Journal of Manufacturing Systems
Vol. 24/No. 1
2005

Beam Search Algorithms for the Early/Tardy Scheduling Problem with Release Dates

Jorge M.S. Valente and **Rui A.F.S. Alves,** Faculdade de Economia, Universidade do Porto, Portugal

Abstract

This paper presents several beam search algorithms for the single-machine earliness/tardiness scheduling problem with release dates and no unforced idle time. These algorithms include classical beam search procedures, with both priority and total cost evaluation functions, as well as the filtered and recovering variants. Both priority evaluation functions and problem-specific properties were considered for the filtering step used in the filtered and recovering procedures.

The computational results show that the recovering beam search algorithms outperform their filtered counterparts, while the priority-based filtering procedure proves superior to the rules-based alternative. The beam search procedure with a total cost function provides very good results but is computationally expensive. The recovering algorithm is quite close in solution quality and is significantly faster, so it can be used to solve even large instances.

Keywords: *Scheduling, Early/Tardy, Beam Search, Heuristics*

Introduction

This paper considers a single-machine scheduling problem with release dates and due dates, earliness and tardiness costs, and no unforced machine idle time. Scheduling models with only one machine may appear to arise infrequently in practice. However, the performance of many production systems is often dictated by the quality of the schedules for a single bottleneck machine. Therefore, models with only one processor are most useful in practice for scheduling such a machine. Furthermore, the analysis of single-machine problems provides valuable insights that enable more complex systems to be scheduled. In fact, scheduling systems with multiple processors can sometimes be relaxed to a single-machine problem, or a sequence of such single-processor problems. Also, the solution procedures for complex systems, such as job shop environments, often require solving single-machine subproblems.

The different job due dates can represent the delivery dates committed to the customers, or the time a certain part or component is required by a stage further down the production or assembly line. Scheduling problems with different release dates are also appealing because in most real production settings the orders are released to the shop floor over time.

Scheduling models with both earliness and tardiness penalties are compatible with the philosophy of just-in-time (JIT) production. The JIT production philosophy emphasizes producing goods only when they are needed and, therefore, takes up the view that both earliness and tardiness should be discouraged. In a JIT production environment, jobs that are completed early must be held in inventory until their due dates, while jobs that finish late may cause a customer delay, or in a further stage in the production line, even shut down operations. Therefore, an ideal schedule in one in which all jobs are completed exactly on their due dates. Scheduling models with both early and tardy costs are then compatible with the JIT philosophy because jobs are indeed scheduled to finish as close as possible to their due dates.

The earliness penalty, in addition to a holding cost for parts or finished products, may also represent deterioration in the production of perishable goods, as well as the cost of completing a project early in project management critical path analyses, as suggested by Sidney (1977). The tardiness penalty can represent rush shipping costs, lost sales, and loss of goodwill, as well as disruptions and delays in stages further down the production line. This paper considers a general model with different penalties for earliness and tardiness. Furthermore, the penalties may be different for each job, reflecting the fact that each order may have different customer priorities as well as distinct storage requirements and costs.

It is assumed that no unforced machine idle time is allowed, so the machine is idle only when no unscheduled jobs are available. This assumption is appropriate for many production settings. When the

capacity of the machine is limited when compared with the demand, the machine must be kept running in order to satisfy the customers' orders. Idle time must also be avoided for machines with high operating costs because the cost of keeping the machine running is then higher than the earliness cost incurred by completing a job before its due date. In certain production environments, it might even not be feasible to leave the machine idle for a period of time. Also, the assumption of no idle time is justified when starting a new production run involves high setup costs or times. Examples of production settings where the no idle time assumption is appropriate have been given by Korman (1994) and Landis (1993). More specifically, Korman considers the Pioneer Video Manufacturing (now Deluxe Video Services) disc factory at Carson, California, while Landis analyzes the Westvaco envelope plant at Los Angeles.

The assumption of no unforced idle time is compatible with the existence of different release dates, as long as the forced idle time caused by the presence of distinct release dates is inexistent or quite small. If that is not the case, the assumption becomes unrealistic because the machine capacity is then clearly not limited when compared with the demand, and it is unlikely that the machine idleness cost is higher than the earliness cost.

The problem considered here can be formally 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 assumed to be continuously available from time zero onward and preemptions are not allowed. Job J_j, $j = 1, 2, \ldots, n$, becomes available for processing at its release date, r_j, requires a processing time, p_j, and should ideally be completed on its due date, d_j. Given a schedule, the earliness of J_j is defined as $E_j = \max\{0, d_j - C_j\}$, while the tardiness of J_j can be defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of J_j. The objective is then to find a schedule that minimizes the sum of weighted earliness and weighted tardiness costs $\sum_{j=1}^{n}\left(h_j E_j + w_j T_j\right)$ subject to the constraint that no unforced machine idle time is allowed, where h_j and w_j are the earliness and tardiness penalties, respectively, of job J_j.

As a generalization of weighted tardiness scheduling (Lenstra, Rinnooy Kan, and Bruckner 1977), the problem is strongly NP-hard. Several lower-bounding procedures and a branch-and-bound algorithm based on a decomposition of the problem into weighted earliness and weighted tardiness subproblems were presented by Valente and Alves (2005). The performance of various heuristics, including dispatch rules, a greedy procedure, and a decision theory algorithm, was analyzed in Valente and Alves (2003). The early/tardy problem with equal release dates and no idle time has also been considered by several authors, and both exact and heuristic approaches have been proposed. Among the exact approaches, branch-and-bound algorithms were presented by Abdul-Razaq and Potts (1988), Li (1997), and Liaw (1999). Among the heuristics, Ow and Morton (1989) developed several dispatch rules and a filtered beam search procedure, while Li (1997) presented a neighborhood search algorithm. The weighted tardiness problem with release dates was also previously considered. A dominance rule and several heuristics were presented by Akturk and Ozdemir (2001), while Akturk and Ozdemir (2000) developed lower-bounding procedures and a branch-and-bound algorithm.

