Available online at www.sciencedirect.com
ScienceDirect
Computers & Industrial Engineering 55 (2008) 663–675
computers & industrial engineering
www.elsevier.com/locate/caie

Beam search heuristics for the single machine early/tardy scheduling problem with no machine idle time

Jorge M.S. Valente *

LIAAD, Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal

Received 24 October 2007; received in revised form 29 November 2007; accepted 10 February 2008
Available online 16 February 2008

Abstract

In this paper, we present beam search heuristics for the single machine early/tardy scheduling problem with job-independent penalties, and no machine idle time. These heuristics include priority and detailed classic beam search algorithms, as well as filtered and recovering procedures. Three dispatching rules are considered as evaluation functions, in order to analyse the effect of different rules on the performance of the beam search heuristics.

The computational results show that the performance of the beam search procedures does improve with the quality of the dispatching rule. The detailed and recovering algorithms clearly outperform the best existing heuristic, and the improvement is particularly higher for the more difficult instances. The detailed beam search algorithm provides the best performance, and is recommended for small to medium size instances. For larger instances, however, this algorithm requires excessive computation times. The recovering beam search procedure is computationally more efficient, and is then the heuristic of choice for medium to large instances.

Keywords: Scheduling; Single machine; Early/tardy; Job-independent penalties; Heuristics; Beam search

1. Introduction

In this paper, we consider a single machine earliness/tardiness scheduling problem with job-independent penalties, and no machine idle time. Single machine scheduling environments actually occur in several practical operations (for a recent example in the chemical industry, see Wagner, Davis, & Kher, 2002). Also, the performance of many production systems is often determined by the quality of the schedules for a single bottleneck machine. Furthermore, the analysis of single machine models provides results and insights that are often useful for more complex scheduling environments.

Scheduling with earliness and tardiness costs has received considerable and increasing attention from the scheduling community, due to its practical importance and relevance. In fact, early/tardy scheduling problems are compatible with the concepts of just-in-time production and supply chain management, which have been

* Tel.: +351 22 557 11 00; fax: +351 22 550 50 50.
E-mail address: jvalente@fep.up.pt

0360-8352/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2008.02.005

adopted by many organisations. Indeed, these production strategies view both early and tardy deliveries as undesirable.

In this paper, we consider job-independent earliness and tardiness penalties. Job-independent earliness penalties are appropriate in production settings where the jobs are identical, or quite similar (e.g., parts required for stages further down the production or assembly line). A job-independent tardiness penalty is also adequate when the jobs are similar, or when the company assigns the same importance to all customers.

The assumption that no machine idle time is allowed is also appropriate for many production settings. Indeed, idle time should be avoided when the machine has limited capacity or high operating costs. This assumption is also justified when starting a new production run involves high setup costs or times. Some specific examples of production settings where the no idle time assumption is appropriate have been given by Kor-man (1994) and Landis (1993).

Formally, the problem can be stated as follows. A set of n independent jobs $\{J_1,J_2,\ldots,J_n\}$ has to be scheduled on a single machine that can handle at most one job at a time. The machine is assumed to be continuously available from time zero onwards, and preemptions are not allowed. Job $J_j,j=1,2,\cdots,n$, requires a processing time p_j and should ideally be completed on its due date d_j. Also, let h and w denote the job-independent earliness and tardiness penalties, respectively. For a given schedule, the earliness and tardiness of J_j are respectively defined as $E_j=\max\{0,d_j-\ C_j\}$ and $T_j=\max\{0,C_j-\ d_j\}$, 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 $\sum_{j=1}^{n}(hE_j+wT_j)$, subject to the constraint that no machine idle time is allowed.

This problem has been previously considered by Szwarc (1993), Azizoglu, Kondakci, and Kirca (1991) and Valente (2007). Szwarc (1993) developed time-dependent precedence relations that hold for adjacent jobs in an optimal schedule, and presented a branching scheme based on these adjacent ordering conditions. Azizoglu et al. (1991) actually considered the equivalent problem $\sum_{j=1}^{n}((1-q)E_j+qT_j)$, with $0<q<1$. They proposed a lower bounding procedure, as well as a branch-and-bound algorithm. Valente (2007) presented and analysed several dispatching heuristics, and also proposed an improvement procedure that incorporates Szwarc's adjacent ordering conditions.

The single machine problem with job-dependent penalties $\sum_{j=1}^{n}(h_jE_j+w_jT_j)$ and no idle time has also been considered by several authors, and both exact and heuristic approaches have been proposed. Among the exact approaches, lower bounds and branch-and-bound algorithms were presented by Abdul-Razaq and Potts (1988), Li (1997), Liaw (1999) and Valente and Alves (2005c). Among the heuristics, several dispatching rules and beam search algorithms were presented by Ow and Morton (1989), Valente and Alves (2005b) and Valente and Alves (2005a). Li (1997) also presented a neighbourhood search algorithm, while Valente, Gonçalves, and Alves (2006) proposed several genetic algorithms.

