Computers & Operations Research 39 (2012) 919–928

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Minimizing the weighted sum of squared tardiness on a single machine

Jeffrey Schaller [a,*], Jorge M.S. Valente [b]

[a] Department of Business Administration, Eastern Connecticut State University, 83 Windham St., Willimantic, CT 06226-2295, USA
[b] Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal

ARTICLE INFO

Available online 29 July 2011

Keywords:
Scheduling
Branch and Bound
Single machine
Weighted squared tardiness

ABSTRACT

This paper considers a problem in which there is a set of jobs to be sequenced on a single machine. Each job has a weight and the objective is to sequence the jobs to minimize total weighted squared tardiness. A branch-and-bound algorithm is developed for optimally solving the problem. Several dominance conditions are presented for possible inclusion in the branch-and-bound algorithm. The dominance conditions are included in the branch-and-bound algorithm, which is tested on randomly generated problems of various numbers of jobs, due date tightness and due date ranges. The results show that the dominance conditions dramatically improve the efficiency of the branch-and-bound algorithm.

1. Introduction and problem description

Over the last two decades companies have increasingly adopted the philosophy of just-in-time inventory control. The change to this approach has caused customers to view tardy delivery of products as undesirable. Tardy deliveries can result in lost sales and loss of customer goodwill. In this environment the cost of tardiness can increase rapidly as the tardiness of a job increases. A quadratic penalty can be used to represent a customer's dissatisfaction with the tardiness, as proposed in the loss function of Taguchi [32]. This paper considers the objective of minimizing the sum of weighted squared tardiness values for all jobs to be processed on a single machine. Suppose there is a set of n jobs available to be processed on a single machine. Let p_j, w_j and d_j represent the processing time, weight and due date of job $j (j=1, \ldots, n)$, respectively, and C_j the completion time of job j, $j=1, \ldots, n$. The tardiness of job j, T_j, is defined as: $T_j=\max\{0, C_j-d_j\}$, $j=1, \ldots, n$. The objective function, Z, can be expressed as: $\sum_{j=1}^{n} w_j T_j^2$. Since the objective is regular non-delay (or semi-active) schedules are dominant and there is no need to insert idle time [3]. Therefore with a given sequence of jobs, we can associate the total weighted squared tardiness of the schedule obtained by scheduling jobs as soon as possible in the order of the sequence.

Single machine scheduling environments actually occur in several practical operations; for a specific example in the chemical industry, see Ref. [40]. Also, the performance of many production systems is frequently determined by the quality of the schedules for a single bottleneck machine. Moreover, results and insights obtained for single machine problems can often be applied to more complex scheduling environments, such as flow shops or job shops.

Two streams of research that are related to the problem studied in this research are problems involving a quadratic measure of performance for scheduling a single machine and scheduling a single machine with minimizing total weighted tardiness as the objective. Relatively little work has been done on problems involving a quadratic measure of performance for scheduling a single machine. The single machine scheduling problem with the objective of minimizing the sum of squares of the job completion times has been studied by Schild and Fredman [24], Townsend [34], Bagga and Kalra [2], Gupta and Sen [12], and Szwarc et al. [31]. Schild and Fredman [24] developed precedence relationships when the objective is a weighted combination of quadratic and linear completion times. Townsend [34] developed a lower bound for minimizing the sum of quadratic completion times and a branch-and-bound algorithm for the problem. Bagga and Kalra [2] showed that the problem could be decomposed into two subproblems under certain conditions. Gupta and Sen [12] developed precedence relations, and Szwarc et al. [31] developed a precedence relation for ordering adjacent jobs for the problem. Gupta and Sen [11], Sen et al. [25], Su and Chang [28] and Schaller [22] developed procedures for minimizing the sum of squares job lateness on a single machine. Schaller [23], Valente [35–37], Valente and Gonçalves [38] and Valente and Schaller [39] developed procedures for the objective of minimizing the sum of early and squared tardiness values on a single machine.

Abdul-Razaq et al. [1] provide a survey and computational comparison of exact methods for the weighted tardiness problem. The exact methods include dynamic programming methods developed by Lawler [16], Schrage and Baker [21] and Srinivasan [27] and branch-and-bound algorithms developed by Gelders and Kleindorfer [9,10], Potts and van Wassenhove [18] and Rinnooy

* Corresponding author. Tel.: +1 860 465 5226; fax: +1 860 465 54469.
E-mail addresses: schallerj@ecsu.ctstateu.edu (J. Schaller), jvalente@fep.up.pt (J.M.S. Valente).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.07.018

Kan et al. [20]. Dominance tests for the problem have also been found to be very effective in reducing the search space for exact methods. Rinnooy Kan et al. [20] extend the dominance test developed by Emmons [6] for the unweighted tardiness problem to the weighted tardiness objective. Rachamadugu [19] developed a dominance test for adjacent jobs and Kanet [15] developed seven dominance tests for deciding precedence for pairs of jobs. Sen et al. [26] provide a survey for minimizing weighted and unweighted tardiness including non-exact methods for the weighted tardiness problem.

To the best of our knowledge, there have been relatively few papers that have considered the objective of minimizing the sum of weighted squared tardiness values. All of the approaches for the weighted squared tardiness objective use a Lagrangian relaxation in which the machine capacity constraints are relaxed to obtain a lower bound and then a heuristic is used to create a feasible solution and obtain an upper bound. The Lagrangian relaxation and its solution procedure is based on the one used by Refs. [7,8] for other objectives. Hoitomt et al. [13] and Luh and Hoitomt [17] developed procedures for scheduling jobs on parallel machines. Hoitomt et al. [13]'s procedure was for parallel machines in which jobs have multiple operations with precedence constraints. The procedure is demonstrated on three examples from a Pratt and Whitney plant. Luh and Hoitomt [17]'s procedure was for identical parallel machines and was also demonstrated using data from a Pratt and Whiney plant, including an example with 112 jobs and 44 machines. Sun et al. [30] consider the single machine problem with release dates and sequence dependent setup times. They compared their Lagrangian relaxation based heuristic against some simple dispatching rules, a tabu search and simulated annealing algorithms. These heuristics were tested using a variety of data sets most of which consisted of 40 jobs and ranged between 10 and 80 jobs. Luh and Hoitomt [17], Sun and Noble [29] and Thomalla [33] considered the job shop scheduling problem. Sun and Noble [29] considered the job shop scheduling problem with sequence dependent setups and Thomalla [33] considers the problem with alternative processing plans.

The objective of this research is to develop methods that will increase the efficiency of an optimal branch-and-bound algorithm for the problem. This optimal procedure can be used to solve small problems and approaches developed for the algorithm can provide insights that might be helpful in developing heuristic procedures that can solve larger problems. Also the optimal results obtained on problems solved with the branch-and-bound procedure can be used to evaluate the effectiveness of heuristic procedures.

Dominance conditions proved to be especially useful in reducing the size of problems when scheduling jobs on a single machine to minimize weighted total tardiness [15]. Ref. [14] informally defines a dominance rule as identifying a subset of solutions that contains at least one optimal solution for a problem. In the context of scheduling Kanet [15] defines a dominance condition as a rule that specifies that one job will precede another if certain conditions hold. In Section 2 we show how dominance conditions can be developed for the single machine scheduling problem when minimizing weighted squared tardiness is the objective. In Section 3 a branch-and-bound algorithm is presented. Section 4 describes the computational tests and presents the results. Section 5 concludes the paper.

2. Dominance conditions

In this section dominance conditions for eliminating nodes in the branch-and-bound algorithm are presented. Several conditions and/or the respective proofs utilize the rate of change in the objective as a job's completion time changes. Note that the objective function can be rewritten as $Z=\sum_{j=1}^{n} Z_j$, where

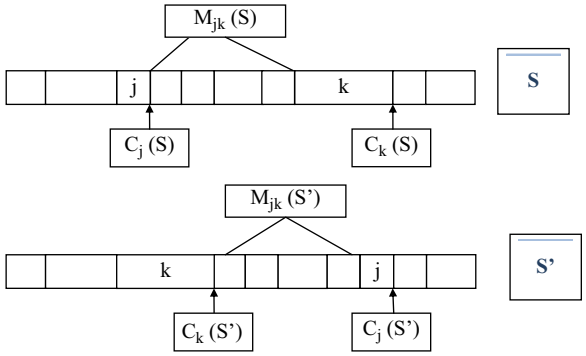

Fig. 1. Swapping positions of jobs j and k.

$Z_j=w_jT_j^2$. Suppose $C_j=t$ and let $D_j(t)$ equal the increase in Z_j if C_j is increased from t to $t+1$. Then $D_j(t)=0$ if $t<d_j$, and $D_j(t)=w_j(2T_j+1)$ if $t\geq d_j$. Let $D_j^2(t)$ be the increase of $D_j(t)$ if C_j is increased from t to $t+1$. $D_j^2(t)=0$ if $t<d_j$, and $D_j^2(t)=2w_j$ if $t\geq d_j$.