This paper presents several heuristic algorithms based on the beam search technique. These algorithms include classical beam search procedures, with both priority and total cost evaluation functions, as well as the filtered and recovering variants. Considered are both priority evaluation functions and problem-specific properties for the filtering step used in the filtered and recovering beam search heuristics. Extensive computational tests were performed to determine the parameter values that provided the best balance between solution quality and computational effort for each algorithm. Also considered were the use of some dominance rules to improve the solutions obtained by the heuristics.

Beam Search Approach

Beam search is a heuristic method for solving combinatorial optimization problems that consists of a truncated branch-and-bound procedure. At each level of the search tree, only the most promising nodes are retained for further branching, while the remaining nodes are pruned off permanently. Because only some nodes are kept at each tree level, the running time is polynomial in the problem size.

The beam search approach was first used in the artificial intelligence community for the speech recognition problem (Lowerre 1976), and some applications to job shop scheduling problems have appeared in the literature since then (Ow and Smith

1988; Sabuncuoglu and Bayiz 1999). A variation of this technique, called filtered beam search, was presented in Ow and Morton (1988, 1989) and applied to the single-machine early/tardy problem. Another variation of the beam search approach, denoted as recovering beam search, was recently developed by Della Croce and T'kindt (2002) and tested on the single-machine completion time problem with release dates.

The classical beam search procedure consists of an adaptation of branch-and-bound 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*. Because the intent of this technique is to search quickly, the remaining nodes are simply eliminated 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. The beam search approach deals with this danger by choosing a number (the beam width) of promising paths that are searched concurrently. A wider beam width reduces the risk of pruning a node that would ultimately lead to the optimum solution, but at the cost of increased computational effort.

The node evaluation process plays a major role in the effectiveness of beam search algorithms. Two different types of evaluation functions have been used: *priority evaluation functions* and *total cost evaluation functions*. The first simply calculates a priority or urgency rating, typically by using a dispatch heuristic to calculate the priority of the last job added to the current partial schedule. The second calculates an estimate of the minimum total cost of the best solution that can be obtained from the partial schedule associated with the node. This estimate is usually obtained by using a dispatch rule to schedule the remaining jobs and complete the existing partial sequence.

The priority evaluation function only considers the next decision to be made (the next job to schedule) and, therefore, has a local view of the problem. The total cost evaluation function, on the other hand, has a more global view because it projects from the current partial solution to a complete schedule to obtain a total cost estimate. The classical beam search procedure has, therefore, two variants, denoted as priority beam search and detailed beam search, that differ only in the type of evaluation function used. In priority beam search, a priority evaluation function is used, while detailed beam search uses a total cost evaluation.

The priority evaluation functions can pose a slight problem. The dispatch rules used to calculate the urgency rating of the last scheduled job are usually functions of the current partial schedule, namely functions of the current time. Different nodes on the same level correspond to different partial schedules and have different completion times. Therefore, the priorities obtained for the offspring of a node cannot be legitimately compared with those obtained from expanding another node; these priorities are then context-dependent. This problem can be overcome by initially selecting the best β children of the root node (that is, the node containing only unscheduled jobs). At lower levels of the search tree, only the best descendant of each beam node is saved for the next iteration. The total cost evaluation function is not affected by this problem because the total cost estimates are context-independent and can be compared.

Priority evaluation functions can be computed quickly, but they are potentially inaccurate and may result in the elimination of good nodes. The total cost functions, on the other hand, perform a more accurate evaluation but require a higher computational effort. The filtered beam search method uses a two-stage approach that takes advantage of both crude and accurate evaluations, thus trying to provide a high-quality evaluation procedure that does not require excessive computation times. A computationally efficient filtering procedure is first applied to choose only some of the offsprings of each beam node for a more detailed evaluation. A total cost function is then used to accurately evaluate these nodes, and the best β nodes are then kept for further branching. Typically, a priority evaluation function is used in the filtering procedure to calculate an urgency value for each offspring. The best α children of each beam node are then selected for the detailed evaluation step, where α is the so-called *filter width*. Recently, Della Croce and T'kindt (2002) proposed a new filtering procedure that uses problem-specific properties to determine the nodes that advance to the detailed evaluation step.

The recovering beam search procedure, like the filtered beam search method, also uses a two-stage approach with both crude and detailed evaluations. However, it differs in three major ways from the filtered beam search technique. First, only a single node is retained at each level of the search tree to reduce

the computational requirements; this means that the beam width has a predefined value of one ($\beta = 1$). Second, the accurate evaluation uses a weighted sum of both lower and upper bounds on the total cost of the best solution that can be obtained from each node. Finally, a *recovering step* is applied when the best node and the corresponding best partial solution are determined at each level of the search tree. This recovering step typically consists in applying a neighborhood search procedure to check whether the current partial solution, σ, is dominated by another partial solution, $\bar{\sigma}$, having the same level of the search tree. If a better partial solution, $\bar{\sigma}$, does indeed exist, then $\bar{\sigma}$ becomes the new current partial solution.

In the context of scheduling problems, local search procedures typically use transpose, interchange, or insert neighborhood operators. The transpose operator consists in swapping adjacent jobs, while the interchange operator swaps a pair of jobs, regardless of their adjacency. The insertion operator removes one job from its current position in the schedule and inserts it in a different location. The total number of nodes explored by the recovering beam search procedure is still polynomial in the problem size because the recovering step can only replace a partial solution with another partial solution with the same depth of the search tree.

Recovering beam search and classical or filtered beam search methods use different strategies to deal with the danger of eliminating a node that would ultimately lead to the optimal solution. While classical or filtered beam search allow several paths to be searched in parallel, recovering beam search only keeps a single node at each level and instead relies on the recovering step to recover from previous wrong decisions.

Proposed Heuristic Procedures

This section describes the several algorithms that were considered. Both priority and detailed classical beam search algorithms were tested, as well as filtered and recovering beam search procedures. Also considered were both types of filtering procedures that have been previously used. The first type of filter requires a priority evaluation function, while the second uses problem-specific properties to choose the nodes that advance to the accurate evaluation step. From now on, the priority and detailed beam search algorithms will be denoted as PBS and DBS, respectively. The filtered beam search algorithms with priority evaluation function and problem-specific rules-filtering procedures will be respectively identified as FBS_P and FBS_R. Similarly, RBS_P and RBS_R will denote the recovery beam search algorithms with a priority-based and a rules-based filter, respectively.