A large number of papers have been published on scheduling models with earliness and tardiness costs. Baker and Scudder (1990) provide an excellent survey of the initial work on early/tardy scheduling. A recent survey of multicriteria scheduling problems is given in Hoogeveen (2005). This survey considers and reviews problems with earliness and tardiness penalties. Also, Kanet and Sridharan (2000) give a review of scheduling models with inserted idle time that complements our focus on a problem with no machine idle time.

In this paper, we present several heuristic algorithms based on the beam search technique. The classic beam search procedure was first used in artificial intelligence problems by Lowerre (1976) and Rubin (1978). Two variations of the traditional beam search algorithm have since been developed. Ow and Morton (1988) and Ow and Morton (1989) proposed a technique denoted by filtered beam search. Recently, the recovering beam search approach was introduced by Della Croce and T'kindt (2002) and Della Croce et al. (2004). Beam search algorithms have been applied to several combinatorial optimization problems, particularly in the scheduling field. Some recent applications of beam search to scheduling include (Della Croce & T'kindt, 2002; Della Croce, Ghirardi, & Tadei, 2004; Valente & Alves, 2005a; Ghirardi & Potts, 2005; Esteve, Aubijoux, Chartier, & T'kindt, 2006).

The proposed heuristics include priority and detailed classic beam search algorithms, as well as the more recent filtered and recovering variants. Extensive preliminary tests are first performed in order to determine adequate values for the parameters required by the beam search procedures. The beam search heuristics require evaluation functions, which are typically derived from dispatching rules. We consider three dispatch-

ing rules, and their effect on the performance of the beam search algorithms is also analysed in these preliminary tests. The best-performing beam search versions are then compared with the best of the existing heuristics.

The remainder of this paper is organised as follows. In Section 2, we describe the beam search technique and its several variations, and present the proposed heuristic procedures. The computational results are given in Section 3. Finally, we provide some concluding remarks in Section 4.

2. The beam search heuristics

Beam search is a heuristic method for solving combinatorial optimization problems. It consists of a truncated branch-and-bound procedure in which only the most promising nodes at each level of the search tree are kept for further branching. The remaining nodes are pruned off, and no backtracking is performed, so the running time is polynomial in the problem size. In the following subsections, we present the classic beam search technique, as well as the filtered and recovering variations. The proposed beam search algorithms are then described, and their implementation details are provided.

2.1. Classic beam search

The classic or traditional beam search approach consists of a truncated branch-and-bound procedure with a breadth-first strategy in which 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*. The remaining nodes are discarded, and backtracking is not allowed. Therefore, the performance and effectiveness of a beam search algorithm greatly depends on the node evaluation process. Two different types of evaluation functions have been used in classic beam search algorithms: *priority evaluation functions* and *total cost evaluation functions*.

Priority evaluation functions simply calculate an urgency rating for the last job added to the current partial sequence. Typically, this is done by using the priority index of a dispatching heuristic. Total cost evaluation functions, on the other hand, calculate an estimate of the minimum total cost of the best solution that can be reached from the current node. This is usually done by using a dispatching rule to schedule the remaining jobs, in order to complete the existing partial sequence. The priority evaluation functions only consider the next decision to be made (i.e., the next job to schedule), and therefore have a local view of the problem. The total cost evaluation functions, however, have a global view, since they project from the current partial schedule to a complete solution in order to obtain an estimate of the total cost.

The priority evaluation functions are usually context-dependent, and this can pose a slight problem. In fact, the priority index that is used to calculate the urgency rating of the last scheduled job quite often depends on the current partial schedule, particularly on the current time. Different nodes, even if at the same tree level, correspond to different partial sequences, and may then have different completion times. Therefore, the urgency ratings are context-dependent, which means that the priorities calculated for the offspring of one node cannot be legitimately compared with those obtained from the expansion of another node. This problem can be solved by initially selecting the best β offspring of the root node. At lower levels of the search tree, only the best descendant of each beam node is kept for further branching, so the number of beam nodes is kept at β. The total cost evaluation functions are not affected by this problem. Indeed, total cost estimates are context-independent, and can be compared for all offspring nodes.

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

Priority beam search:

Step 1. Initialization:
Set $B = \varnothing$, $C = \varnothing$.
Branch n_0, generating the corresponding children.
Calculate the priority of the last scheduled job for each child node.
Select the best 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 $C = \varnothing$.
Step 3. Stopping condition:
If the nodes in B are leaf (i.e., they hold a complete sequence), select the node with the lowest total cost as the best sequence found and stop.
Otherwise, go to step 2.

Detailed beam search:

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

2.2. Filtered and recovering beam search

The priority and total cost evaluation functions have opposite advantages and weaknesses. On the one hand, the priority evaluation functions are computationally cheap, but are rather crude and potentially inaccurate, so they may lead to the elimination of good nodes. The total cost evaluation functions, on the other hand, are more accurate, but require a much higher computational effort. The filtered and recovering beam search algorithms combine crude and accurate evaluations, in order to try to achieve a high quality evaluation, without requiring excessive computation times. This is done by means of a two-stage approach. First, the filtered and recovering algorithms apply a computationally cheap *filtering step*. In this step, a crude evaluation is performed, in order to select only a reduced number of the offspring of each beam node. The chosen nodes are then accurately evaluated, and the best β are kept for further branching.