The dominance conditions stated in Theorem 1, Propositions 1–4 and Corollaries 1–6 identify conditions in which it can be shown that a job with a smaller processing time will precede a job with a larger processing time. These conditions are proved by swapping the positions of a pair of jobs j and k in a sequence as illustrated in Fig. 1. The following definitions are used in these proofs. Let S be a sequence with job j sequenced before job k. Let $C_j(S)$ and $C_k(S)$ be the completion time of job j and job k, respectively, in the schedule associated with sequence S. Also, let $Z_j(S)$ and $Z_k(S)$ be the weighted squared tardiness of job j and job k, respectively, in this schedule. Let S' be a sequence that is the same as S except the positions of jobs j and k are exchanged in S'. Let $C_j(S')$ and $C_k(S')$ be the completion time of job j and job k, respectively, in the schedule associated with sequence S'. Similarly, let $Z_j(S')$ and $Z_k(S')$ be the weighted squared tardiness of job j and job k, respectively, in this schedule. Let $M_{jk}(S)$ be the set of jobs that are between jobs j and k in sequences S and $S'(M_{jk}(S)=M_{kj}(S'))$.

Please note that since $p_j\leq p_k$ then the completion times of the jobs in set $M_{jk}(S)$ will occur before or at the same time in the schedule associated with sequence S as the completion times in the schedule corresponding to sequence S' as shown in Fig. 1. Also define $B(k)$ as the set of jobs known to be sequenced before job k in at least one optimal sequence and $A(j)$ as the set of jobs known to be sequenced after job j in at least one optimal sequence. Finally, let $t_{B(k)}=\sum_{l\in B(k)}p_l$ and $t_{A(j)}=\sum_{l=1}^{n}p_l-\sum_{l\in A(j)}p_l$. $t_{B(k)}$ is a lower bound on the start time of job k and $t_{A(j)}$ is an upper bound on the completion time for job j.

As shown in Fig. 1, $C_k(S)=C_j(S')$. Also, the completion times of the jobs sequenced before job j and after job k in sequence S will not change in sequence S'. The completion times of the jobs in set $M_{jk}(S)$ will be no later under sequence S' than under sequence S. Since the increase in weighted squared tardiness of job j if it is completed at time $C_j(S')$ instead of $C_j(S)$ is at least as great as the increase in weighted squared tardiness of job j if it is completed at time $C_j(S')$ instead of $C_k(S')$ it only needs to be shown that the increase in job j's weighted squared tardiness when its completion time is increased from $C_k(S')$ to $C_j(S')$ is greater than the increase in job k's weighted squared tardiness when its completion time is increased from $C_k(S')$ to $C_k(S)$ when $p_j\leq p_k$ to prove that job j precedes job k in at least one optimal schedule. Theorem 1, Propositions 1–4 and Corollaries 1–6 are proved using this concept and the rate of change function described above.

Theorem 1. *If $p_j\leq p_k$ and $D_j(t)\geq D_k(t)$ for $C_k(S')\leq t\leq C_k(S)$ then sequence S will result in a total weighted squared tardiness that is less than or equal to that of sequence S'.*

Proof. As previously noted, the jobs in set $M_{jk}(S)$ will have a completion time in the schedule associated with sequence S that is less than or equal to their completion time under sequence S'. Also, the completion times of the jobs sequenced before job j in S(or k in S') and the jobs sequenced after job k in S(or job j in S') will not be changed. Therefore, to prove this theorem only the weighted squared tardiness of jobs j and k need to be considered, and it must be shown that, when sequence S is used instead of sequence S', the increase in job j's weighted squared tardiness $(Z_j(S')-Z_j(S))$ is greater than or equal to the decrease in job k's weighted squared tardiness $(Z_k(S)-Z_k(S'))$. Also note that $C_k(S)=C_j(S')$ and $C_k(S')=C_j(S)+p_k-p_j\geq C_j(S)$. Since $C_k(S')\geq C_j(S)$, we then have $Z_j(S')-Z_j(S)\geq Z_j(S')-w_j\max\{C_k(S')-d_j,0\}^2$. Since $D_j(t)\geq D_k(t)$ for $C_k(S')\leq t\leq C_k(S)$, then $Z_j(S')-w_j\max\{C_k(S')-d_j,0\}^2\geq Z_k(S)-Z_k(S')$, which implies $Z_j(S')-Z_j(S)\geq Z_k(S)-Z_k(S')$, and the theorem is proved. □

Propositions 1–4 and Corollaries 1–6 provide specific conditions that implement Theorem 1. Proposition 1 is a version of the Emmons rule [6] which the proof for proposition 1 shows is valid for this problem.

Proposition 1. *If $p_j\leq p_k$, $d_j\leq d_k$ and $w_j\geq w_k$ then job j is sequenced before job k in at least one optimal sequence.*

Proof. Since $d_j\leq d_k$ and $w_j\geq w_k$ then $D_j(t)\geq D_k(t)$ when $t=C_k(S')$. Since $w_j\geq w_k$ then $D_j(t)\geq D_k(t)$ for $t\geq C_k(S')$. Therefore, $D_j(t)\geq D_k(t)$ for $C_k(S')\leq t\leq C_k(S)$, and the conditions of Theorem 1 are met for any S and S'. □

Proposition 2 and Corollaries 1–4 provide conditions for job j preceding job k when $p_j\leq p_k$, $d_k<d_j$, and $w_j>w_k$.

Proposition 2. *If $p_j\leq p_k$, $d_k<d_j$, $w_j>w_k$, $C_k(S')>d_j$ and $w_j(2\max\{C_k(S')-d_j,0\}+1)\geq w_k(2\max\{C_k(S')-d_k,0\}+1)$ then sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S'.*

Proof. Using the rate of change functions defined earlier it is shown in this proof that if the rate of increase in job j's weighted squared tardiness is greater than or equal to the rate of increase in job k's at time $t=C_k(S')$ (job k's completion time in schedule S'), then the weighted squared tardiness of sequence S will be less than or equal to that of sequence S'. We have $w_j(2\max\{C_k(S')-d_j,0\}+1)\geq w_k(2\max\{C_k(S')-d_k,0\}+1)$, which implies $D_j(t)\geq D_k(t)$ for $t=C_k(S')$. Also, since $w_j>w_k$ we have $D_j^2(t)\geq D_k^2(t)$ for all $t\geq C_k(S')$. Therefore, $D_j(t)\geq D_k(t)$ for $C_k(S')\leq t\leq C_k(S)$ and the conditions of Theorem 1 hold. □

Corollaries 1 and 2 use set $B(k)$ and the lower bound on the completion time of job $k(t_{B(k)}+p_k)$ to develop conditions such that job j can be sequenced before job k in at least one optimal sequence, and hence can be added to set $B(k)$. The proofs for these two corollaries assume that job k is necessarily sequenced after the jobs in set $B(k)$, in both sequences S and S'.

Corollary 1. *If $j\notin B(k)$, $p_j\leq p_k$, $d_k<d_j$, $w_j>w_k$, $t_{B(k)}+p_k>d_j$ and $w_j(2\max\{t_{B(k)}+p_k-d_j,0\}+1)\geq w_k(2\max\{t_{B(k)}+p_k-d_k,0\}+1)$ then job j can be added to B(k).*

Proof. We have $C_k(S')\geq t_{B(k)}+p_k$. Also the conditions of the corollary together imply that $w_j(2\max\{C_k(S')-d_j,0\}+1)\geq w_k(2\max\{C_k(S')-d_k,0\}+1)$. Therefore, the conditions of Proposition 2 hold, so sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S' and job j can be added to set $B(k)$. □

Corollary 2. *If $j\notin B(k)$, $p_j\leq p_k$, $d_k<d_j$, $w_j>w_k$, $t_{B(k)}+p_k>d_j$, $(w_j\max\{t_{B(k)}+p_k+p_j-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k-d_k,0\}^2)\geq(w_j\max\{t_{B(k)}+p_j-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k+p_j-d_k,0\}^2)$ and $w_j(2\max\{t_{B(k)}+p_k+p_j-d_j,0\}+1)\geq w_k(2\max\{t_{B(k)}+p_k+p_j-d_k,0\}+1)$ then job j can be added to set $B_{(k)}$.*

Proof. We have $C_k(S')\geq t_{B(k)}+p_k$. If $D_j(t)>D_k(t)$ for $t=t_{B(k)}+p_k$, then $D_j(t)>D_k(t)$ for $C_k(S')\leq t\leq C_k(S)$ and the conditions of Theorem 1 hold, so sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S'. Now consider the case $D_j(t)\leq D_k(t)$ for $t=t_{B(k)}+p_k$. In this case, $w_j>w_k$ and $w_j(2\max\{t_{B(k)}+p_k+p_j-d_j,0\}+1)\geq w_k(2\max\{t_{B(k)}+p_k+p_j-d_k,0\}+1)$, so $D_j(t)>D_k(t)$ for $t_{B(k)}+p_k+p_j\leq t$. Consequently, $(Z_j(S')+Z_k(S')-(Z_j(S)+Z_k(S)))\geq(w_j\max\{t_{B(k)}+p_k+p_j-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k-d_k,0\}^2)-(w_j\max\{t_{B(k)}+p_j-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k+p_j-d_k,0\}^2)\geq 0$. Therefore, sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S' in this case, and job j can be added to set $B(k)$. □

Consider a pair of jobs j and k with $w_j>w_k$ and $d_k<d_j$. For such a pair of jobs, let $\delta\{jk\}$ denote the moment in time such that, if both jobs are both completed at or after time $t=\delta\{jk\}$, then $D_j(t)\geq D_k(t)$. It can be shown that $\delta\{jk\}=\max\{d_j+(2w_k(d_j-d_k)+w_k-w_j)/(2(w_j-w_k)),\ d_j\}$. The time $\delta\{jk\}$ is used to develop two additional corollaries to Proposition 2 (Corollaries 3 and 4).