To apply these algorithms to the early/tardy problem, it is necessary to specify their main components, namely the branching scheme, priority evaluation function, total cost evaluation function (that is, upper-bounding procedure), filtering procedure, lower-bounding procedure, and recovering step. The branching scheme, common to all algorithms, is the usual n-ary forward branching: the sequence is constructed by adding one job at a time starting from the first position, and the search tree is such that a branch at level l indicates the job scheduled in position l.

The priority and total cost evaluation functions are also common to all algorithms. These evaluation functions use the LINET heuristic, which provided the best results of all the dispatch rules analyzed in Valente and Alves (2003). The LINET heuristic uses the following priority index, $I_j(t)$, to determine the job, J_j, to be scheduled at any instant, t, when both the machine and at least one unscheduled job are available:

$$I_j(t) = \begin{cases} W_j & \text{if } s_j \le 0 \\ W_j - \dfrac{s_j\left(H_j + W_j\right)}{k\bar{p}} & \text{if } 0 \le s_j \le k\bar{p} \\ -H_j & \text{otherwise,} \end{cases}$$

where $W_j = w_j / p_j$, $H_j = h_j / p_j$, and $s_j = d_j - t - p_j$ is the slack of job J_j at time t, $\bar{p}$ is the average processing time of the remaining jobs, and k is a look-ahead parameter. The value of k was set at 2.5, as recommended in Valente and Alves (2003). In the priority evaluation function, the priority index of the LINET heuristic is used to calculate the priority of the last scheduled job in each node. The total cost evaluation function uses the LINET dispatch rule to sequence the unscheduled jobs, thereby completing the existing partial schedule and calculating a total cost estimate.

The main steps of the PBS and the DBS algorithms are now presented. Throughout this section, let B be the set of nodes retained in the beam for further

branching, C be a set of offspring nodes, and n_0 be the root node.

PBS:

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 using the LINET heuristic priority index $I_j(t)$.
   Select the min{β, number of children} best child nodes and add them to B.
2. 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 using the LINET heuristic priority index $I_j(t)$.
   (c) Select the best child node and add it to C.
3. Set $B = C$.
   Set $C = \varnothing$.
4. Stopping condition:
   If the nodes in B are leaves (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:

1. Initialization:
   Set $C = \varnothing$.
   Set $B = \{n_0\}$.
2. For each node in B:
   (a) Branch the node generating the corresponding children.
   (b) For each child node, sequence the unscheduled jobs using the LINET heuristic and calculate an upper bound on the optimal solution value.
   (c) Select the min{β, number of children} best child nodes and add them to C.
3. Set $B = \varnothing$.
   Select the min{β, $|C|$} best nodes in C and add them to B.
   Set $C = \varnothing$.
4. Stopping condition:
   If the nodes in B are leaves, select the node with the lowest total cost as the best sequence found and stop.
   Otherwise, go to step 2.

A filtering procedure is required by both the filtered and the recovering beam search algorithms. Both types of filtering procedures that have been previously used are considered. The first requires a priority evaluation function, and the α best children of each beam node are selected for the accurate evaluation step. This priority-based filter once again uses the priority index of the LINET heuristic to calculate the priority of the last scheduled job in each node.

The second filtering procedure uses problem-specific properties to choose the nodes that advance to the accurate evaluation step. Let x be a partial sequence and let $i, j \notin x$ be a pair of jobs that can be feasibly scheduled in the next position in the sequence. Three criteria are now presented that were used to determine the nodes eliminated from the detailed evaluation step.

Criterion 1—If i and j are both early, regardless of their order, in the next two positions in the sequence, and $h_i / p_i \leq h_j / p_j$, then job j is eliminated.

Criterion 2—If i and j are both tardy, regardless of their order, in the next two positions in the sequence, and $w_i / p_i \geq w_j / p_j$, then job j is eliminated.

Criterion 3—If j is always early and i is always tardy when scheduled in the next two positions in the sequence, then job j is eliminated.

Criterion 1 is based on a condition for the weighted earliness scheduling problem with no idle time. If two adjacent jobs are always completed early, regardless of their order, those jobs should be scheduled in weighted longest processing time order, that is, in non-decreasing order of their h_j / p_j ratios. When two jobs, i and j, are being considered for the next position in the sequence, and those jobs are both early, regardless of their order, when scheduled in the next two positions, criterion 1 then eliminates the job with the higher h_j / p_j ratio.

Similarly, criterion 2 is based on a condition for the weighted tardiness problem. In fact, when two adjacent jobs are necessarily tardy, regardless of their order, those jobs should be scheduled in weighted shortest processing time order, that is, in non-increasing order of their w_j / p_j ratios. If jobs i and j are being considered for the next position, and they are both tardy, regardless of their order, when scheduled in the next two positions, criterion 2 eliminates the job with the lower w_j / p_j ratio. Finally, when i and j are considered for the next position, and j (i) is

always early (tardy) when scheduled in the next two positions, criterion 3 eliminates the job that is always early. The main steps of the filtered beam search algorithm with a priority-based filter are now presented.

FBS_P:

1. Initialization:
   Set $C = \varnothing$.
   Set $B = \{n_0\}$.
2. 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 using the LINET heuristic priority index $I_j(t)$.
   (c) Select the min{α, number of children} best child nodes and add them to C.
3. Set $B = \varnothing$.
   For all nodes in C, sequence the unscheduled jobs using the LINET heuristic and calculate an upper bound on the optimal solution value.
   Select the min{β, $|C|$} best nodes in C and add them to B.
   Set $C = \varnothing$.
4. Stopping condition:
   If the nodes in B are leaves, select the node with the lowest total cost as the best sequence found and stop.
   Otherwise, go to step 2.

The steps of the filtered beam search algorithm with a rule-based filter are similar to those presented for the FBS_P algorithm, with the exception that steps 2 (b) and 2 (c) are replaced by:

2 (b) Select the child nodes that are not eliminated by criteria 1 to 3 and add them to C.