Two types of filtering step have been used. Ow and Morton (1988) and Ow and Morton (1989) proposed using a priority evaluation function to calculate an urgency rating for each offspring. Then, the best α children of each beam node are selected for accurate evaluation; α is the so-called *filter width*. Recently, Della Croce and T'kindt (2002) and Della Croce et al. (2004) introduced the second type of filtering phase, in conjunction with the development of the recovering beam search algorithm. In this approach, problem-dependent dominance conditions (denoted as valid dominance conditions), when available, are applied together with so-called pseudo-dominance conditions (which hold in a heuristic context only). Whenever one of these conditions holds for a given node, that node is eliminated. The priority function filtering procedure was then originally applied in a filtered beam search algorithm, while the dominance conditions approach was initially used in a recovering beam search heuristic. Nevertheless, either of these two filtering procedures can be used in both filtered and recovering algorithms.

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

Second, the RBS procedure includes a so-called *recovering phase*. This phase is performed after the accurate evaluation of the nodes that passed the filtering step, and it selects the β nodes that will be kept for further branching. In this phase, the nodes are considered in non-decreasing order of their evaluation function values. For each candidate node, we then check if its current partial schedule σ is dominated by another partial schedule $\overline{\sigma}$ with the same level of the search tree. Typically, this is done by applying neighbourhood operators. If a better partial schedule $\overline{\sigma}$ does exist, then σ is replaced by $\overline{\sigma}$. If the possibly modified node is not already in the set of beam nodes, then the node is added to B. This process is repeated until either β nodes have been selected, or no additional candidate node remains.

Classic and filtered recovering beam search algorithms cannot recover from wrong decisions: if a node leading to the optimal solution is eliminated, there is no way to reach that solution afterwards. Wider beam and/or filter widths reduce the risk of eliminating a node that would ultimately lead to the optimal solution, but at the cost of increased computational effort. The recovering phase tries to overcome this problem, and often allows the RBS procedure to recover from previous incorrect decisions. Note also that, in the recovering phase, a partial schedule can only be replaced by another partial schedule with the same tree level. Therefore, 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. In the RBS algorithm, let n_{best} and UB_{best} denote the current best node and the current best upper bound, respectively.

Filtered beam search:

Step 1. Initialization:
Set $B = \{n_0\}$ and $C = \varnothing$.
Step 2. Filtering step:
For each node in B:
(a) Branch the node, generating the corresponding children.
(b) Add to C all the child nodes that are not eliminated by the filtering procedure.
Set $B = \varnothing$.
Step 3. Node selection:
Calculate an upper bound on the optimal solution value for all nodes in C.
Select the best β nodes in C and add them to B.
Set $C = \varnothing$.
Step 4. Stopping condition:
If the nodes in B are leaf, select the node with the lowest total cost as the best sequence found and stop.
Otherwise, go to step 2.

Recovering beam search:

Step 1. Initialization:
Set $B = \{n_0\}$, $C = \varnothing$, $n_{best} = \varnothing$and $UB_{best} = \infty$.
Step 2. Filtering step:
For each node in B:
(a) Branch the node, generating the corresponding children.
(b) Add to C all the child nodes that are not eliminated by the filtering procedure.
Set $B = \varnothing$.
Step 3. Accurate evaluation:
For all nodes n_k,$k = 1, \ldots, |C|$ in C:
(a) Calculate a lower bound LB_k and an upper bound UB_k on the optimal solution value of node n_k.

(b) Compute the evaluation function $V_k = (1 - \gamma)\ LB_k + \gamma UB_k$.
(c) If $UB_k < UB_{best}$, set $n_{best} = n_k$ and $UB_{best} = UB_k$.

Step 4. Recovering step:
Sort all nodes in C in non-decreasing order of the evaluation function value V_k.
Set $k = 1$.
While $|B| < \beta$ and $k \leqslant |C|$:
(a) Let σ represent the partial solution associated with the current node n_k.
(b) Search for a partial solution $\overline{\sigma}$ that dominates σ by means of neighbourhood operators.
(c) If $\overline{\sigma}$ is found, set $\sigma = \overline{\sigma}$.
(d) If $n_k \notin B$
(i) Set $B = B \cup \{n_k\}$.
(ii) If $UB_k < UB_{best}$, set $n_{best} = n_k$ and $UB_{best} = UB_k$.
(e) Set $k = k + 1$.

Step 5. Stopping condition:
If the nodes in B are leaf, stop with n_{best} and UB_{best} as the best node and lowest total cost found, respectively.
Otherwise, go to step 2.

2.3. Implementation details

In this paper, we consider both priority and detailed beam search algorithms, and also filtered and recovering beam search procedures. In order to apply these algorithms to the single machine early/tardy scheduling problem with job-independent penalties, it is necessary to specify their main components, such as branching scheme, evaluation function, filtering procedure and recovering step. In the following, the implementation details of the proposed beam search heuristics are provided.

2.3.1. Branching scheme

The branching scheme is identical for all algorithms. It 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 that will be scheduled in position l.

