9.3	–	–	–

Table 5
Average beam and filter width values

n	Beam				Filter	
	DBS	FBS	PBS	RBS	FBS	RBS
15	3.07	3.10	3.23	2.98	3.08	3.04
20	3.15	2.93	3.28	2.98	2.67	3.00
25	3.15	2.99	3.18	2.86	2.75	2.77
30	3.22	3.05	3.16	2.92	2.84	2.85
40	3.12	3.05	3.21	2.95	2.81	2.88
50	3.19	3.12	3.06	2.98	2.93	2.94
75	3.12	3.17	2.98	3.03	3.03	3.03
100	3.08	3.13	3.12	2.97	2.96	2.91
250	–	3.12	3.02	3.11	3.04	3.20
500	–	–	2.97	–	–	–

above which the equal distribution hypothesis is to be rejected, were nearly always equal to 0.000, and they were never larger than 0.01.

The beam search versions with a variable width are marginally superior to their fixed value counterparts. The variable width versions provide a lower ofv, and they give better results for a larger number of test instances. The Wilcoxon test values also indicate that the difference in distribution between the fixed and variable width implementations is statistically significant. The superior performance of the variable width algorithms was achieved with an average beam width that is quite close to the value used in the fixed width versions, while the average filter width is nearly always lower than the corresponding fixed value (we recall that both the beam and filter widths were set at 3 in the fixed width versions). The preliminary tests described in the previous subsection also showed that the variable width versions are still superior to the fixed width alternative even if somewhat lower average beam and filter widths were used. Therefore, the variable width versions provide marginally better results, even when they use lower average beam and filter widths.

The proposed beam search algorithms outperform the ATCS dispatching heuristic. The best results are given by the recovering beam search algorithms, followed by the FBS and DBS procedures. This may seem somewhat surprising, since DBS algorithms perform a detailed evaluation for all nodes, while recovering and filtered beam search procedures use a two-stage evaluation and only the nodes that pass the filtering procedure are accurately evaluated. The two-stage evaluation procedure used in the FBS and RBS algorithms, however, can benefit from a synergy between the priority and total cost evaluations. In fact, the filtering procedure can eliminate poor nodes that would erroneously lead to good detailed evaluation values. The early elimination of these nodes can therefore improve the performance and lead to better results than those that would be obtained if only a detailed evaluation was performed.

The recovering beam search algorithms also perform a more accurate detailed evaluation than either the DBS or FBS procedures, since RBS uses both lower and upper bounds in the detailed evaluation step. Moreover, at each iteration

Table 6
Relative improvement over the ATCS rule for the RBS_V heuristic and 50 job instances

η	τ	$R = 0.1$	$R = 0.3$	$R = 0.5$	$R = 0.7$	$R = 0.9$
0.50	0.1	48.50	100.00	–	–	–
	0.3	23.28	27.87	41.21	41.41	77.28
	0.5	12.89	18.16	21.15	17.53	5.25
	0.7	7.35	9.29	4.63	4.01	2.35
	0.9	3.40	3.78	3.22	3.23	2.73
1.00	0.1	100.00	100.00	100.00	–	–
	0.3	34.98	37.16	62.92	76.13	51.93
	0.5	17.62	22.78	31.59	22.79	11.41
	0.7	10.48	13.16	13.59	11.32	7.16
	0.9	5.26	6.11	5.40	5.16	5.40
1.50	0.1	85.12	100.00	100.00	100.00	100.00
	0.3	34.36	37.16	50.78	56.95	55.67
	0.5	19.11	26.28	27.37	28.81	19.73
	0.7	12.73	14.50	17.71	15.82	14.36
	0.9	7.39	7.63	6.94	7.41	6.91
2.00	0.1	64.08	82.87	75.11	97.61	99.99
	0.3	36.61	35.83	42.29	41.67	38.13
	0.5	21.56	26.67	30.13	31.55	23.98
	0.7	14.19	18.03	17.30	15.45	12.31
	0.9	7.36	8.12	7.61	7.30	8.90

the RBS algorithms can improve the current partial solutions during the recovering step. Therefore, the superior performance of the RBS algorithms when compared with the FBS procedures can then be explained by the more accurate detailed node evaluation, on the one hand, and by the local improvement performed in the recovering step, on the other hand. The PBS algorithms, which use only priority evaluations, cannot match the performance of the DBS and FBS procedures, but they still improve upon the ATCS results. The relative improvement over the ATCS procedure decreases with the instance size. For the RBS, FBS and DBS procedures, this improvement is quite large (over 10%) for the smaller instances, and even for problems with 75 and 100 jobs these algorithms give a substantial improvement of about 7–9%.