The recovering beam search algorithms additionally require a lower bounding procedure and a recovering step. The lower bound is calculated using the method presented by Valente and Alves (2005). This lower-bounding procedure, which will be denoted as IRDLB, relaxes the assumption that a job cannot be scheduled before its release date to calculate a lower bound for a problem with identical release dates. The recovering step uses an insertion procedure to determine whether the current partial solution is dominated by another partial solution with the same level of the search tree. The last job in the current partial schedule is inserted before the previously scheduled jobs until a maximum of $\delta(n-1)$, $0 \leq \delta \leq 1$ insertions have been performed; this insertion procedure is also stopped if the next insertion would lead to an infeasible schedule (that is, the job would be scheduled to start before its release date). The parameter δ determines the maximum number of insertions (alternative schedules) that are considered and, therefore, controls the extent of the local search performed during the recovering step.

Presented next are the main steps of the recovering beam search algorithm with a priority filter. In the following, let σ denote the node that is retained in the beam, and let $0 \leq \gamma \leq 1$ be the upper-bound weight in the weighted sum of lower and upper bounds.

RBS_P:

1. Initialization:
   Set $C = \varnothing$.
   Set $\sigma = n_0$.
2. Branch σ generating the corresponding children.
   Calculate the priority of the last scheduled job for each child node using the LINET heuristic priority index $I_j(t)$.
   Select the min{α, number of children} best child nodes and add them to C.
3. For all nodes in C:
   (a) Sequence the unscheduled jobs using the LINET heuristic and calculate an upper bound UB on the optimal solution value.
   (b) Apply the IRDLB lower bounding procedure to the unscheduled jobs and calculate a lower bound LB on the optimal solution value.
   (c) Compute the evaluation function $V = (1 - \gamma)\, LB + \gamma\, UB$.
4. Let σ^* be the node in C with the lowest value of V.
   Set $\sigma = \sigma^*$.
   Set $C = \varnothing$.
5. Recovering step:
   Insert the last job in the current partial schedule before the previously scheduled jobs until a maximum of $\delta(n-1)$ insertions have been performed.
   If the best alternative partial solution $\bar{\sigma}$ found dominates σ, set $\sigma = \bar{\sigma}$.

6. Stopping condition:
   If σ is a leaf node, stop; σ's cost is the best objective function value found.
   Otherwise, go to step 2.

The steps of the recovering beam search algorithm with a rule-based filter are similar to those presented for the RBS_P algorithm, with the exception that step 2 is replaced by:

2. Branch σ generating the corresponding children.
   Select the child nodes that are not eliminated by criteria 1 to 3 and add them to *C*.

It must also be noted that the existence of different release dates motivated a slight change in the PBS, FBS_P, and RBS_P procedures. The previous section indicated that the PBS procedure selects only the best child of each beam node, while the other two algorithms will also select just a single child for detailed evaluation when the filter width is one. When release dates are allowed to be different, it is quite possible that the root node has only one (or very few) offspring. If no correction was made to the algorithms, only one node would be retained in the beam throughout the whole procedure, independently of the beam width. Therefore, in such situations, the number of chosen offspring is increased temporarily to allow the number of beam nodes to increase up to β. The proposed algorithms were compared with two other heuristics previously analyzed in Valente and Alves (2003), namely the LINET dispatch rule and the Decision Theory Search (DTS) algorithm. The DTS algorithm is based on the decision theory approach of Kanet and Zhou (1993) and is identical to the detailed beam search algorithm with a beam width of one.

Two dominance conditions were developed by Ow and Morton (1989) and Liaw (1999) for the problem with identical release dates. Ow and Morton's rule imposes a condition on adjacent pairs of jobs, while the dominance rule presented by Liaw applies to non-adjacent jobs with identical processing times. These rules can still be used when the release dates are allowed to differ, provided care is taken to avoid making unfeasible job swaps. These dominance rules were used in Valente and Alves (2003) to improve the solution quality of several heuristic procedures with little additional computational effort. These dominance rules were also considered to be used as an improvement step once an initial solution has been obtained by the heuristics (see Valente and Alves 2003 for details).

Computational Results

This section presents the results from the computational tests. A set of problems with 15, 20, 25, 30, 50, 75, 100, 200, 250, 300, 400, 500, and 1000 jobs was randomly generated as follows. For each job, J_j, an integer processing time, p_j, an integer earliness penalty, h_j, and an integer tardiness penalty, w_j, were generated from one of the two uniform distributions, [1, 10] and [1, 100], to create low and high variability, respectively. For each job, J_j, an integer release date, r_j, was generated from the uniform distribution, $\left[0, R\sum_{j=1}^{n} p_j\right]$, where R was set at 0.25, 0.50, and 0.75. The maximum value of the range of release dates, R, was chosen so that the forced idle time would be small or inexistent. Preliminary tests showed that $R = 1.00$ would lead to excessive amounts of forced idle time, which would be incompatible with the assumption that no unforced idle time may be inserted in a schedule.

Instead of determining due dates directly, slack times were generated between a job's due date and its earliest possible completion time. For each job, J_j, an integer due date slack, s_j^d, was generated from the uniform distribution, $\left[0, D\sum_{j=1}^{n} p_j\right]$, where the due date slack range, D, was set at 0.10, 0.25, and 0.50. The due date, d_j, of J_j was then set equal to $d_j = r_j + p_j + s_j^d$. For each combination of problem size n, processing time, and penalty variability (var), R and D, 50 instances were randomly generated. Therefore, a total of 450 instances were generated for each combination of problem size and variability.

All the algorithms were coded in Visual C++ 6.0 and executed on a Pentium IV–1500 MHz personal computer. Due to the large computational times that would be required, the DTS heuristic was not applied to the 1,000 job instances, while the DBS algorithm was only used to solve instances with up to 400 jobs. Throughout this section, and to avoid excessively large tables or figures, results sometimes will be presented only for some representative cases.