2.3.2. Dispatching rules

The beam search procedures require a dispatching rule, in order to calculate an upper bound and/or to provide a priority evaluation function. Three alternative dispatching heuristics were considered, in order to analyse the effect of using different rules on the performance of the beam search algorithms. These three heuristics are the EDD, LES_AS and LINET_vk dispatching rules analysed in Valente (2007). The EDD rule is not only quite well-known, but also widely used in practice for problems with due dates. The LES_AS heuristic combines the EDD rule with two other simple and commonly used heuristics, and provides a significant improvement over these simpler rules. The LINET_vk heuristic was the best-performing of all the dispatching rules considered in Valente (2007). Therefore, we considered three versions (corresponding to these three rules) for each type of beam search procedure. In the following, the LES_AS and LINET_vk rules will be denoted simply as LES and LINET.

2.3.3. Priority beam search

The priority beam search algorithms require a priority evaluation function, in order to calculate an urgency rating for the last scheduled job. For each of the three versions of the PBS procedure, the priority function is given by the priority index of the appropriate dispatching rule (EDD, LES or LINET).

2.3.4. Detailed beam search

The detailed beam search algorithms require a total cost evaluation function, i.e., an upper bounding procedure. This procedure is used, for a given node, to schedule the remaining unscheduled jobs, thereby com-

pleting the existing partial schedule and obtaining an upper bound on the total cost of the node. For each DBS version, the upper bounding procedure is provided by the appropriate dispatching heuristic.

2.3.5. Filtered beam search

The filtered beam search algorithms require filtering and upper bounding procedures. The upper bounding procedure is provided by the relevant dispatching rule, just as previously described for the DBS algorithms. The filtering step uses a priority evaluation filter. Therefore, a priority evaluation function is needed, in order to calculate an urgency rating for the offspring of a given node. This priority evaluation function is provided by the priority index of the appropriate dispatching rule, as previously mentioned for the PBS procedures.

2.3.6. Recovering beam search

The recovering beam search algorithms require a filtering procedure, upper and lower bounding procedures for the accurate evaluation step, and an improvement procedure for the recovering phase. The filtering and upper bounding procedures are identical to those used in the FBS algorithms. The lower bound is obtained with the method proposed by Azizoglu et al. (1991). 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 set equal to the sum of the cost of the existing partial schedule and the lower bound for the unscheduled jobs. Finally, for the recovering phase we used the improvement procedure proposed in Valente (2007). This procedure uses the dominance conditions developed by Szwarc (1993), in order to generate a sequence that cannot be further improved through adjacent swaps.

2.3.7. Improvement step

In the next section, we compare the beam search heuristics with the best existing approach. In Valente (2007), the best performance was achieved by first using the LINET dispatching rule, and then applying the proposed improvement procedure. Therefore, the beam search heuristics were then compared with the LINET rule followed by the dominance conditions improvement procedure. Consequently, the adjacent ordering conditions procedure was also applied, as an improvement step, to the beam search heuristics, i.e., this procedure is used to improve the schedule generated by the beam search algorithms. This improvement step is actually redundant for the RBS heuristics since, as we previously described, the dominance conditions procedure is already applied during the recovering phase.

3. Computational results

In this section, we first present the set of test problems used in the computational tests. Then, the preliminary computational experiments are described. The purpose of these initial experiments is twofold. First, they were performed in order to determine adequate values for the several parameters required by the beam search heuristics. Second, these tests were used to analyse the performance of the beam search procedures under the EDD, LES and LINET rules, in order to select the best-performing versions. Finally, the beam search procedures are compared with the best existing heuristic. Throughout this section, and in order to avoid excessively large tables, we will sometimes present results only for some representative cases.

3.1. 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 as follows. For each job J_j, an integer processing time p_j was generated from one of the two uniform distributions [1, 10] and [1, 100], to create low (L) and high (H) variability, respectively. An integer earliness penalty h and an integer tardiness penalty w were also generated from one of these two uniform distributions. The tardiness penalty weight $\phi = w/(h + w)$ was set at 0.1, 0.3, 0.5, 0.7 and 0.9. The earliness and tardiness penalties are then randomly generated in such a way that the desired ratio ϕ is obtained. For each job J_j, an integer due date d_j is generated from the uniform distribution $[P(1 - T - R/2), P(1 - T + R/2)]$, where P is the sum of the processing times of all jobs, T is the tardiness factor, set at 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0, and R is the range of due dates, set at 0.2, 0.4, 0.6 and 0.8.

For each combination of problem size n, processing time and penalty variability (var), ϕ, T and R, 50 instances were randomly generated. Therefore, a total of 6000 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 −2.8 GHz personal computer. Due to the large computational times that would be required, the detailed beam search algorithm was only used on instances with up to 100 jobs, while the filtered and recovering procedures were not applied to the 500 job instances.

3.2. Preliminary tests

The preliminary computational tests are described in this section. These initial experiments were used, on the one hand, to determine appropriate values for the beam search parameters and, on the other hand, to select the best-performing of the three dispatching rules. These preliminary tests were conducted on a separate problem set. This set included instances with 25, 50, 75 and 100 jobs, and contained 5 instances for each combination of instance size, processing time and penalty variability, ϕ, T and R. The instances in this smaller test set were generated randomly just as previously described for the full problem set.