In Table 6, we present the effect of the τ, R and η parameters on the relative improvement over the ATCS procedure for the RBS_V heuristic and instances with 50 jobs. The relative improvement over the ATCS heuristic increases as the tardiness factor τ decreases. Some of the relative improvement values, however, are somewhat misleading. When $\tau = 0.1$ or 0.3, most jobs will be early, and the average weighted tardiness is low. Therefore, the quite large relative improvement values that are given for some parameter combinations with $\tau \leqslant 0.3$ are deceptive, since they correspond to situations where the variation in the average weighted tardiness is small when measured in absolute terms. For some parameter combinations, the average weighted tardiness is actually 0 for both the ATCS and the beam search procedures, thus making the calculation of the relative improvement impossible. A—sign is used to identify those parameter combinations in Table 6.

As τ increases, more jobs will be tardy, and the average weighted tardiness is high, so a positive relative improvement then represents a significant absolute reduction in the ofv. The high relative improvement values for $\tau = 0.5$ or 0.7 therefore mean that the beam search algorithms provide substantial savings over the dispatching heuristic. The relative improvement also increases with the setup time severity factor η.

The heuristic runtimes (in s) are given in Table 7. The runtimes are similar for the fixed and variable beam width versions, since the average widths in the variable versions were set close to the value selected for the fixed value implementations. The DBS algorithms are computationally quite demanding, and can only be used for small or medium size instances. The RBS procedures are a superior alternative, since they not only provide better results, but are also faster and can be applied to somewhat larger instances. The PBS procedure is the fastest of the beam search algorithms, and is therefore an alternative to the ATCS dispatching rule for the largest instances.

Table 7
Heuristic runtimes (in s)

Heur.	$n = 75$	$n = 100$	$n = 250$	$n = 500$
ATCS	0.001	0.001	0.007	0.032
DBS_F	1.722	5.337	–	–
DBS_V	1.933	6.059	–	–
FBS_F	0.107	0.246	3.924	–
FBS_V	0.128	0.281	4.313	–
PBS_F	0.019	0.043	0.728	8.443
PBS_V	0.020	0.043	0.715	7.643
RBS_F	0.130	0.294	4.488	–
RBS_V	0.152	0.320	5.143	–

6. Conclusion

In this paper, we considered the single machine weighted tardiness scheduling problem with sequence-dependent setups, and presented heuristic algorithms based on the beam search technique. These algorithms include the classic beam search procedures, as well as the filtered and recovering variants. Previous implementations of the beam search approach use fixed beam and filter widths. We considered the usual fixed width procedures, and also developed new versions that use variable beam and filter widths. The proposed algorithms were compared with a heuristic procedure presented in [6].

The beam search versions with a variable width provide marginally better results than their fixed value counterparts. This superior performance is achieved even when they use a lower average number of beam and filter nodes. Also, the beam search versions with a variable width provide a much larger set of available choices in the trade-off between solution quality and computation times. The new versions therefore seem promising and appear to be a good alternative to the current fixed width implementations. As a possible step for future research, it seems worthy to investigate their behaviour on other problems.

The best results are given by the recovering beam search algorithms, followed by the filtered beam search procedures. The priority beam search algorithms cannot match the solution quality of the other beam search heuristics, but are superior to the dispatching heuristic. The RBS procedures are the heuristic of choice for the small and medium size instances, but they require excessive computation times for large problems. The PBS procedure is the fastest of the beam search algorithms, and can then be used for the largest instances.

Acknowledgement

The authors would like to thank an anonymous referee for several, and most useful, comments and suggestions that were used to improve this paper.

Appendix A

A.1. Introduction

Following the acceptance of this article for publication, two additional papers [25,26] proposing heuristic procedures for the single machine total weighted tardiness scheduling problem with sequence-dependent setups have come to our attention. Cicirello and Smith [25] consider four improvement-type algorithms, namely limited discrepancy search (LDS), heuristic-biased stochastic sampling (HBSS), value-biased stochastic sampling (VBSS) and hill-climbing using VBSS (VBSS-HC), as well as a simulated annealing algorithm. These procedures were tested on a proposed set of 120 benchmark problem instances. Liao and Juan [26] present an ant colony optimization (ACO) algorithm. This algorithm was tested on the problem set proposed in [25], and compared with the best-known solutions achieved by the Cicirello and Smith procedures. The computational tests show that the ACO algorithm performs better than the existing procedures.