Extensive computational tests were first performed to determine appropriate values for the parameters used by the algorithms. Increasing the value of the parameters usually improves the objective function

value, at the cost of increased computation times (the only exception being the γ parameter). Therefore, values were determined that provided a good trade-off between solution quality and computational effort. The following values were considered for the several parameters:

$\alpha = \{1, 2, \ldots, 10\}$
$\beta = \{1, 2, \ldots, 8\}$
$\gamma = \{0.1, 0.2, \ldots, 0.9\}$
$\delta = \{0.05, 0.10, \ldots, 0.50\}$

The algorithms were applied to selected problem sizes for all combinations of the relevant parameters. Then a thorough analysis was performed for objective function values and run times, and parameter values were selected that seemed to provide the best balance between solution quality and computation time. The chosen values are presented in *Table 1*, and they give an adequate compromise between schedule cost and computational requirements for all the instance types included in these preliminary tests.

Table 2 presents the average objective function value (mean ofv) for each heuristic, both before (bfr) and after (aft) the application of the dominance rules, and the average of the relative improvements in the objective function values (%ch), calculated as $(H - H_{DR}) / H * 100$, where H and H_{DR} are the objective function values of a heuristic before and after the dominance rules, respectively. Also given is the number of times each heuristic produces the best result when compared with the other heuristics (# best), both before and after the use of the dominance rules. A test was also performed to determine if the differences between the heuristic objective function values before and after the dominance rules are statistically significant. Given that the heuristics were used on exactly the same problems, a paired-samples test is appropriate. Because the hypotheses of the paired-samples t-test were not all met, the non-parametric Wilcoxon test was selected. The significance values of this test—that is, the level of significance values above which the equal distribution hypothesis is to be rejected—were nearly always equal to 0.000 and were never larger than 0.05.

The best results are usually given by the DBS heuristic. The DTS and RBS_P are then the best performing algorithms, followed by the FBS_P procedure. The RBS_P and DTS algorithms, in particular, are quite close to the best heuristic procedure, providing results that are usually less than 0.5%

Table 1
Heuristic Parameter Values

Heuristic	α	β	γ	δ
PBS	—	4	—	—
DBS	—	3	—	—
FBS_P	3	3	—	—
FBS_R	—	3	—	—
RBS_P	3	—	0.8	0.10
RBS_R	—	—	0.8	0.10

(1%) above those of the DBS algorithm for instances with low (high) variability. The PBS procedure provides better results than the LINET dispatch rule, particularly for instances with high variability. The algorithms with a rule-based filter were clearly outperformed by their priority evaluation function counterparts. In fact, the performance of the FBS_R and RBS_R algorithms is rather poor because these algorithms cannot even match the simple LINET dispatch procedure. The simple rules that were used could not avoid eliminating nodes that would lead to good solutions. Therefore, better rules would be needed to implement a competitive-rule filter procedure. However, the development of improved conditions that could indeed lead to an effective rule-based filter would likely require further theoretical results concerning the early/tardy problem with release dates.

The RBS algorithms also provide better results than their FBS alternatives, for both types of filtering procedures. *Table 2* shows that the use of the dominance rules improves the heuristic results, and the Wilcoxon test values indicate that this improvement is statistically significant. The relative improvement given by the dominance rules is much higher for instances with low variability. For these problems, even the best heuristics can benefit from a 1% to 2% decrease in objective function value.

The effect of the R and D parameters on the relative objective function value improvement is given in *Table 3* for the RBS_P heuristic. The relative improvement is usually non-decreasing with the due date slack range, D, and the highest relative improvement values occur when D is equal to 0.50. The improvement provided by the dominance rules is usually lower when D is equal to 0.10 and the range of release dates, R, is set at 0.25 or 0.50.

Figure 1 presents the heuristic run times, in seconds, before the application of the dominance rules. The run time of the LINET dispatch heuristic is not given because it could hardly be measured, even for

Table 2
Heuristic Results: Objective Function Value and Number of Times Each Heuristic Gives the Best Result

		Low Var					High Var				
		Mean ofv			# Best		Mean ofv			# Best	
n	Heuristic	bfr	aft	%ch	bfr	aft	bfr	aft	%ch	bfr	aft
50	PBS	6116	5993	2.5	0	20	463630	455351	2.4	0	15
	DBS	5862	5813	1.1	132	199	442640	439277	1.0	140	216
	DTS	5924	5854	1.5	27	99	446444	441617	1.4	32	123
	FBS_P	5875	5844	0.7	107	164	444064	442086	0.6	84	161
	FBS_R	6210	6109	1.9	13	26	471941	467434	1.2	8	26
	LINET	6174	6048	2.5	1	15	469706	461679	2.3	0	11
	RBS_P	5842	5823	0.3	228	169	441436	440634	0.2	230	174
	RBS_R	6179	6109	1.2	35	27	468312	466825	0.3	34	28
100	PBS	21914	21546	2.1	1	3	1688047	1668808	1.6	0	2
	DBS	21115	20898	1.4	153	192	1615371	1602600	1.1	162	205
	DTS	21264	20991	1.7	29	85	1627424	1612292	1.3	25	82
	FBS_P	21216	21086	0.8	81	72	1629726	1624045	0.5	74	68
	FBS_R	22197	21828	2.1	0	4	1722101	1707834	1.1	0	2
	LINET	22052	21665	2.1	0	2	1705855	1686123	1.6	0	1
	RBS_P	21124	20996	0.7	190	136	1618767	1614084	0.4	196	140
	RBS_R	22089	21791	1.6	6	4	1711754	1706704	0.3	5	2
250	PBS	129856	127219	2.2	3	9	9763218	9696879	0.9	0	1
	DBS	126444	124716	1.8	169	169	9413967	9362293	0.7	217	234
	DTS	127172	125077	2.3	44	91	9446220	9389107	0.8	45	88
	FBS_P	127418	125904	1.4	88	57	9531987	9508352	0.3	73	45
	FBS_R	131051	128155	2.4	0	1	9895687	9828904	0.9	0	0
	LINET	130382	127601	2.3	1	2	9802122	9731758	0.9	0	1
	RBS_P	126958	125301	1.5	144	123	9500053	9471279	0.4	122	104
	RBS_R	130583	127992	2.1	1	0	9853781	9820274	0.3	2	1
500	PBS	502348	490681	2.5	7	8	37199701	36978650	0.7	0	0
	DBS	—	—	—	—	—	—	—	—	—	—
	DTS	494202	484877	2.6	183	246	36072578	35916221	0.6	294	329
	FBS_P	496529	488347	1.8	84	56	36598921	36489490	0.3	65	37
	FBS_R	504867	492401	2.7	0	0	37528605	37286214	0.8	0	0
	LINET	503212	491349	2.5	0	9	37303483	37072673	0.7	0	0
	RBS_P	494709	486566	1.8	173	128	36536430	36383156	0.5	91	84
	RBS_R	503587	491907	2.4	3	3	37419585	37253876	0.4	0	0
1000	PBS	1954877	1899942	2.9	13	33	145069697	143974245	0.8	0	2
	DBS	—	—	—	—	—	—	—	—	—	—
	DTS	—	—	—	—	—	—	—	—	—	—
	FBS_P	1941260	1895334	2.5	121	123	143641142	142860758	0.5	200	185
	FBS_R	1958266	1903105	2.9	0	14	145899873	144652625	0.9	0	1
	LINET	1956643	1901018	2.9	10	19	145251244	144108069	0.8	1	4
	RBS_P	1935456	1890361	2.4	300	246	143460049	142472835	0.7	249	258
	RBS_R	1955269	1901935	2.8	6	15	145664958	144614087	0.7	0	1