Corollary 3. *If $p_j\leq p_k$, $d_k<d_j$, $w_j>w_k$ and $C_k(S')>\delta\{jk\}$ then sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S'.*

Proof. Note that, since $p_j\leq p_k$ the jobs in set $M_{jk}(S)$ will have a completion time in the schedule associated with sequence S that is less than or equal to their completion time under sequence S'. The completion times of the jobs sequenced before job j in S (or k in S') and the jobs sequenced after job k in S (or job j in S') will not be changed. Therefore, to prove this theorem it must be shown that $Z_j(S)+Z_k(S)\leq Z_j(S')+Z_k(S')$.

If $t=d_j$, then $D_k(t)=2w_k(d_j-d_k)+w_k$ and $D_j(t)=w_j$. Also, $D_k^2(t)=2w_k$ and $D_j^2(t)=2w_j$. If $w_j\geq 2w_k(d_j-d_k)+w_k$, then $D_j(t)\geq D_k(t)$ for $t=d_j$, and $w_j>w_k$ implies $D_j(t)\geq D_k(t)$ for $t\geq d_j$. We also have $C_k(S')>\delta\{jk\}\geq d_j$. Therefore, all the conditions of Proposition 2 hold, and $Z_j(S)+Z_k(S)\leq Z_j(S')+Z_k(S')$.

If $w_j<2w_k(d_j-d_k)+w_k$ then $D_j(t)=D_k(t)$ when $t=d_j+(2w_k(d_j-d_k)+w_k-w_j)/(2(w_j-w_k))$, and $w_j>w_k$ implies $D_j(t)\geq D_k(t)$ for $t\geq d_j+(2w_k(d_j-d_k)+w_k-w_j)/(2(w_j-w_k))$. Since $C_k(S')>\delta\{jk\}\geq d_j+(2w_k(d_j-d_k)+w_k-w_j)/(2(w_j-w_k))$, the conditions of Proposition 2 hold, and $Z_j(S)+Z_k(S)\leq Z_j(S')+Z_k(S')$. □

Corollary 4 compares the weighted squared tardiness of jobs j and k if the jobs in set $B(k)$ are sequenced at the beginning, and either job j or job k immediately follows the jobs in set $B(k)$, and the other is completed at time $\delta\{jk\}$.

Corollary 4. *If $j\notin B(k)$, $p_j\leq p_k$, $d_k<d_j$, $w_j>w_k$ and $(w_j\max\{\delta\{jk\}-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k-d_k,0\}^2)>(w_j\max\{t_{B(k)}+p_j-d_j,0\}^2+w_k\max\{\delta\{jk\}-d_k,0\}^2)$ then job j can be added to set B(k).*

Proof. This proof assumes that job k is necessarily sequenced after the jobs in set $B(k)$, in both sequences S and S'. Note that since $p_j\leq p_k$, the jobs in set $M_{jk}(S)$ will have a completion time in the schedule associated with sequence S that is before or equal to their completion time under sequence S'. The completion times of the jobs sequenced before job j in S (or k in S'), and the jobs sequenced after job k in S (or job j in S') will not be changed. Therefore, to prove this corollary it must be shown that $Z_j(S)+Z_k(S)\leq Z_j(S')+Z_k(S')$ or equivalently, that $Z_j(S')+Z_k(S')-(Z_j(S)+Z_k(S))\geq 0$. We have $C_k(S')\geq t_{B(k)}+p_k$. If $C_k(S')\geq\delta\{jk\}$ then by Corollary 3 sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S'.

We have $D_j(t)\geq D_k(t)$ for $t\geq\delta\{jk\}$ and $D_j(t)\leq D_k(t)$ for $t<\delta\{jk\}$. Therefore, for $t_{B(k)}+p_k\leq C_k(S')\leq\delta\{jk\}$, we have $(Z_k(S')-w_k\max$

$\{t_{B(k)}+p_k-d_k,0\}^2) \geq (Z_j(S)-w_j\max \{t_{B(k)}+p_j-d_j,0\}^2)$. Also, if $C_k(S)<\delta\{jk\}$, then $(w_k\max \{\delta\{jk\}-d_k,0\}^2-Z_k(S)) \geq (w_j\max \{\delta\{jk\}-d_j,0\}^2-Z_j\ (S'))$. Furthermore, if $C_k(S)>\delta\{jk\}$, then $(Z_k(S)-w_k\max\{\delta\{jk\}-d_k,0\}^2) \geq (Z_j(S')-w_j\max\{\delta\{jk\}-d_j,0\}^2)$. Therefore, and combining the previous results, for $t_{B(k)}+p_k \leq C_k(S') \leq \delta\{jk\}$ we have $(Z_j(S')+Z_k(S')-(Z_j(S)+Z_k(S))) \geq (w_j\max \{\delta\{jk\}-d_jd_j,0\}^2+w_k\max\{t_{B(k)}+p_k-d_k,0\}^2-(w_j\max\{t_{B(k)}+p_j-d_j,0\}^2+w_k\max\{\delta\{jk\}-d_k,0\}^2)) > 0$. Consequently, sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S', and job j can be added to set $B(k)$. □

Proposition 3 and Corollaries 5 and 6 provide conditions for job j preceding job k when $p_j \leq p_k$ and $d_j < d_k$, but $w_j < w_k$.

Proposition 3. *If $p_j \leq p_k$, $d_j < d_k$, $w_j < w_k$ and $w_j(2\max\{C_k(S)-d_j, 0\}+1) \geq w_k(2\max\{C_k(S)-d_k, 0\}+1)$ then sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S′.*

Proof. This proof shows that if the rate of increase in job j's weighted squared tardiness is greater than or equal to the rate of increase in job k's at time $t=C_k(S)$ (job k's completion time in sequence S), then the weighted squared tardiness of sequence S will be less than or equal to that of sequence S'. We have $w_j(2\max\{C_k(S)-d_j,0\}+1) \geq w_k(2\max\{C_k(S)-d_k,0\}+1)$, which implies that $D_j(t) \geq D_k(t)$ for $t=C_k(S)$. Also, if $t<d_k$ we have $D_k(t)=0$. Therefore, $D_j(t) \geq D_k\ (t)$ (since $D_j(t) \geq 0$ for all t) for $C_k(S') \leq t < d_k$. For $d_k \leq t < C_k(S)$, $D_k(t)=w_k(2(C_k(S)-d_k)+1)-2w_k(C_k(S)-\mathrm{t})$ and $D_j(t)=w_j(2(C_k(S)-d_j)+1)-2w_j(C_k(S)-\mathrm{t})$. Since $w_k > w_j$, we have $2w_k(C_k(S)-\mathrm{t}) > 2w_j(C_k(S)-\mathrm{t})$ for $d_k \leq t < C_k(S)$. Combining this result with the condition, $w_j(2\max\{C_k(S)-d_j,0\}+1) \geq w_k(2\max\{C_k(S)-d_k,0\}+1)$, we obtain that $w_k(2(C_k(S)-d_k)+1) \geq w_j(2(C_k(S)-d_j)+1)$ when $d_k \leq t < C_k(S)$. Therefore, we have $D_j(t) \geq D_k(t)$ for $C_k(S') \leq t \leq C_k(S)$, and the conditions of Theorem 1 then hold. □

Corollaries 5 and 6 use set $A(j)$ and the upper bound on the completion time of job j ($t_{A(j)}$) to develop conditions such that job j can be sequenced before job k in at least one optimal sequence, and hence job k can be added to set $A(j)$. The proofs for these two corollaries assume that the jobs in set $A(j)$ are necessarily sequenced after job j, in both sequences S and S'.