3.2.1. Parameter adjustment

Extensive tests were conducted in order to select adequate values for the beam width, filter width and upper bound weight parameters. There is a trade-off between solution quality and computation time for the beam and filter width parameters. In fact, increasing the value of these parameters usually improves the objective function value, but at the cost of increased computational effort. The following values were considered for the beam width, filter width and upper bound weight parameters:

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

The several beam search versions were applied to the test instances for all combinations of the relevant parameter values. We then calculated and plotted the mean objective function values and runtimes. A thorough analysis of these charts, as well as the underlying data, revealed the usual behaviour in beam search algorithms: the computation time increases linearly with α and β, while the solution quality improves, but with diminishing returns. Therefore, increasing the value of these two parameters eventually leads to little or virtually no improvement in the objective function value. An example chart is given in Fig. 1. This chart presents the average objective function value and runtime for the DBS algorithm (with the LINET rule), for instances with 25 jobs and high variability.

We then selected the parameter values that seemed to provide the best trade-off between solution quality and computation time. For all beam search versions, a value of 3 was chosen for both the beam and filter width parameters. In the RBS procedures, the upper bound weight was set at 0.6.

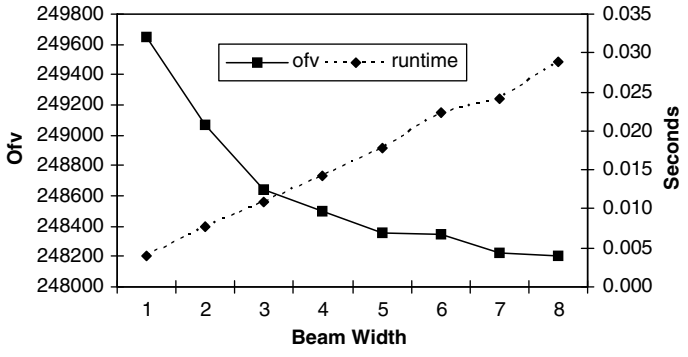

Fig. 1. Objective function value and runtime for the DBS heuristic with the LINET rule on instances with 25 jobs and high variability.

Table 1
Preliminary results

Var	Heur	Rule	$n = 25$		$n = 50$		$n = 75$		$n = 100$	
			%imp	%best	%imp	%best	%imp	%best	%imp	%best
L	PBS	EDD	–	1.33	–	1.67	–	2.67	–	1.83
		LES	12.55	8.33	13.29	10.17	12.81	9.33	13.70	13.17
		LINET	26.73	92.00	26.34	88.17	26.39	88.00	26.95	85.00
	DBS	EDD	–	9.50	–	7.67	–	8.67	–	7.00
		LES	0.23	10.83	0.67	11.83	0.75	14.33	1.15	17.17
		LINET	6.12	86.00	6.96	80.83	6.67	77.00	7.28	76.00
	FBS	EDD	–	1.67	–	1.67	–	2.67	–	2.33
		LES	10.30	14.50	13.53	16.83	13.19	15.00	13.79	19.33
		LINET	17.42	90.33	20.21	81.67	21.10	82.33	23.01	78.33
	RBS	EDD	–	39.17	–	37.00	–	45.17	–	45.00
		LES	1.98	45.00	0.81	27.17	−0.05	21.67	0.22	19.67
		LINET	3.03	84.50	0.56	65.83	−0.21	56.50	0.67	52.83
H	PBS	EDD	–	2.17	–	2.00	–	2.17	–	1.83
		LES	13.92	8.00	13.97	7.17	13.76	9.00	15.86	10.50
		LINET	28.38	90.33	29.60	90.83	29.50	88.83	29.99	87.67
	DBS	EDD	–	10.83	–	9.00	–	10.17	–	10.17
		LES	−0.81	7.67	−0.32	10.17	−0.51	13.67	0.33	14.67
		LINET	6.85	85.50	7.51	81.17	7.29	76.33	7.47	75.17
	FBS	EDD	–	2.50	–	1.67	–	2.17	–	1.67
		LES	10.11	12.33	12.65	14.17	13.46	14.67	15.66	16.33
		LINET	18.77	89.67	23.09	84.33	24.60	83.17	26.16	82.00
	RBS	EDD	–	36.33	–	34.00	–	38.50	–	41.17
		LES	0.94	38.83	0.64	21.67	0.09	16.50	0.64	16.50
		LINET	3.34	83.33	2.73	69.00	1.67	59.67	1.62	50.17

3.2.2. Rule selection

The performance of the EDD, LES and LINET rules was also analysed in the preliminary tests, in order to evaluate their impact on the effectiveness of the beam search procedures, and determine the best-performing rule. In Table 1 we present, for each beam search algorithm, the average of the relative improvements in objective function value over the EDD rule (%imp), as well as the percentage number of times a rule achieves the best objective function value found when compared with the other rules (%best). More specifically, the relative improvement over the EDD rule is calculated as (edd_ofv − rule_ofv)/ edd_ofv × 100, where edd_ofv and rule_ofv are the objective function values obtained by the EDD rule and the appropriate rule (LES or LINET), respectively. These values are omitted for the EDD rule, since they would necessarily be equal to 0.