Table A1
Heuristic solutions for the benchmark instances

#	CCRL	ACO	RBS	#	CCRL	ACO	RBS	#	CCRL	ACO	RBS
1	978	894	1760	41	73176	73578	77081	81	387148	387866	389607
2	6489	6307	8065	42	61859	60914	64839	82	413488	413181	412206
3	2348	2003	2810	43	149990	149670	154046	83	466070	464443	460946
4	8311	8003	12128	44	38726	37390	41771	84	331659	330714	330176
5	5606	5215	6844	45	62760	62535	65102	85	558556	562083	560892
6	8244	5788	11282	46	37992	38779	40505	86	365783	365199	365297
7	4347	4150	5419	47	77189	76011	83659	87	403016	401535	407977
8	327	159	628	48	68920	68852	75240	88	436855	436925	437999
9	7598	7490	8058	49	84143	81530	87273	89	416916	412359	417407
10	2451	2345	2745	50	36235	35507	36869	90	406939	404105	405445
11	5263	5093	9306	51	58574	55794	61763	91	347175	345421	341196
12	0	0	0	52	105367	105203	104463	92	365779	365217	364758
13	6147	5962	8542	53	95452	96218	106848	93	410462	412986	410520
14	3941	4035	5968	54	123558	124132	124848	94	336299	335550	336118
15	2915	2823	4375	55	76368	74469	88827	95	527909	526916	533909
16	6711	6153	8961	56	88420	87474	94040	96	464403	461484	466019
17	462	443	844	57	70414	67447	74734	97	420287	419370	422835
18	2514	2059	3087	58	55522	52752	58645	98	532519	533106	527512
19	279	265	624	59	59060	56902	67941	99	374781	370080	368398
20	4193	4204	6124	60	73328	72600	83571	100	441888	441794	438498
21	0	0	405	61	79884	80343	101292	101	355822	355372	354620
22	0	0	0	62	47860	46466	52994	102	496131	495980	497212
23	0	0	0	63	78822	78081	83555	103	380170	379913	387540
24	1791	1551	5732	64	96378	95113	112549	104	362008	360756	359022
25	0	0	558	65	134881	132078	147626	105	456364	454890	458510
26	0	0	763	66	64054	63278	71587	106	459925	459615	458459
27	229	137	1340	67	34899	32315	38946	107	356645	354097	357073
28	72	19	1804	68	26404	26366	31249	108	468111	466063	467088
29	0	0	224	69	75414	64632	81838	109	415817	414896	419244
30	575	372	1375	70	81200	81356	90278	110	421282	421060	425096
31	0	0	0	71	161233	156272	164161	111	350723	347233	351540
32	0	0	0	72	56934	54849	71005	112	377418	373238	372473
33	0	0	0	73	36465	34082	48624	113	263200	262367	271314
34	0	0	0	74	38292	33725	55999	114	473197	470327	479053
35	0	0	0	75	30980	27248	48219	115	460225	459194	459742
36	0	0	0	76	67553	66847	72685	116	540231	527459	539292
37	2407	2078	4161	77	40558	37257	51069	117	518579	512286	515037
38	0	0	0	78	25105	24795	36889	118	357575	352118	366235
39	0	0	0	79	125824	122051	156197	119	583947	584052	575773
40	0	0	0	80	31844	26470	40775	120	399700	398590	401236

In this appendix, we consider the best of our beam search algorithms, namely the recovering beam search heuristic with a variable width (here denoted simply as RBS). This procedure is applied to the benchmark problem set, and the results are compared with those obtained by the procedures developed by Cicirello and Smith and the ant colony algorithm of Liao and Juan.

A.2. Computational results

The 120 benchmark instances proposed by Cicirello and Smith [25] can be obtained at http://www.ozone.ri.cmu.edu/benchmarks. These problem instances are characterized by the same three factors τ, R and η used in our test problems. The following values are considered for these parameters: $\tau = \{0.3, 0.6, 0.9\}$, $R = \{0.25, 0.75\}$ and $\eta = \{0.25, 0.75\}$. The benchmark set contains 10 problem instances, each with 60 jobs, for each combination of these parameter values. The RBS heuristic was applied to these instances on a Pentium IV 2.8 GHz personal computer.