Corollary 5. *If job $k \notin A(j)$, $p_j \leq p_k$, $d_j < d_k$, $w_j < w_k$ and $w_j(2\max\{t_{A(j)}-d_j, 0\}+1) \geq w_k(2\max\{t_{A(j)}-d_k, 0\}+1)$ then job k can be added to A(j).*

Proof. Consider the following two cases: 1) $d_k > C_k(S)$; 2) $d_k \leq C_k(S)$. If $d_k > C_k(S)$, then $D_k(t)=0$. Therefore, $D_j(t) \geq D_k(t)$ (since $D_j(t) \geq 0$ for all t) for $C_k(S') \leq \mathrm{t} \leq C_k(S)$. By Theorem 1, sequence S will then have a total weighted squared tardiness that is less than or equal to that of sequence S' in case 1). If $d_k \leq C_k\ (S)$, then $w_j(2\max\{t_{A(j)}-d_j,0\}+1)=w_j2(t_{A(j)}-d_jd_j+1)$ and $w_k(2\max\{t_{A(j)}-d_k,0\}+1)=w_k2(t_{A(j)}-d_k+1)$. Also, we have $w_j(2\max\{C_k(S)-d_j,0\}+1)=w_j2(C_k(S)-d_j+1)=w_j2(t_{A(j)}-d_j+1)-w_j2(t_{A(j)}-C_k(S))$. Furthermore, $w_k(2\max\{C_k(S)-d_k,0\}+1)=w_k2(C_k(S)-d_k+1)=w_k2(t_{A(j)}-d_k+1)-w_k2(t_{A(j)}-C_k(S))$. Since $w_k > w_j$, we have $w_k2(t_{A(j)}-C_k(S)) > w_j2(t_{A(j)}-C_k(S))$. Also, $w_j(2\max\{t_{A\ (j)}-d_j,0\}+1) \geq w_k(2\max\{t_{A(j)}-d_k,0\}+1)$ implies that $w_j2(t_{A(j)}-d_j+1) \geq w_k2(t_{A(j)}-d_k+1)$. Therefore, $w_j(2\max\{C_k(S)-d_j,0\}+1) \geq w_k(2\max\{C_k(S)-d_k,0\}+1)$, so the conditions of Proposition 3 hold and sequence S dominates sequence S' in case 2). Since sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S' in both cases, job k can be added to set $A(j)$. □

Corollary 6. *If job $k \notin A(j)$, $p_j \leq p_k$, $d_j < d_k$, $w_j < w_k$, $(w_j\max\{t_{A(j)}-d_j,0\}^2+w_k\max\{t_{A(j)}-p_j-d_k,0\}^2) \geq (w_j\max\{t_{A\ (j)}-p_k-d_j,0\}^2+w_k\max\{t_{A(j)}-d_k,0\}^2)$ and $w_j(2\max\{t_{A(j)}-p_j-d_j,0\}+1) \geq w_k(2\max\{t_{A(j)}-p_j-d_k,0\}+1)$ then job k can be added to set A(j).*

Proof. Naturally, we have $C_k(S) \leq t_{A\ (j)}$. If $D_j(t) > D_k(t)$ for $t=t_{A(j)}$, then $D_j(t) > D_k(t)$ for $C_k(S') \leq t \leq C_k(S)$ and the conditions of Theorem 1 hold, so sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S'. Now consider the case $D_j\ (t) \leq D_k(t)$ for $t=t_{A(j)}$. In this case, $w_k > w_j$ and $w_j(2\max\{t_{A(j)}-p_j-d_j,0\}+1) \geq w_k(2\max\{t_{A(j)}-p_j-d_k,0\}+1$ together imply that $D_j\ (t) > D_k(t)$ for $t \leq t_{A(j)}-p_j$. Therefore, $(Z_j(S')+Z_k(S'))-(Z_j(S)+Z_k(S)) \geq (w_j\max\{t_{A(j)}-d_j,0\}^2+w_k\max\{t_{A(j)}-p_k-d_k,0\}^2)-(w_j\max\{t_{A(j)}-p_k-d_j,0\}^2+w_k\max\{t_{A(j)}-d_k,0\}^2)) \geq 0$. Therefore, sequence S will have a total weighted squared tardiness that is less than or equal to that of sequence S' in this case. Consequently, job k can be added to set $A(j)$. □

Propositions 4–6, and Corollary 7 develop conditions, which determine whether or not a job can be sequenced last among jobs forming an optimal initial partial sequence. The following notation is used in Propositions 4–6 and Corollary 7. Let U be a set of unsequenced jobs that will be sequenced at the beginning of a complete sequence. Also, let $t_U=\sum_{l \in U} p1$(i.e. t_U is the completion time of the last job in set U). In the following proofs, the definitions of sequences S and S' are somewhat different from those used so far, and will be provided in each proof.

Proposition 4 provides conditions in which job j cannot be sequenced last among the jobs in set U in an optimal sequence if there is a job k that belongs to set U such that $p_j \leq p_k$ and $w_j > w_k$. The fifth condition of the proposition compares the weighted squared tardiness of jobs j and k if the jobs in set $B(k)$ are sequenced at the beginning of a sequence and either job j or job k immediately follows the jobs in set $B(k)$ and the other job is sequenced last among the jobs in set U.

Proposition 4. *If jobs j and $k \epsilon U$, $p_j < p_k$, $w_j > w_k$, job $j \notin B(k)$ and $(w_j\max\{t_U-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k-d_k,0\}^2) > (w_j\max\{t_{B(k)}+p_j-d_j,0\}^2+w_k\max\{t_U-d_k,0\}^2)$, then job j cannot be sequenced last among the jobs in set U in all optimal sequences beginning with jobs belonging to set U.*

Proof. The proof for this proposition uses set $B(k)$ and the lower bound on the completion time of job $k(t_{B(k)}+p_k)$, and shows that if the weighted squared tardiness for jobs j and k when job j immediately follows set $B(k)$ with job k sequenced last among the jobs in set U is less than the weighted squared tardiness for jobs j and k if their positions are reversed and the other conditions hold, then job j cannot be sequenced last among the jobs in set U. Let S' be a sequence with job j sequenced last among the jobs in U and job k sequenced before job j and after the set of jobs in $B(k)$. Let S be a sequence that is the same as S' except the positions of jobs j and k are exchanged, so job k is sequenced last among the jobs in set U. Note that $C_k(S') \geq t_{B\ (k)}+p_k$. There are two possible cases: (1) $d_j \leq d_k$ and (2) $d_k < d_j$. In case (1), the conditions of Proposition 1 hold, so sequence S will have a total weighted squared tardiness that is less than that of S', and job j cannot be sequenced last among the jobs in set U.

We now consider case (2) ($d_k < d_j$). In this case, either $D_j(t) \geq D_k(t)$ for $t=t_{B(k)}+p_k$, or $D_k(t) > D_j(t)$ for $t=t_{B(k)}+p_k$. If $D_j(t) \geq D_k(t)$ for $t=t_{B(k)}+p_k$ then $w_j > w_k$ implies $D_j(t) \geq D_k(t)$ for $t \geq t_{B(k)}+p_k$, so the conditions of Theorem 1 hold. Therefore, sequence S will have a total weighted squared tardiness that is less than that of S', and job j cannot be sequenced last among the jobs in set U. If $D_k(t) > D_j(t)$ for $t=t_{B(k)}+p_k$ then $(Z_j(S')+Z_k(S'))-(Z_j(S)+Z_k(S)) > ((w_j\max\{t_U-d_j,0\}^2+w_k\max\{t_{B(k)}+p_k-d_k,0\}^2)-(w_j\max\{t_{B(k)}+p_j-d_j,0\}^2+w_k\max\{t_U-d_k,0\}^2)) > 0$. Thus, sequence

S will have a total weighted squared tardiness that is less than that of S', and job j cannot be sequenced last among the jobs in set U. □

Propositions 5 and 6 and Corollary 7 are proved using an insert-after strategy. In the insert-after strategy, conditions are found in which inserting a job k immediately after a job j in a sequence never results in a higher weighted squared tardiness.

The next proposition was developed by Elmagraby [5] for the total tardiness objective on a single machine, and states that if a post-partial sequence is sequenced to begin before an unsequenced job's due date, then that job can be appended at the beginning of the post-partial sequence. The proof for the proposition shows that it can be applied to the total weighted squared tardiness objective on a single machine.

Proposition 5. *If there is a job $j \epsilon U$ such that $d_j \geq t_U$ then job j can be sequenced last among the jobs in set U in at least one optimal sequence beginning with the jobs in set U if such an optimal sequence exists.*

Proof. Let S' be a sequence with job j not sequenced last among the jobs in set U. Let S be a sequence that is the same as S' except job j is removed from its position in S' and inserted at the end of the sequence of jobs in set U. These sequences as well as the relation between d_j and T_U are shown in Fig. 2.

Since the jobs that were sequenced after job j in sequence S' will be completed earlier in the schedule associated with sequence S, their weighted squared tardiness in sequence S will be less than or equal to that in sequence S'. Since $d_j \geq t_U$, $Z_j(S)=0$ and the total weighted squared tardiness of the jobs in sequence S is no greater than that of S'. Therefore, job j can be sequenced last among the jobs in set U in an optimal sequence. □

Proposition 6 identifies unsequenced jobs that cannot be sequenced last among the set of unsequenced jobs U (i.e., appended to the beginning of an optimal post-partial sequence). The proposition uses set $B(k)$ defined earlier to obtain a lower bound on the weighted squared tardiness of job k. The weighted squared tardiness of jobs j and k are found if the jobs are sequenced last among the jobs in set U (so the jobs are completed at time t_U). Job j's weighted squared tardiness is also found if job k is sequenced last in set U (job k is completed at time t_U) and job j is sequenced immediately before job k (job j is completed at time $t_U - p_k$).