the largest instances. The computational time required by the dominance rules was barely noticeable, and the use of these rules as an improvement procedure is therefore recommended because they allow for improvements in solution quality. As *Figure 1* clearly shows, the DBS and DTS heuristics are computationally demanding and are, therefore, limited to small and medium-sized problems, while the FBS and particularly the PBS and RBS algorithms are much faster and can be used to solve even large instances. The procedures with a rule-based filter are somewhat faster than their priority function counterparts. *Figure 1* also shows that the RBS algorithms are much faster than the FBS procedures and can solve even the largest instances within reasonable computational times (around 30 seconds for instances

Table 3
Relative Improvement for RBS_P Heuristic

		Low Var			High Var		
n	*R*	*D*=0.10	*D*=0.25	*D*=0.50	*D*=0.10	*D*=0.25	*D*=0.50
100	0.25	0.3	0.7	0.8	0.1	0.1	0.2
	0.50	0.1	0.5	1.1	0.1	0.3	0.6
	0.75	0.5	0.5	1.4	0.4	0.3	1.0
300	0.25	0.6	1.7	3.2	0.1	0.3	0.5
	0.50	0.7	1.4	2.3	0.1	0.3	0.5
	0.75	1.1	1.2	1.8	0.4	0.4	0.8
500	0.25	0.8	2.1	3.9	0.2	0.4	0.8
	0.50	0.9	2.0	2.9	0.1	0.4	0.6
	0.75	1.4	1.2	1.3	0.5	0.5	1.0

with 1000 jobs). The variability of the processing times and penalties only had a noticeable effect on the run times of the FBS_R and RBS_R procedures, which require lower computational times when the variability is high.

The heuristic results were also compared with the optimum objective function values for instances with up to 30 jobs; results obtained after the application of the dominance rules are indicated by appending "+ DR" to the heuristic identifiers. *Table 4* presents the average of the relative deviations from the optimum, calculated as $(H - O) / O * 100$, where H and O are the heuristic and optimum objective function values, respectively. *Figure 2* gives the percentage number of times each heuristic generates an optimum solution after the application of the dominance rules. From *Table 4*, it can be seen that all the heuristics are somewhat closer to the optimum for problems with a low processing time and penalty variability. The average performance of the DBS heuristic is quite good because it provides results that are 0.5% to 1.5% above the optimum. Also, as shown in *Figure 2*, the DBS procedure calculates an optimum schedule for a large number of instances. The RBS_P procedure also performs quite well because it gives solutions that are about 1% to 2% above the optimum, and generates an optimum schedule for roughly 60% (30%) of the 20 (30) job test instances.

Table 5 presents the effect of the R and D parameters on the relative deviation from the optimum for the RBS_P + DR heuristic. The relative deviation appears to increase with the due date slack range, D, particularly when R is lower than 0.75. The heuristic performance is worst when D is equal to 0.50 and R is equal to 0.25 or 0.50. The heuristics are usually closer to the optimum when D is equal to 0.10 and the range of release dates, R, is set at 0.25 or 0.50. These results are similar to those reported for the relative improvement provided by the dominance rules and seem to indicate that the problem is harder when the due date slack range is high and the release dates are not widely spread. This is to be expected because the number of jobs that can be scheduled in a given position is larger when the release dates are only slightly scattered. Also, most jobs will likely be tardy when D is low, and as D increases there will be a greater balance between the number of early and tardy jobs, so the problem becomes much harder.

Table 4
Relative Deviation from Optimum Objective Function Value

	Low Var				High Var			
Heuristic	*n* = 15	*n* = 20	*n* = 25	*n* = 30	*n* = 15	*n* = 20	*n* = 25	*n* = 30
PBS	6.0	6.3	6.6	7.5	5.6	6.5	7.6	9.0
DBS	0.7	1.0	1.5	2.3	0.5	1.1	1.6	2.4
DTS	2.2	2.3	2.7	3.6	1.9	2.3	2.8	3.7
FBS_P	0.8	1.2	1.7	2.3	0.6	1.2	1.7	2.6
FBS_R	4.2	5.6	7.5	9.4	3.9	5.7	8.0	9.2
LINET	8.1	7.5	8.1	9.5	7.3	8.2	8.9	10.6
RBS_P	0.7	1.1	1.3	2.0	0.6	1.0	1.4	2.3
RBS_R	3.9	5.1	6.9	8.4	3.5	4.9	7.2	8.4
PBS + DR	2.6	3.2	3.6	4.5	2.4	3.4	4.2	5.8
DBS + DR	0.4	0.5	1.0	1.5	0.3	0.8	1.1	1.6
DTS + DR	1.2	1.2	1.6	2.3	0.9	1.4	1.8	2.4
FBS_P + DR	0.5	0.8	1.3	1.8	0.4	0.9	1.4	2.1
FBS_R + DR	3.5	4.3	6.2	7.5	3.2	4.8	6.6	8.1
LINET + DR	4.5	4.2	5.0	6.4	3.7	5.0	5.5	7.3
RBS_P + DR	0.7	0.9	1.2	1.8	0.6	1.0	1.4	2.0
RBS_R + DR	3.5	4.5	6.4	7.2	3.2	4.8	7.0	8.1