The results given in Table 1 show that the performance of the beam search algorithms improves with the quality of the dispatching rule that is used to provide the priority and/or total cost evaluation functions. Indeed, the LINET rule provides the best results among the three alternative dispatching heuristics, while the LES procedure usually outperforms the simpler EDD rule. In fact, the LES and LINET rules give a (sometimes quite) large relative improvement in objective function value. Also, these rules provide the best results for a larger number of instances. The LINET rule, in particular, quite often gives the best results for over 80% of the test instances.

The relative improvement provided by the LES and LINET rules is much higher for the PBS (which relies only on a priority evaluation) and FBS (which uses both priority and detailed evaluations) algorithms. Indeed, the LINET rule gives a quite large average relative improvement of 20–30% for these beam search procedures. The improvement is smaller for the DBS algorithms, which use only a detailed evaluation, but the LINET rule

still provides an improvement in objective function value of about 7%. Therefore, it certainly seems that high quality rules should be used to provide priority evaluation functions and upper bounding procedures for the single machine early/tardy problem with job-independent penalties.

The objective function values given by the three rules were closer for the RBS algorithm, even though the more sophisticated LES and LINET heuristics were still superior to the EDD rule. This is most likely due to the recovering phase, which can correct previous wrong decisions. Thus, incorrect choices made by an inferior rule can later be corrected in the recovering phase, and so the results provided by the several rules are closer. The LINET rule was then selected, since it provided the best results among the three dispatching heuristics. In the following sections, we will therefore present results only for the beam search versions that use the LINET rule.

3.3. Heuristic results

In this section, we compare the beam search algorithms with the best existing heuristic, namely the LINET dispatching rule. As previously mentioned, the dominance conditions improvement procedure proposed in Valente (2007) is used as an improvement step. Therefore, this procedure is used to improve the schedules generated by the heuristics.

In Table 2, we provide the average of the relative improvements in objective function value over the LINET procedure (%imp), as well as the percentage number of times each heuristic achieves the best result when compared with the other heuristics (%best). The relative improvement over the LINET procedure is calculated as (linet_ofv − heur_ofv)/ linet_ofv × 100, where heur_ofv and linet_ofv are the objective function values of the appropriate heuristic and the LINET dispatching rule, respectively. The relative improvement values are omitted for the LINET heuristic, since they are necessarily equal to 0. Table 3 gives the percentage number of times the beam search algorithms perform better (<), equal (=) or worse (>) than the LINET dispatching rule.

The best results are given by the DBS procedure, which is clearly superior to the LINET dispatching rule. Indeed, the DBS algorithm not only provides a 2–4% average relative improvement over the LINET heuristic, but also gives better results for 60–70% of the instances. Also, the DBS procedure achieves the best results for a quite large percentage of the test instances. The RBS heuristic also clearly outperforms the LINET dispatching rule. In fact, the RBS procedure gives a relative improvement that usually ranges from 1% to 3%. Moreover, the RBS heuristic provides better (worse) results than the LINET rule for a large (small) number of instances. The FBS procedure is superior to the LINET heuristic for the high variability instances. When the variability is low, however, the FBS heuristic fails to improve on the results of the LINET dispatching rule. The PBS algorithm performs poorly, since it is usually outperformed by the LINET rule.

The beam search algorithms perform comparatively better when the processing time and penalty variability is high. Indeed, both the relative improvement and the number of instances for which they provide a better objective function value are larger for the high variability instances. The average relative improvement over

Table 2
Heuristic results – relative improvement and percentage of best results

Var	Heur	$n = 25$		$n = 50$		$n = 100$		$n = 250$	
		%imp	%best	%imp	%best	%imp	%best	%imp	%best
L	LINET	–	44.80	–	31.65	–	22.30	–	51.95
	PBS	−0.58	42.50	−1.65	27.78	−2.31	17.62	−2.63	12.62
	DBS	2.34	80.23	2.77	76.30	2.80	79.52	–	–
	FBS	1.47	66.82	−0.04	45.35	−2.10	33.33	−2.74	34.15
	RBS	2.16	79.27	1.13	52.52	0.41	33.02	0.36	70.53
H	LINET	–	34.18	–	19.70	–	10.82	–	13.77
	PBS	0.42	34.37	0.01	19.32	−0.18	10.33	−0.29	10.30
	DBS	3.27	70.62	3.90	59.62	4.04	63.77	–	–
	FBS	2.46	57.60	1.71	34.72	0.92	23.33	0.20	45.03
	RBS	3.25	77.43	3.07	53.70	2.02	31.97	0.73	56.93