We remark that the weighted squared tardiness of job k when it is completed at time t_U minus job k's weighted squared tardiness when it is completed at time $t_{B(k)}+p_k$ is an upper bound on the increase in job k's weighted squared tardiness if it is sequenced last among the set of jobs in U instead of earlier in the sequence. If the reduction in job j's weighted squared tardiness

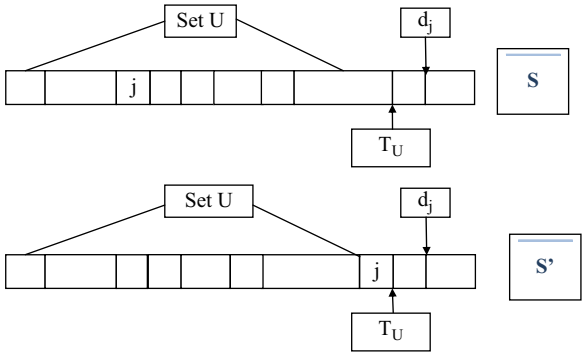

Fig. 2. Insert-after strategy for proof of Proposition 5.

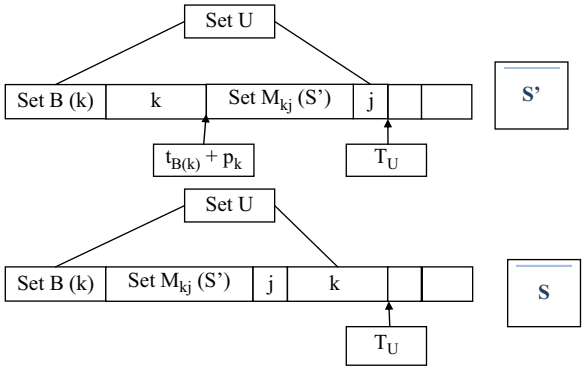

Fig. 3. Insert-after strategy for proofs of Proposition 6 and Corollary 7.

when job j is sequenced next to last among the jobs in set U immediately before job k, instead of last among the jobs in set U, is greater than this upper bound, then job j cannot be sequenced last among the jobs in set U. These concepts are used in the insert-after strategy illustrated in Fig. 3 to prove Proposition 6.

Proposition 6. *Let j and k be jobs in set U where $j \notin B(k)$. If $(w_j \max\{t_U - d_j, 0\}^2 - w_j \max\{t_U - p_k - d_j, 0\}^2) > (w_k \max\{t_U - d_k, 0\}^2) - w_k \max\{t_{B(k)} + p_k - d_k, 0\}^2)$, then job j cannot be sequenced last among the jobs in set U in at least one optimal sequence beginning with the jobs in set U.*

Proof. Let S' be a sequence with job j sequenced last among the jobs in U. Let S be a sequence that is the same as S' except that job k is removed from its position in S' and inserted after job j so it is sequenced last among the jobs in U (see Fig. 3). Since the completion times of the jobs that are sequenced before job k in sequence S' are the same as those in sequence S, and the completion times of the jobs in set $M_{kj}(S')$ are earlier in the schedule associated with sequence S than under sequence S', to prove this theorem it only needs to be shown that $Z_j(S) + Z_k(S) \leq Z_j(S') + Z_k(S')$. Since $t_{B(k)}+p_k$ is a lower bound on the completion time of job k, $C_k(S') \geq t_{B(k)}+p_k$, and therefore $(w_k \max\{t_U - d_k, 0\}^2) - w_k \max\{t_{B(k)} + p_k - d_k, 0\}^2) \geq Z_k(S) - Z_k(S')$. Also, $Z_j(S') - Z_j(S) = (w_j \max\{t_U - d_j, 0\}^2 - w_j \max\{t_U - p_k - d_j, 0\}^2)$. Since the propositions conditions state that $(w_j \max\{t_U - d_j, 0\}^2 - w_j \max\{t_U - p_k - d_j, 0\}^2) > (w_k \max\{t_U - d_k, 0\}^2) - w_k \max\{t_{B(k)} + p_k - d_k, 0\}^2)$, we have $Z_j(S') - Z_j(S) > Z_k(S) - Z_k(S')$ and $Z_j(S) + Z_k(S) \leq Z_j(S') + Z_k(S')$. Therefore, sequence S dominates sequence S' and job j cannot be sequenced last among the jobs in set U in at least one optimal sequence beginning with the jobs in set U. □

The following notation is used to develop the condition in Corollary 7. Let $Z_j(t_U) = w_j \max\{t_U - d_j, 0\}^2$, $Z_j(t_U - p_k) = w_j \max\{t_U - p_k - d_j, 0\}^2$, $\Delta Z_j = Z_j(t_U) - Z_j(t_U - p_k)$ and $Z_k(t_U) = w_k \max\{t_U - d_k, 0\}^2$. For jobs j and k belonging to set U, Corollary 7 sets an upper bound on job k's completion time if job j is sequenced last among the jobs in set U. Job k's completion time upper bound is based on the maximum weighted squared tardiness job k can have, which corresponds to removing job k from its position and inserting it immediately after job j so job k is sequenced last among the jobs in set U.

Corollary 7. *If jobs j and $k \epsilon U$, and job j is sequenced last among the jobs in set U with a completion time of t_U, then in an optimal sequence $C_k \leq \sqrt{\max\{Z_k(t_U) - \Delta Z_j, 0\}/w_k} + d_k$.*

Proof. Let S' be a sequence with job j sequenced last among the jobs in U. Let S be a sequence that is the same as S' except that job k is removed from its position in S' and inserted after job j so it is sequenced last among the jobs in U. Since the completion times of

the jobs that are sequenced before job k in sequence S' are the same as those in sequence S, and the completion times of the jobs in set $M_{kj}(S')$ are earlier in the schedule associated with sequence S than under sequence S', to prove this theorem it only needs to be shown that if $C_k=C_k\ (S')>\sqrt{\max\{Z_k(t_U)-\Delta Z_j,0\}/w_k}+d_k$ then $Z_j(S)+Z_k(S)<Z_j(S')+Z_k(S')$ and sequence S would result in a lower weighted squared tardiness than sequence S'. Therefore job j could not be sequenced last among the jobs in set U in an optimal sequence. $Z_j\ (S)+Z_k(S)<Z_j(S')+Z_k(S')$ implies $Z_k(S)-Z_k(S')<Z_j(S')-Z_j(S)$. Also, $Z_j(S')-Z_j(S)=\Delta Z_j$ and $Z_k(t_U)=Z_k(S)$ so if $Z_k(S')>\max\{Z_k\,(t_U)-\Delta Z_j,0\}$, then $Z_k(S)-Z_k(S')<Z_j(S')-Z_j(S)$. Since $Z_k(S')=w_k\max\{C_k(S')-d_k,0\}^2$, $C_k(S')>\sqrt{\max\{Z_k(t_U)-\Delta Z_j,0\}/w_k}+d_k$ implies $Z_k(S)-Z_k(S')<Z_j(S')-Z_j(S)$, which concludes the proof. □

The next proposition identifies which among two adjacently sequenced jobs will be first. Let j and k be a pair of jobs that are to be sequenced adjacent to each other. Let $B(jk)$ be the set of jobs that are sequenced before the jobs j and k in an optimal sequence. Also, let $t_{B(jk)}=\sum_{l\in B(jk)}p_l$.

Proposition 7. *If jobs j and k are to be sequenced adjacent to each other after the set of jobs in B(jk), and* $(w_j\max\{t_{B(jk)}+p_j-d_j,0\}^2+w_k\max\{t_{B(jk)}+p_k+p_j-d_k,0\}^2)<(w_j\max\{t_{B(jk)}+p_k+p_j-d_j,0\}^2+w_k\max\{t_{B(jk)}+p_k-d_k,0\}^2)$, *then job j precedes job k.*

Proof. Let S be a sequence in which jobs j and k are sequenced adjacent to each other with job j sequenced before job k. Let S' be a sequence that is the same as S except the positions of jobs j and k are exchanged in S'. The sum of the weighted squared tardiness of the jobs before and after jobs j and k is the same under either sequence S or S'. The conditions of the proposition establish that $Z_j(S)+Z_k(S)<Z_j(S')+Z_k(S')$. Therefore, the total weighted tardiness of sequence S is less than that of sequence S', and job j precedes job k. □

Proposition 8. *Let σ and σ′ be post-partial subsequences consisting of the same set of jobs. Let Z(σ), and Z(σ′) be the total weighted squared tardiness of the jobs in the associated partial subsequence. If* $Z\,(\sigma)<Z(\sigma')$ *then σ dominates σ′ and the partial sequence σ′ can be eliminated from consideration.*

Proof. Let U be a set of unsequenced jobs that are not included in post-partial subsequences σ and σ' and will be sequenced at the beginning of a complete sequence. Also, let $t_U=\sum_{l\in U}p_l$, which is the completion time of the jobs in set U and the start time of the first job in σ and σ'. Let π be any initial partial subsequence consisting of the jobs in set U. $Z(\sigma)+Z(\pi)<Z(\sigma')+Z(\pi)$ for any π if $Z(\sigma)<Z(\sigma')$ therefore σ will result in a lower total weighted tardiness if it is used instead of σ' as a post-partial subsequence. □

3. Branch-and-bound algorithm

This section describes a branch-and-bound procedure that finds a sequence for a set of jobs that minimizes the sum of weighted squared tardiness. Branch-and-bound has been defined by Ref. [4] as: "an algorithmic technique to find the optimal solution by keeping the best solution found so far. If a partial solution cannot improve on the best, it is abandoned." In the branch-and-bound procedure a sequence is constructed starting at the end and working forward so a node at the pth level in the branch-and-bound tree corresponds to a partial sequence (and the associated sub-problem) for the last p jobs in a sequence. Branching adds a job to the last unassigned position (at the beginning, since the scheduling is done backwards) in a partial sequence. From a node at level-p, up to q ($q=n-p$) branches may be generated, one for each job not in the partial sequence corresponding to the level-p node from which branching occurs. Before branching takes place, the dominance conditions described in Section 2 are checked to see if any of the candidates for the qth position in a sequence can be eliminated from consideration. Therefore, less than q branches may be generated.