Table 5
Relative Deviation from Optimum for RBS_P + DR Heuristic

		Low Var			High Var		
n	*R*	*D*=0.10	*D*=0.25	*D*=0.50	*D*=0.10	*D*=0.25	*D*=0.50
20	0.25	0.4	0.6	2.2	0.1	1.1	2.5
	0.50	0.5	0.9	1.9	0.9	1.0	1.7
	0.75	0.4	1.1	0.4	0.3	0.4	1.1
30	0.25	0.3	2.1	3.4	0.3	1.5	5.0
	0.50	0.7	1.6	4.4	0.5	2.2	4.2
	0.75	0.8	1.6	1.5	1.6	1.0	1.7

The DBS procedure provides very good results, but its computational requirements are only acceptable for small or medium-sized instances. The RBS_P heuristic is close to the DBS algorithm in solution quality and is significantly faster, solving even large instances within reasonable computation times. Therefore, this procedure is a good choice for medium and large-sized problems.

Conclusion

This paper considers the single-machine scheduling problem with earliness and tardiness penalties, release dates, and no unforced idle time. Considered are heuristics based on the beam search technique and classical beam search algorithms, as well

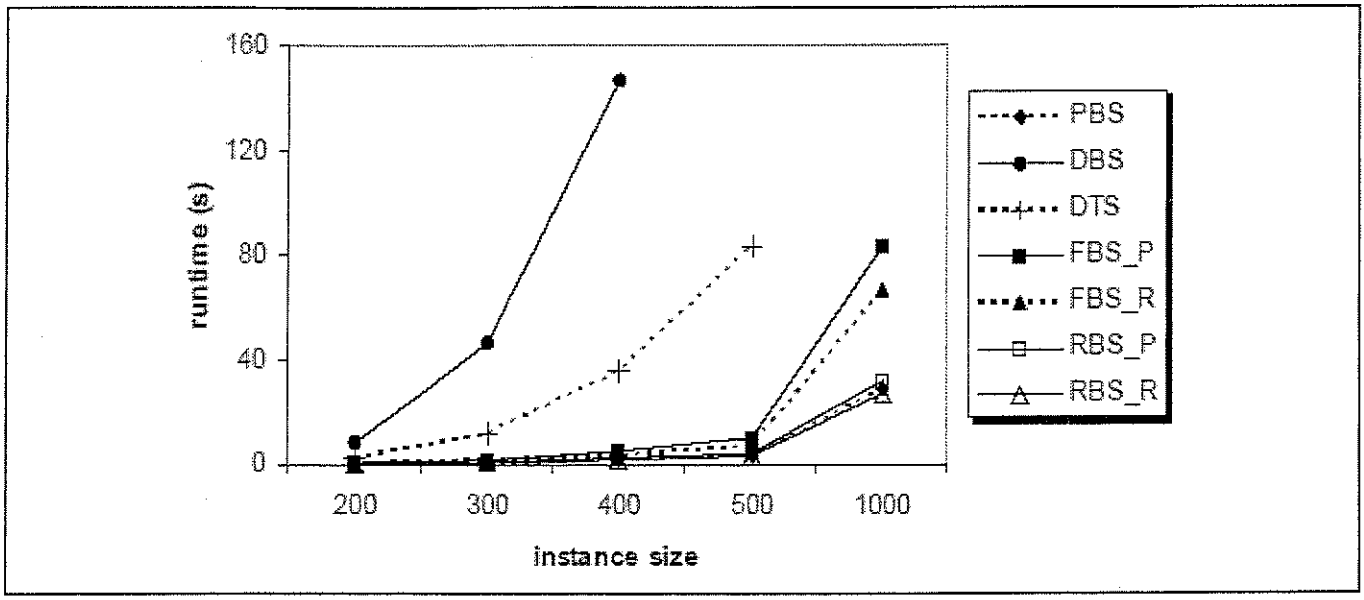

Figure 1
Run Times for Instances with Low Variability (in seconds)