Table A2
Objective function values

			CCRL	ACO	RBS
		ALL	100.00	99.28	101.85
$\tau = 0.3$	$R = 0.25$	$\eta = 0.25$	100.00	90.70	127.92
		$\eta = 0.75$	100.00	95.72	147.51
	$R = 0.75$	$\eta = 0.25$	100.00	77.95	457.48
		$\eta = 0.75$	100.00	86.33	172.87
$\tau = 0.6$	$R = 0.25$	$\eta = 0.25$	100.00	99.10	105.12
		$\eta = 0.75$	100.00	98.38	107.40
	$R = 0.75$	$\eta = 0.25$	100.00	97.25	112.80
		$\eta = 0.75$	100.00	94.93	121.28
$\tau = 0.9$	$R = 0.25$	$\eta = 0.25$	100.00	99.81	100.04
		$\eta = 0.75$	100.00	99.77	99.72
	$R = 0.75$	$\eta = 0.25$	100.00	99.77	100.28
		$\eta = 0.75$	100.00	99.12	100.16

The filter relative deviation parameter in the RBS algorithm was set at 0.7. This value was determined by performing an interpolation between the values chosen for our instances with 50 and 75 jobs. The values of the remaining parameters required by the RBS procedure were identical for all our instance sizes, and those same values were therefore also used in this experiment.

The results given by the RBS algorithm are compared with those obtained by the procedures developed by Cicirello and Smith [25] and the ACO algorithm of Liao and Juan [26]. The Cicirello and Smith results (henceforth denoted by CCRL) correspond to the best of the solutions generated by the four improvement-type heuristics (LDS, HBSS, VBSS and VBSS-HC) and the simulated annealing algorithm.

In Table A1, we provide the objective function values given by the CCRL, ACO and RBS procedures for each instance in the benchmark set. In Table A2, we present the average objective function values for all instances, as well as for each combination of the τ, R and η parameter values. The average objective function values are calculated relative to the CCRL results, and are therefore presented as index numbers. More precisely, these values are calculated as heur_ofv / ccrl_ofv * 100, where heur_ofv and ccrl_ofv are the average objective function values of the appropriate heuristic and the Cicirello and Smith procedures, respectively. The number of times the RBS algorithm performs better ($<$), equal ($=$) or worse ($>$) than the CCRL and ACO procedures is given in Table A3.

The results presented in these tables show that the RBS heuristic is outperformed, as far as solution quality is concerned, by both the CCRL and ACO procedures. In fact, the CCRL and ACO procedures not only provide a lower average objective function value, but also give better results for around 75% of the benchmark instances. Moreover, the RBS algorithm fails to obtain an optimal solution for 4 of the 16 instances with a zero weighted tardiness.

The tardiness factor τ has a significant effect on the relative performance of the RBS algorithm. Indeed, the relative performance of the RBS heuristic is worse when the tardiness factor has a low or medium value. In fact, when $\tau = 0.3$ or 0.6, the RBS procedure is outperformed in most of the test instances, and its average objective function value is much higher. However, the average objective function values for $\tau = 0.3$ are somewhat misleading. When $\tau = 0.3$, most jobs are early, and the weighted tardiness is low. Therefore, the large differences in the relative objective function values for $\tau = 0.3$ are somewhat deceptive, since they correspond to situations where the variation in the objective function value is small when measured in absolute terms.

The RBS algorithm, on the other hand, is competitive for the instances with a high tardiness factor. In fact, when $\tau = 0.9$, the RBS heuristic provides an average objective function value that is quite close, and for one parameter combination actually better, than the CCRL and ACO procedures. Also, the RBS algorithm provides better results for about half of the instances with $\tau = 0.9$.

The computation times required by the CCRL procedures are not provided in the benchmark library, so no comparison is then possible. However, it is possible to compare the computation times of the RBS and ACO heuristics. Indeed, the ACO procedure of Liao and Juan [26] was also executed on a Pentium IV 2.8 GHz personal computer. Liao and Juan report that each run of their ACO procedure required an average computation time of 4.99 s. For each instance, the