When branching occurs and new nodes are created, a lower bound on the sum of weighted squared tardiness that would be obtained by the completion of the partial sequence corresponding to those nodes is calculated. If the lower bound is less than the lowest sum of weighted squared tardiness found so far for complete sequences (incumbent value) and the node does not represent a complete sequence, the node is retained for additional branching. If the lower bound is less than the incumbent value and all the jobs have been sequenced in the branch ending with the node (the node represents a complete sequence), then the incumbent value is updated to equal the lower bound, the sequence is recorded and the node is eliminated. If the lower bound is greater than the incumbent value, the node is eliminated. The algorithm uses a depth first strategy. This strategy selects for branching the node at the lowest level of the tree, breaking ties by choosing the node with the least lower bound.

Let σ be a post-partial sequence for a node in the branch-and-bound tree and U the set of jobs not included in σ. Since the completion time of the jobs in U will equal the sum of their processing times, the completion times and weighted squared tardiness of the jobs in σ can be calculated to obtain a lower bound. Let $Z(\sigma)$ equal the sum of the weighted squared tardiness of the jobs in σ. This lower bound is strengthened by including a job based lower bound for the jobs in set U. For each job j in set U, $t_{B(j)}+p_j$ is a lower bound on its completion time, and $\max\{t_{B(j)}+p_j-d_j,0\}^2w_j$ is a lower bound on its weighted squared tardiness, where $B(j)$ and $t_{B(j)}$ are as defined in Section 2. To obtain the lower bound on the completion of a post-partial sequence $\sigma(LB_\sigma)$, we add the job based lower bounds for the jobs in set U to $Z(\sigma)$:$LB_\sigma=\sum_{j\in U}\max\{t_{B(j)}+p_j-d_j,0\}^2w_j+Z(\sigma)$. The complexity of this lower bound is $O(n)$.

Several issues need to be considered when implementing the branch-and-bound algorithm. These include: preprocessing to reduce the problem size by applying some of the conditions, ensuring that not all of the optimal solutions are eliminated by applying the conditions and implementing the individual conditions.

Some of the conditions can be applied before the branch-and-bound algorithm starts to determine the precedence relations between pairs of jobs. Clearly this is the case with Proposition 1. However, if it is determined that job j is to precede job k using Proposition 1 then job j can be added to set $B(k)$ and job k can be added to set $A(j)$. These sets can be used to calculate minimum and maximum completion times for jobs, which can then be used in other conditions such as Propositions 2 and 3 to generate additional precedence relations between jobs. In some cases it may be possible by successively applying the conditions to generate an optimal solution before branching begins.

Care must be taken when the dominance conditions are applied that not all optimal solutions are eliminated. An example is if two jobs j and k have equal processing times, weights and due dates. In this case it does not matter which of the two jobs precedes the other and one of the orderings can be eliminated but not both orderings. A second example is if there are two or more unscheduled jobs that have due dates that will be greater than the completion time of the unscheduled jobs. In our implementation we sequence these jobs in EDD order so the job with the latest due date is sequenced at the end of this set.

Most of the dominance conditions are easy to implement but some require some thought. An example is Corollary 7. In our

implementation each time a job is scheduled the other unscheduled jobs are evaluated by using Corollary 7 to determine a maximum completion time for each of these jobs and then this information is retained in the node that is created. When the node is evaluated for branching if scheduling a job would cause it to be completed after its maximum completion time a branch is not created for that job.

4. Computational results

In this section, the computational experiments and results are presented. First, the set of test problems used in the computational tests is described. Then, the computational results are presented.

4.1. Experimental design

The computational tests were performed on a set of problems with 10, 15, 20, 25, 30 and 40 jobs. The approach used to generate the problems is the same as the one that is used in the OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html) for the linear weighted tardiness problem. These problems were randomly generated as follows. For each job j, an integer processing time p_j was generated from a uniform distribution [1, 100] and an integer weight w_j was generated from a uniform distribution [1, 10]. For each job j, an integer due date d_j was generated from the uniform distribution $[P(1-T-R/2), P(1-T+R/2)]$, where P is the sum of the processing times of all jobs, T is the tardiness factor and R is the range of due dates. The tardiness factor and the range of due dates were both set at 0.2, 0.4, 0.6, 0.8 and 1.0. For each combination of problem size n, T and R, 10 instances were randomly generated. Therefore, a total of 250 instances were generated for each problem size n.

Two versions of the branch-and-bound procedure described in Section 3 were created. The first procedure, referred to as D, includes all of the dominance conditions in Section 2 as well as the lower bound described in Section 3, while the second procedure, D', does not include the dominance conditions, but does include the lower bound described in Section 3. The procedures were coded in Turbo Pascal, and executed on a Dell Inspiron 1525 1.6 GHz Lap Top computer. The two procedures were applied to all the problems. For each procedure and each problem, we recorded the total time (in seconds) that was required to solve the problem, as well as the number of nodes generated. If a procedure was unable to solve a problem within 600 s, the procedure was terminated for that problem.

Also, Luh and Hoitomt [17]'s procedure for scheduling parallel machines was used on the data by setting the number of machines equal to one. This is a Lagrangian relaxation based procedure that relaxes the machine capacity constraint so that more than one job could process on a machine at any given time so a lower bound for the objective can be found. The procedure then sorts jobs by their starting times obtained from the relaxation and creates a feasible schedule to obtain an upper bound. The procedure uses subgradient optimization to update the Lagrangian multipliers for the next iteration and runs for a fixed number of iterations. For additional information about the procedure see Luh and Hoitomt [17].

4.2. Results

Table 1 shows the average time in seconds per problem (Sec.), the average number of nodes generated (# of Nodes) and the number of problems solved within the 600 s time limit (# S.) for each branch-and-bound procedure for each problem size (n). The

Table 1
Average seconds per problem, average number of nodes generated per problem and number solved.

	Algorithm					
	D			D'		
n	Sec.	# of Nodes	# S.	Sec.	# of Nodes	# S.
10	0.006	22.96	250	0.082	4121.76	250
15	0.013	64.84	250	44.8	1,710,880	245
20	0.061	289.3	250	259.2	7,100,322	149
25	0.185	675.3	250	314.5	5,556,447	124
30	0.818	2445.4	250	356.4	5,073,338	108
40	12.5	24954	250	394.5	4,830,179	90

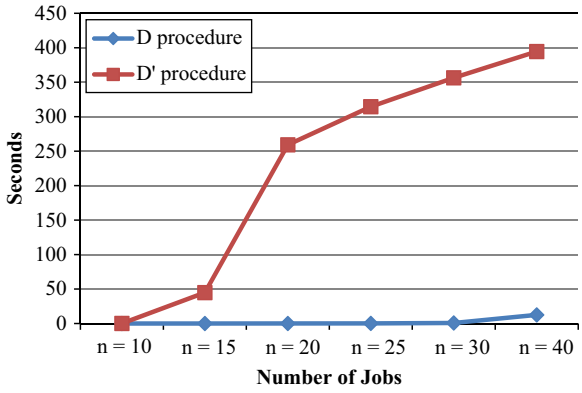

Fig. 4. Average time in seconds.

average time in seconds per problem required for each procedure for each problem size (n) is also shown in Fig. 4.

The results show that as the number of jobs increases the time required and the number of nodes generated increased for both branch-and-bound procedures. The results also show that, for all problem sizes, the D procedure used considerably less time, and generated only a small fraction of the number of nodes, when compared to the D' procedure. Furthermore, the D procedure was able to solve all of the problems in less than 600 s. The D' procedure was only able to solve all the problems with 10 jobs within the 600 s time limit.