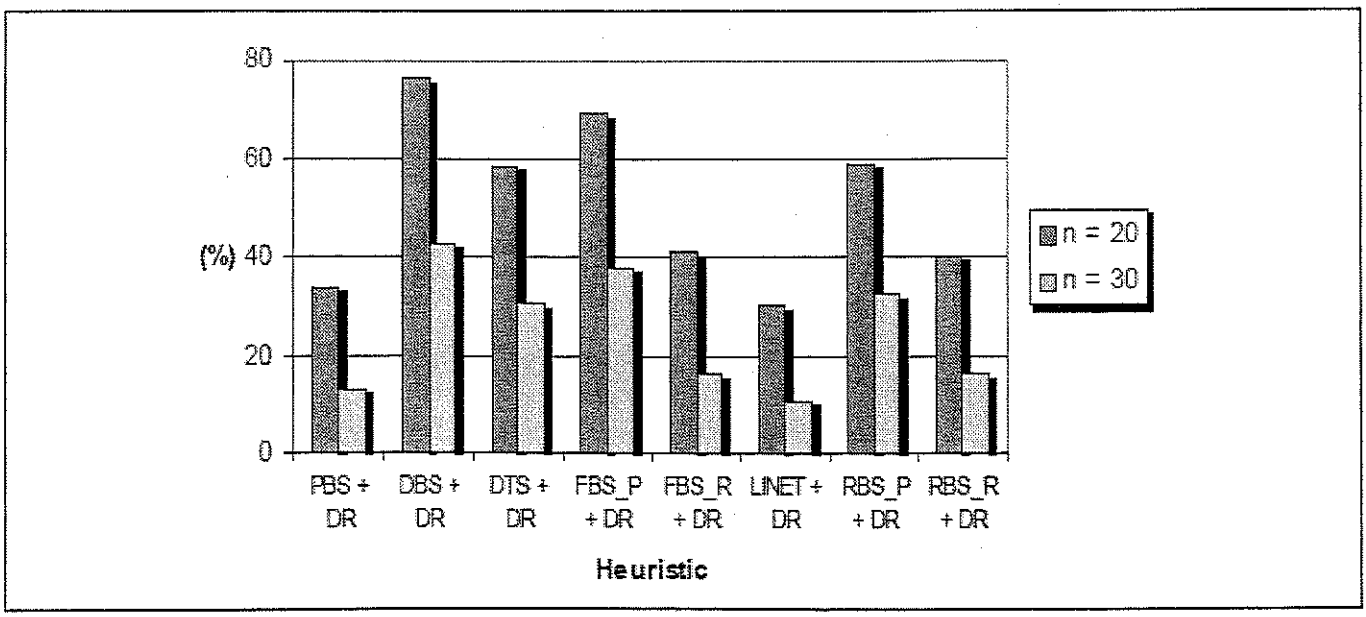

Figure 2
Percentage Number of Times a Heuristic Generates an Optimum Schedule for Instances with Low Variability

as filtered and recovering variants. Both priority evaluation functions and problem-specific properties were considered for the filtering step used in the filtered and recovering heuristics. The algorithms use several parameters whose value must be specified. Extensive computational tests were performed to determine the parameter values that provided the best balance between solution quality and computational effort. Also considered was using dominance rules to improve the solutions generated by the heuristics.

The computational results show that the use of the dominance rules is recommended because they can improve the solution quality, particularly for instances with a low processing time and penalty variability, and require little additional computational time. The algorithms with a priority evaluation function filtering procedure were superior to their rules-based counterparts in solution quality. The recovering beam search procedures clearly outperformed the filtered beam search algorithms because they provided better solutions with a lower computational effort. The DBS heuristic gives the best results, but it is computationally demanding and can be applied only to small or medium-sized instances. The RBS_P procedure is significantly faster and provides results that are quite close to the best in solution quality. Therefore, this procedure is recommended for medium and even large problems. The recovering beam search technique performed well and appears to achieve a good balance between computational efficiency and solution quality. These results confirm the potential of this recently introduced approach, and as a possible step for future research it certainly seems worthy to investigate its behavior on other problems.

References

Abdul-Razaq, T. and Potts, C.N. (1988). "Dynamic programming state-space relaxation for single machine scheduling." *Journal of the Operational Research Society* (v39, n2), pp141-152.

Akturk, M.S. and Ozdemir, D. (2000). "An exact approach to minimizing total weighted tardiness with release dates." *IIE Trans.* (v32, n11), pp1091-1101.

Akturk, M.S. and Ozdemir, D. (2001). "A new dominance rule to minimize total weighted tardiness with unequal release dates." *European Journal of Operational Research* (v135, n2), pp394-412.

Della Croce, F. and T'kindt, V. (2002). "A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem." *Journal of the Operational Research Society* (v53, n11), pp1275-1280.

Kanet, J.J. and Zhou, Z. (1993). "A decision theory approach to priority dispatching for job shop scheduling." *Production and Operations Mgmt.* (v2, n1), pp2-14.

Korman, K. (1994). "A pressing matter." *Video* (Feb. 1994), pp46-50.

Landis, K. (1993). "Group technology and cellular manufacturing in the Westvaco Los Angeles VH department." Project report in IOM 581. Los Angeles: School of Business, Univ. of Southern California.

Lenstra, J.K.; Rinnooy Kan, A.H.G.; and Bruckner, P. (1977). "Complexity of machine scheduling problems." *Studies in Integer Programming*, v1 of *Annals of Discrete Mathematics*, P.L. Hammer et al., eds. Amsterdam: North-Holland, pp343-362.

Li, G. (1997). "Single machine earliness and tardiness scheduling." *European Journal of Operational Research* (v96, n3), pp546-558.

Liaw, C.-F. (1999). "A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem." *Computers Operations Research* (v26, n7), pp679-693.

Lowerre, B.T. (1976). "The HARPY speech recognition system." PhD thesis. Pittsburgh: Carnegie-Mellon Univ.

Ow, P.S. and Morton, T.E. (1988). "Filtered beam search in scheduling." *Int'l Journal of Production Research* (v26, n1), pp35-62.

Ow, P.S. and Morton, T.E. (1989). "The single machine early/tardy problem." *Mgmt. Science* (v35, n2), pp177-191.

Ow, P.S. and Smith, S.F. (1988). "Viewing scheduling as an opportunistic problem-solving process." *Annals of Operations Research* (v12, n1), pp85-108.

Sabuncuoglu, I. and Bayiz, M. (1999). "Job shop scheduling with beam search." *European Journal of Operational Research* (v118, n2), pp390-412.

Sidney, J. (1977). "Optimal single-machine scheduling with earliness and tardiness penalties." *Operations Research* (v25, n1), pp62-69.

Valente, J.M.S. and Alves, R.A.F.S. (2003). "Heuristics for the early/tardy scheduling problem with release dates." Working Paper 129. Portugal: Faculdade de Economia do Porto.

Valente, J.M.S. and Alves, R.A.F.S. (2005). "An exact approach to early/tardy scheduling with release dates." *Computers Operations Research* (v32, n11), pp2905-2917.

Authors' Biographies

Jorge Miguel Silva Valente is an assistant professor of operations research with the Faculty of Economics, University of Porto, Portugal. He holds a PhD in management science and an MS in economics from the University of Porto. His current research interests include production scheduling, combinatorial optimization, heuristic techniques, and agent-based computational economics.

Rui Alberto Ferreira dos Santos Alves is an associate professor of operations research and operations management with the Faculty of Economics, University of Porto, Portugal. He holds a PhD in business administration and an MS in operations research from the University of Rochester, New York. His current research interests include production scheduling, queueing phenomena, and quality management.