Table 3
Heuristic results – comparison of objective function values

Var	n	PBS			DBS			FBS			RBS		
		$<$	$=$	$>$	$<$	$=$	$>$	$<$	$=$	$>$	$<$	$=$	$>$
L	25	5.4	80.4	14.2	50.9	43.5	5.7	46.7	47.5	5.8	50.9	46.5	2.6
	50	3.3	64.4	32.3	63.0	28.4	8.6	48.6	33.9	17.6	51.2	42.0	6.8
	100	2.3	45.9	51.8	72.5	17.8	9.8	38.9	23.1	38.1	37.0	52.0	11.0
	250	2.1	31.2	66.8	–	–	–	32.2	10.0	57.8	33.2	51.0	15.8
H	25	7.6	89.8	2.6	60.7	31.2	8.1	58.4	34.4	7.3	62.9	34.2	3.0
	50	3.8	90.5	5.7	71.9	15.9	12.1	69.7	19.2	11.1	73.8	21.2	5.0
	100	2.1	85.3	12.6	79.8	6.2	14.0	76.8	10.0	13.2	74.9	15.4	9.7
	250	1.7	69.5	28.8	–	–	–	76.3	6.9	16.8	65.5	15.4	19.2

Table 4
Relative improvement over the LINET heuristic, for the RBS procedure and instances with 50 jobs

ϕ	T	Low var				High var			
		$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$
0.1	0.0	0.00	0.01	0.01	0.03	0.01	0.01	0.01	0.00
	0.2	0.29	0.22	0.32	0.07	1.04	0.90	0.42	0.20
	0.4	0.95	2.20	2.98	3.85	1.90	4.86	8.53	8.75
	0.6	1.74	0.99	2.38	0.72	4.10	10.11	7.32	2.05
	0.8	0.70	0.23	0.09	0.07	1.16	0.27	0.11	0.13
	1.0	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.03
0.3	0.0	0.01	0.00	0.01	0.01	0.00	0.02	0.03	0.04
	0.2	0.67	0.49	0.51	0.13	1.33	0.80	0.31	0.49
	0.4	1.09	3.33	2.65	2.66	3.30	6.43	9.14	11.39
	0.6	1.80	1.27	0.74	0.47	4.44	6.63	6.34	1.22
	0.8	0.80	0.26	0.10	0.10	1.06	0.33	0.14	0.08
	1.0	0.00	0.01	0.01	0.02	0.01	0.02	0.02	0.02
0.5	0.0	0.00	0.00	0.03	0.06	0.01	0.03	0.06	0.03
	0.2	1.04	0.87	0.17	0.43	2.85	1.83	1.02	0.67
	0.4	2.27	2.58	4.28	3.47	4.19	8.14	10.58	9.84
	0.6	1.66	1.14	0.56	1.39	4.51	7.28	3.90	1.66
	0.8	1.11	0.37	0.14	0.05	1.27	0.46	0.28	0.13
	1.0	0.00	0.01	0.01	0.01	0.01	0.03	0.01	0.03
0.7	0.0	0.02	0.03	0.05	0.07	0.05	0.08	0.07	0.12
	0.2	2.24	1.44	0.50	0.59	5.60	3.70	2.30	1.33
	0.4	4.02	3.77	5.75	4.11	6.99	11.06	15.69	15.38
	0.6	1.68	1.10	1.35	1.37	4.48	5.16	4.39	2.06
	0.8	1.29	0.36	0.14	0.15	1.35	0.50	0.36	0.20
	1.0	0.00	0.01	0.02	0.02	0.01	0.01	0.04	0.04
0.9	0.0	0.03	0.09	0.14	0.18	0.19	0.15	0.38	0.51
	0.2	7.74	5.94	2.28	1.31	14.35	12.54	6.26	3.24
	0.4	4.78	5.39	7.46	6.39	9.39	17.08	20.69	21.78
	0.6	1.29	1.03	0.90	1.54	4.56	5.88	2.10	1.74
	0.8	1.16	0.39	0.16	0.08	1.23	0.43	0.25	0.21
	1.0	0.00	0.01	0.02	0.03	0.02	0.03	0.05	0.03

the LINET rule also increases with the instance size for the DBS heuristic. For the other beam search procedures, however, the relative improvement is decreasing in the instance size.

The effect of the ϕ, T and R parameters on the relative improvement over the LINET dispatching rule is presented in Table 4, for the RBS procedure and instances with 50 jobs. The relative improvement provided by

Table 5
Heuristic runtimes (in seconds)

Var	Heur	$n = 25$	$n = 50$	$n = 75$	$n = 100$	$n = 250$
L	LINET	0.000	0.000	0.001	0.001	0.005
	PBS	0.002	0.007	0.018	0.038	0.517
	DBS	0.011	0.141	0.690	2.122	–
	FBS	0.003	0.018	0.052	0.118	2.098
	RBS	0.007	0.035	0.099	0.220	3.501
H	LINET	0.000	0.000	0.001	0.001	0.006
	PBS	0.002	0.008	0.022	0.046	0.530
	DBS	0.011	0.139	0.673	2.120	–
	FBS	0.003	0.018	0.056	0.132	2.104
	RBS	0.007	0.037	0.108	0.243	3.958

the beam search procedures is quite minor when $T = 0.0$ or $T = 1.0$. However, the improvement given by the beam search heuristics increases quite significantly as the tardiness factor assumes more intermediate values. Indeed, for some parameter combinations, the RBS procedure actually provides an improvement of about 10–20% (4–7%) for the instances with high (low) processing time and penalty variability.