As the number of jobs increases, the percentage of problems the D' procedure was able to solve within 600 s steadily decreased. The D' procedure solved 98% (60%, 50%, 43% and 36%) of the problems with 15 (respectively, 20, 25, 30 and 40, respectively) jobs, within 600 s or less. As can be seen in Fig. 4, the average number of seconds required by the D' procedure rises very rapidly between the 15 and 20 job problem sizes, and then does not rise as rapidly as the problem size increases. It should be noted, however, that the 600 s time limit is artificially restricting the time used by the D' procedure when n is greater than 20. These results show that the inclusion of the dominance conditions is important in increasing the efficiency of the algorithm: the dramatic reduction in the size of the search tree greatly offsets the computational effort involved in checking the dominance conditions. Therefore, the inclusion of the dominance conditions allows an exact solution to medium sized problems to be obtained relatively rapidly.

Table 2 shows results for Luh and Hoitomt [17]'s Lagrangian relaxation based procedure.

This table shows how the lower bound found by the Lagrangian relaxation compares to the upper bound and the optimal objective value, how the upper bound compares to the optimal objective value, and the number of times the procedure found an optimal

Table 2
Lagrangian relaxation results.

Problem size (n)	GAP		L.B. ERROR	U.B. ERROR		Average seconds
	%	#S. Verified	%	%	#S. Found	
10	23.58	26	22.85	2.42	151	0.20
15	20.98	30	20.23	3.02	116	0.84
20	19.46	33	18.34	4.24	94	2.24
25	18.05	34	16.97	3.94	85	4.85
30	19.97	35	17.86	4.93	85	9.18
40	17.70	36	16.44	4.46	78	25.0

solution as well as the number of times the procedure was able to verify an optimal solution. The table also shows the average number of seconds used by the procedure for each problem size. In columns two and three of the table information about the gap between the lower and upper bounds is provided. Column two provides the GAP%, which is the average percent that the lower bound found by the Lagrangian relaxation algorithm is below the upper bound value found by the heuristic used in the Lagrangian relaxation algorithm. Column three, #S. Verified is the number of optimal solutions that could be verified by the Lagrangian relaxation. An optimal solution is verified if the lower bound found by the Lagrangian relaxation equals the upper bound found by the Lagrangian relaxation. Column 4, L.B. ERROR % is the percent the lower bound found by the Lagrangian relaxation is less than the optimal objective value (which was found using the D procedure). Column five, U.B. ERROR % is the percent the value found by the Lagrangian based heuristic is above the optimal objective value and column 6, #S Found is the number of times the value found by the heuristic equals the optimal value (found using the D procedure). The last column (column 7) shows the average seconds used per problem by the Lagrangian relaxation procedure. These results show that using the branch-and-bound algorithm allows better solutions to be obtained for small and medium sized problems. The average error of the best solution found by the procedure compared to the optimal solution ranges from 2.42% for the problems with ten jobs to 4.93 percent. The average tends to rise as the number of jobs increases and the number of optimal solutions found by the procedure decreases as the number of jobs increases. For the problem sizes with 20 or more jobs the procedure was only able to find an optimal solution for less than 40% of the problems. Also the average time used by the branch-and-bound algorithm is less than that used by the Lagrangian relaxed based algorithm. These results demonstrate the value of the exact branch-and-bound procedure for small and medium sized problems.

Table 3 shows how the due date tightness (T) parameter affects the results. This table shows the average time in seconds per problem, the average number of nodes generated and the number of problems solved within the 600 s time limit for each procedure, and for each problem size (n) and each value of the due date tardiness parameter.

These results show that the due date tardiness parameter does affect the time required to solve problems. For both procedures the time required to solve problems increases as the T parameter increases for most problem sizes. The number of nodes generated by the D procedure also increased as the T parameter increases. The D procedure is able to solve the 40 job problems for $T=0.2$ or 0.4 very rapidly (an average of less than 0.2 s). These results indicate the D procedure could solve larger sized problems when the due dates are reasonably loose.

The number of problems the D' procedure solved within 600 s decreased as the T parameter increases. The D' procedure was not able to solve any of the problems within 600 s for $T=0.8$ or 1.0 when the number of jobs (n) was 25 or greater. These results indicate that problems are harder to solve when jobs tend to be very tardy.

Table 3
Average seconds per problem, average number of nodes generated per problem and number solved by due date tardiness parameter (t).

		Algorithm					
		D			D'		
n	T	Sec.	# of Nodes	# S.	Sec.	# of Nodes	# S.
10	0.2	0.002	4.40	50	0.004	74.60	50
	0.4	0.002	12.00	50	0.008	389.4	50
	0.6	0.008	22.80	50	0.042	20.4	50
	0.8	0.010	31.00	50	0.078	40.8	50
	1.0	0.008	44.60	50	0.278	13.992	50
15	0.2	0.002	7.00	50	0.202	10.120	50
	0.4	0.006	22.80	50	2.85	120.065	50
	0.6	0.010	54.00	50	20.3	778.080	49
	0.8	0.022	85.00	50	41.5	1,618,772	50
	1.0	0.024	155.4	50	159	6,027,366	46
20	0.2	0.008	9.00	50	0.034	10.7	50
	0.4	0.016	40.00	50	26.3	942.304	48
	0.6	0.034	152.6	50	150	4,836,508	41
	0.8	0.096	501.2	50	520	15,922,070	10
	1.0	0.152	743.8	50	600	13,799,690	0
25	0.2	0.008	10.60	50	0.072	21.3	50
	0.4	0.020	69.20	50	65.0	2,128,916	45
	0.6	0.110	374.4	50	307	8,622,329	29
	0.8	0.420	15.3	50	600	10,739,974	0
	1.0	0.366	13.9	50	600	6,288,841	0
30	0.2	0.006	12.80	50	0.130	38.2	50
	0.4	0.040	100.4	50	87.2	2,542,399	45
	0.6	0.338	947.4	50	495	11,261,892	13
	0.8	1.61	48.3	50	600	7,392,444	0
	1.0	2.09	63.4	50	600	4,166,072	0
40	0.2	0.020	12.40	50	0.396	91.9	50
	0.4	0.170	302.8	50	174	4,232,004	39
	0.6	2.85	54.5	50	597	9,819,047	1
	0.8	21.6	44.356	50	600	4,424,347	0
	1.0	38.1	74.594	50	600	5,666,387	0

Table 4 shows how the due date range (R) parameter affects the results. This table shows the average time in seconds per problem, the average number of nodes generated and the number of problems solved within the 600 s time limit for each procedure, for each problem size (n) and each value of the due date range parameter.

The results of this table show that the due date range parameter does not have a great effect on the time required by the procedures to solve problems. This is somewhat surprising as it would seem that as the range of due dates increased it would be easier to sequence the jobs. It appears that the due date tightness factor has a much stronger affect on the time and number of nodes required by the algorithms.

Since the branch-and-bound algorithm performed much better when the dominance conditions of Section 2 were included, an additional analysis was performed to determine the effect of individual dominance conditions. This was done by creating eight additional procedures that vary as to which dominance conditions are included. P1′ includes all of the dominance conditions except Proposition 1, P2G′ includes all the dominance conditions except Proposition 2 and Corollaries 1–4, P3G′ includes all the dominance conditions except Proposition 3 and Corollaries 5 and 6, P4′ includes all the dominance conditions except Proposition 4, P5′ includes all the dominance conditions except Proposition 5, P6G′ includes all the dominance conditions except Proposition 6

Table 4
Average seconds per problem, average number of nodes generated per problem and number solved by due date range parameter (R).

		Algorithm					
		D			D'		
n	R	Sec.	# of Nodes	# S.	Sec.	# of Nodes	# S.
10	0.2	0.008	22.20	50	0.042	2064	50
	0.4	0.004	25.60	50	0.056	2958	50
	0.6	0.004	23.20	50	0.082	4013	50
	0.8	0.008	22.80	50	0.098	4939	50
	1.0	0.006	21.00	50	0.132	6635	50
15	0.2	0.014	79.00	50	36.0	1,390,070	49
	0.4	0.012	65.80	50	17.6	703,659	50
	0.6	0.010	55.40	50	35.2	1,355,755	50
	0.8	0.012	60.60	50	57.2	2,174,067	49
	1.0	0.016	63.40	50	78.2	2,930,852	47
20	0.2	0.084	427.0	50	224	7,106,992	33
	0.4	0.062	296.6	50	206	5,826,882	34
	0.6	0.070	315.6	50	247	6,359,420	30
	0.8	0.052	236.8	50	272	7,194,504	29
	1.0	0.038	171.2	50	347	9,013,811	23
25	0.2	0.202	790.2	50	280	5,606,270	28
	0.4	0.196	719.6	50	270	4,848,564	29
	0.6	0.192	694.2	50	289	4,779,576	27
	0.8	0.162	562.6	50	323	5,392,525	24
	1.0	0.172	609.8	50	411	7,155,298	16
30	0.2	1.49	4596	50	334	5,704,068	24
	0.4	0.766	2348	50	331	5,210,939	25
	0.6	0.758	2184	50	357	5,000,803	21
	0.8	0.764	2168	50	360	4,462,595	21
	1.0	0.316	932.2	50	400	4,988,286	17
40	0.2	6.28	12,869	50	440	7,240,588	15
	0.4	10.0	19,965	50	384	5,175,490	20
	0.6	18.9	38,285	50	367	4,241,935	20
	0.8	15.9	30,111	50	385	4,195,600	18
	1.0	11.6	23,459	50	396	3,297,280	17

Table 5
Number of problems solved, average seconds per problem, number of nodes generated and percentage comparison with the D procedure for $n=40$.