Table A3
Comparison of objective function values

			RBS vs. CCRL			RBS vs. ACO		
			<	=	>	<	=	>
		ALL	21	12	87	16	12	92
$\tau = 0.3$	$R = 0.25$	$\eta = 0.25$	0	0	10	0	0	10
		$\eta = 0.75$	0	1	9	0	1	9
	$R = 0.75$	$\eta = 0.25$	0	2	8	0	2	8
		$\eta = 0.75$	0	9	1	0	9	1
$\tau = 0.6$	$R = 0.25$	$\eta = 0.25$	0	0	10	0	0	10
		$\eta = 0.75$	1	0	9	1	0	9
	$R = 0.75$	$\eta = 0.25$	0	0	10	0	0	10
		$\eta = 0.75$	0	0	10	0	0	10
$\tau = 0.9$	$R = 0.25$	$\eta = 0.25$	5	0	5	4	0	6
		$\eta = 0.75$	6	0	4	6	0	4
	$R = 0.75$	$\eta = 0.25$	4	0	6	3	0	7
		$\eta = 0.75$	5	0	5	2	0	8

ACO algorithm was run 10 times, and the best solution was then selected. Therefore, the ACO results were achieved with an average computation time of about 50 s.

The RBS procedure is significantly faster, since its average computation time is just 0.18 s. Therefore, the RBS heuristic is considerably more efficient, and can be applied to much larger instances. Indeed, the RBS procedure can solve even large instances with 250 jobs in under 10 s.

References

[1] Wilbrecht JK, Prescott WB. The influence of setup time on job shop performance. Management Science 1969;16:B274–80.
[2] Panwalkar SS, Dudek RA, Smith ML. Sequencing research and the industrial scheduling problem. In: Elmaghraby SE, editor. Symposium on the theory of scheduling and its applications. New York: Springer; 1973. p. 29–38.
[3] Wortman DB. Managing capacity: getting the most from your firm's assets. Industrial Engineering 1992;24:47–9.
[4] Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.
[5] Raman N, Rachamadugu RV, Talbot FB. Real time scheduling of an automated manufacturing center. European Journal of Operational Research 1989;40:222–42.
[6] Lee YH, Bhaskaran K, Pinedo M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 1997;29:45–52.
[7] Lee YH, Pinedo M. Scheduling jobs on parallel machines with sequence-dependent setup times. European Journal of Operational Research 1997;100:464–74.
[8] Allahverdi A, Gupta JND, Aldowaisan T. A review of scheduling research involving setup considerations. Omega 1999;27:219–39.
[9] Potts CN, Kovalyov MY. Scheduling with batching: a review. European Journal of Operational Research 2000;120:228–49.
[10] Sen T, Sulek JM, Dileepan P. Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. International Journal of Production Economics 2003;83:1–12.
[11] Brucker P. Scheduling algorithms. Berlin: Springer; 2004.
[12] Lowerre BT. The HARPY speech recognition system. PhD thesis, Carnegie-Mellon University, USA, 1976.
[13] Rubin S. The ARGOS image understanding system. PhD thesis, Carnegie-Mellon University, USA, 1978.
[14] Fox MS. Constraint-directed search: a case study of job-shop scheduling. PhD thesis, Carnegie-Mellon University, USA, 1983.
[15] Ow PS, Smith SF. Viewing scheduling as an opportunistic problem-solving process. Annals of Operations Research 1988;12:85–108.
[16] Sabuncuoglu I, Bayiz M. Job shop scheduling with beam search. European Journal of Operational Research 1999;118:390–412.
[17] Ow PS, Morton TE. Filtered beam search in scheduling. International Journal of Production Research 1988;26:35–62.
[18] Ow PS, Morton TE. The single machine early/tardy problem. Management Science 1989;35:177–91.
[19] Della Croce F, T'kindt V. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. Journal of the Operational Research Society 2002;53:1275–80.
[20] Della Croce F, Ghirardi M, Tadei R. Recovering beam search: enhancing the beam search approach for combinatorial problems. Journal of Heuristics 2004;10:89–104.
[21] Valente JMS, Alves RAFS. Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. Computers and Industrial Engineering 2005;48:363–75.

[22] Ghirardi M, Potts CN. Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. European Journal of Operational Research 2005;165:457–67.
[23] Esteve B, Aubijoux C, Chartier A, T'kindt V. A recovering beam search algorithm for the single machine just-in-time scheduling problem. European Journal of Operational Research 2006;172:798–813.
[24] Potts CN, van Wassenhove LN. A branch-and-bound algorithm for the total weighted tardiness problem. Operations Research 1985;33: 363–77.
[25] Cicirello VA, Smith SF. Enhancing stochastic search performance by value-biased randomization of heuristics. Journal of Heuristics 2005;11: 5-34.
[26] Liao CJ, Juan HC. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research 2007;34:1899–1909.