These results are to be expected, since the early/tardy problem is likely to be easier for the extreme values of the tardiness factor T. Indeed, when $T = 0.0$ ($T = 1.0$), most jobs will be early (tardy), so the problem becomes easier. For the intermediate values of T, however, there is a greater balance between the number of early and tardy jobs, and the problem tends to become harder. Therefore, the beam search heuristics provide a minor improvement for the easier instances, where all the heuristics are likely to perform rather well. For the more difficult instances, however, the beam search procedures significantly outperform the LINET dispatching rule.

The heuristic runtimes (in seconds) are presented in Table 5. The DBS procedure is computationally demanding, and consequently it can only be used for small or medium size instances. The FBS and RBS algorithms are more efficient, and can be applied to larger instances. The PBS procedure is much faster, but the LINET dispatching rule is even more computationally efficient, and it provides results of similar or superior quality. For the small and medium size instances, the DBS procedure is then the heuristic of choice. The RBS procedure is more efficient, and is recommended for somewhat larger instances. For the quite large instance sizes, however, only a dispatching rule can provide results in reasonable computation times.

4. Conclusion

In this paper, we presented several beam search heuristics for the single machine early/tardy scheduling problem with job-independent penalties, and no machine idle time. These heuristics included priority and detailed classic beam search procedures, as well as the filtered and recovering variants. The beam search approach requires evaluation functions, which are typically provided by dispatching rules. We considered three dispatching rules, in order to evaluate the impact of different rules on the performance of the beam search heuristics.

Extensive preliminary computational tests were performed in order to determine appropriate values for the beam search parameters. These preliminary tests also show that the beam search performance indeed improves with the quality of the dispatching rule. Therefore, it is recommended to use high quality dispatching rules to provide evaluation functions for the considered problem.

The best-performing versions of the beam search algorithms were then compared with the best existing heuristic (the LINET dispatching rule). The DBS and RBS algorithms clearly outperform the LINET heuristic. The beam search procedures perform comparatively better for the instances with high processing time and penalty variability. The improvement provided by the beam search heuristics increases significantly with the instance difficulty. Indeed, this improvement is quite minor for the easier instances. For the more difficult instances, however, the beam search procedures are greatly superior to the LINET dispatching rule.

The DBS heuristic provides the best performance, and is therefore recommended for small and medium size instances. For larger instances, however, this procedure requires excessive computation times. The RBS algorithm is computationally more efficient, and is the heuristic of choice for medium to large instances.

Acknowledgements

The author would like to thank the helpful comments of an anonymous referee.

References

Abdul-Razaq, T., & Potts, C. N. (1988). Dynamic programming state-space relaxation for single machine scheduling. *Journal of the Operational Research Society, 39*, 141–152.

Azizoglu, M., Kondakci, S., & Kirca, O. (1991). Bicriteria scheduling problem involving total tardiness and total earliness penalties. *International Journal of Production Economics, 23*, 17–24.

Baker, K. R., & Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: A review. *Operations Research, 38*, 22–36.

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

Della Croce, F., & 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, 53*, 1275–1280.

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

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

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

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

Korman, K. (1994). A pressing matter. *Video*, 46–50.

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

Li, G. (1997). Single machine earliness and tardiness scheduling. *European Journal of Operational Research, 96*, 546–558.

Liaw, C.-F. (1999). A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers & Operations Research, 26*, 679–693.

Lowerre, B. T. (1976). The HARPY Speech Recognition System. Ph.D. Thesis, Carnegie-Mellon University, USA.

Ow, P. S., & Morton, T. E. (1988). Filtered beam search in scheduling. *International Journal of Production Research, 26*, 35–62.

Ow, P. S., & Morton, T. E. (1989). The single machine early/tardy problem. *Management Science, 35*, 177–191.

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

Szwarc, W. (1993). Adjacent orderings in single-machine scheduling with earliness and tardiness penalties. *Naval Research Logistics, 40*, 229–243.

Valente, J. M. S. (2007). Dispatching heuristics for the single machine early/tardy scheduling problem with job-independent penalties. *Computers & Industrial Engineering, 52*, 434–447.

Valente, J. M. S., & Alves, R. A. F. S. (2005a). Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. *Computers & Industrial Engineering, 48*, 363–375.

Valente, J. M. S., & Alves, R. A. F. S. (2005b). Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research, 32*, 557–569.

Valente, J. M. S., & Alves, R. A. F. S. (2005c). Improved lower bounds for the early/tardy scheduling problem with no idle time. *Journal of the Operational Research Society, 56*, 604–612.

Valente, J. M. S., Gonçalves, J. F., & Alves, R. A. F. S. (2006). A hybrid genetic algorithm for the early/tardy scheduling problem. *Asia-Pacific Journal of Operational Research, 23*, 393–405.

Wagner, B. J., Davis, D. J., & Kher, H. (2002). The production of several items in a single facility with linearly changing demand rates. *Decision Sciences, 33*, 317–346.