Procedure	Solved	Seconds		# of Nodes	
	# S.	Sec.	% inc.	Nodes	% inc.
P1′	248	31.27	149	53,697	115
P2G′	250	14.48	15	29,323	18
P3G′	248	33.68	169	73,686	195
P4′	247	31.47	151	64,225	158
P5′	247	20.45	63	44,741	79
P6G′	243	52.58	319	307,010	1130
P7′	225	98.60	686	226,126	815
P8′	250	12.11	−3.43	24,954	0.06
D	250	12.54	0	24,938	0

and Corollary 7, P7′ includes all the dominance conditions except Proposition 7 and P8′ includes all of the dominance conditions except Proposition 8. These procedures were used on all the problems containing 40 jobs and the time in seconds required to solve each problem, as well as the number of nodes generated, were recorded. If a procedure was unable to solve a problem within 600 s the procedure was terminated for that problem.

Table 5 shows the average seconds per problem, the average number of nodes generated and the number of problems solved for the problem set with 40 jobs. The table also shows the percentage increase (% inc.) in the time required and number of nodes generated versus the D procedure.

This table shows that seven of the eight procedures missing one or more dominance conditions require more time and generated more nodes than the D procedure. The P8′ procedure was the lone exception as it required slightly less time but also generated more nodes than the D procedure. Also, only the P2G′ and P8′ procedures were able to solve all of the problems within the 600 s time limit. The omission of Propositions 8 and 2 and its related corollaries had the least impact on the efficiency of the branch-and-bound algorithm. The P6G′ and P7′ procedures resulted in the largest percentage increase in the time required to solve problems and the number of nodes generated compared to the D procedure. The P7′ procedure also solved the least problems within the 600 s time limit.

To summarize these results, all of the dominance conditions with the exception of Proposition 8 were found to improve the algorithm's efficiency. Including Proposition 6 and the associated Corollary 7 and Proposition 7 generally improves the efficiency of the branch-and-bound algorithm the most. It should be noted that the effect of including a dominance condition could vary based on the tightness and range of due dates. For example it was found that the dominance condition stated in Proposition 5 has a much larger effect when the due date tightness parameter is low ($T=0.2$ or 0.4). The reason for this is that when due dates are loose it is easier to find jobs that if sequenced at the end would still be early and hence many branches can be eliminated from the branch-and-bound tree using this condition.

In order to test its limits, the D procedure was performed on an additional data set that included problems with larger numbers of jobs. The data set was taken from the OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html). This data set was created in the same manner as the one described in Section 4.1 and includes five problems for each n, T and R. The values of n in the data set are 40, 50 and 100 and the same values of the parameters parameters of T and R were tested. As in the earlier test the D procedure was run with a time limit of 600 s. Table 6 shows the results of this test. Since the T parameter was shown to affect the results, the table shows the average number of seconds the D procedure required on problems for each n and T as well as the average across all the T values for each n.

The results show the D procedure solved all of the problems with 40 jobs and all but one of the problems with 50 jobs but solved less than 40% of the problems with 100 jobs. The average processing time required to solve problems also increased as the number of jobs increased. The problems with 50 jobs required on average over six times as much time as the jobs with 40 problems. Also the tardiness factor (T) significantly affects the time required by the D procedure to solve the problems. As T increases the time required by the D procedure generally increases. Also the D procedure was not able to solve any problems with 100 jobs within 600 s when T was equal to 0.6, 0.8 or 1.0 but was able to solve a large percentage of the problems with 100 jobs within 600 s when the tardiness factor was equal to 0.2 or 0.4 (94%). Based on these results we conclude that the D procedure can

Table 6
Average seconds per problem and number of problems solved for the D algorithm for the ORLIB dataset.

Problem size (n)	T										All T	
	0.2		0.4		0.6		0.8		1.0			
	Sec.	# S.	Sec.	# S.	Sec.	# S.	Sec.	# S.	Sec.	# S.	Sec.	# S.
40	0.01	25	0.15	25	2.24	25	16.9	25	17.1	25	7.27	125
50	0.03	25	0.49	25	19.4	25	88.7	25	121	24	45.9	124
100	0.43	25	125	22	600	0	600	0	600	0	385	47

solve problems with up to 50 jobs in a reasonable amount of time and problems with up to 100 jobs in a reasonable amount of time if the due dates are relatively loose.

5. Conclusion

This paper considers the single machine sequencing problem when the objective is to minimize the sum of weighted squared tardiness. A branch-and-bound algorithm was presented as well as various dominance conditions that can be incorporated into the branch-and-bound algorithm.

Nine procedures were created and tested on randomly generated problems. The test problems were of a variety of different sizes in terms of the number of jobs, different degrees of tightness of due dates and different ranges of due dates.

The results of the tests showed that problems with up to 40 jobs can be solved in a reasonable amount of time on a personal computer. Also, the results show larger sized problems can be solved in a reasonable amount of time if due dates are not very tight. The results show that the proposed dominance conditions greatly improve the efficiency of the branch-and-bound algorithm. A possibility for future research would be the investigation of other possible approaches for obtaining lower bounds for the problem that could be incorporated into a branch-and-bound algorithm with improved efficiency. Also, it would be interesting to see if the approaches used in this research could be extended to other scheduling problems where minimizing the sum of weighted squared tardiness is the objective, such as scheduling on parallel machines.

References

[1] Abdul-Razaq TS, Potts CN, Van Wassenhove LN. A survey of algorithms for single machine total weighted tardiness scheduling problems. Discrete Applied Mathematics 1990;26(2–3):235–53.
[2] Bagga PC, Kalra KR. A node elimination procedure for Townsend's algorithm for solving the single machine quadratic penalty function scheduling problem. Management Science 1980;26:633–6.
[3] Baker KR. Introduction to Sequencing and Scheduling. New York: John Wiley and Sons, Inc.; 1974.
[4] Black PE. Branch and bound. In: Black Paul E, editor. Dictionary of Algorithms and Data Structures [online]. U.S. National Institute of Standards and Technology; 2011. [September 12, 2005], (accessed April 11). Available from 〈http://www.itl.nist.gov/div897/sqg/dads/HTML/branchNbound.html〉.
[5] Elmagraby SE. The one-machine sequencing problem with delay costs. Journal of Industrial Engineering 1968;19:105–8.
[6] Emmons H. One-machine scheduling to minimize certain functions of job tardiness. Operations Research 1969;17:701–15.
[7] Fisher ML. Optimal solution of scheduling problems using Lagrangian multipliers: part I. Operations Research 1973;21(5):1114–27.
[8] Fisher ML. Optimal solution of scheduling problems using Lagrangian multipliers: part II. Operations Research 1973;21(6):294–317.
[9] Gelders L, Kleindorfer PR. Coordinating aggregate and detailed scheduling in the one-machine job shop I: theory. Operations Research 1974;22:46–60.
[10] Gelders L, Kleindorfer PR. Coordinating aggregate and detailed scheduling in the one-machine job shop II: computation and structure. Operations Research 1975;23:312–24.
[11] Gupta SK, Sen T. Minimizing a quadratic function of job lateness on a single machine. Engineering Costs and Production Economics 1983;7:187–94.
[12] Gupta SK, Sen T. On the single machine scheduling problem with quadratic penalty function of completion times: an improved branching procedure. Management Science 1984;30:644–7.
[13] Hoitomt DJ, Luh PB, Max E, Pattipati KR. Scheduling jobs with simple precedence constraints on parallel machines. IEEE Control Systems Magazine 1990:34–40.
[14] Jouglet A, Carlier J. Dominance rules in combinatorial optimization problems. European Journal of Operational Research 2011;212:433–44.
[15] Kanet JJ. New precedence theorems for one-machine weighted tardiness. Mathematics of Operations Research 2007; August:579–88.
[16] Lawler EL. Efficient implementation of dynamic programming algorithms for sequencing problems. Report BW 106; Mathematisch Centrum. Amsterdam, The Netherlands; 1979.
[17] Luh PB, Hoitomt DJ. Scheduling of manufacturing systems using the lagrangian relaxation technique. IEEE Transactions on Automatic Control 1993;38:1066–79.
[18] Potts CN, van Wassenhove LN. A branch and bound algorithm for the total weighted tardiness problem. Operations Research 1985;33:363–77.
[19] Rachamadugu RMV. A note on the weighted tardiness problem. Operations Research 1987;35:450–1.
[20] Rinnooy Kan AHG, Lageweg JK, Lenstra. Minimizing total costs in one-machine scheduling. Operations Research 1975;1975(23):908–27.
[21] Schrage LE, Baker KR. Dynamic programming solution of sequencing problems with precedence constraints. Operations Research 1978;26:444–9.
[22] Schaller J. Minimizing the sum of squares lateness on a single machine. European Journal of Operational Research 2002;143:64–79.
[23] Schaller J. Single machine scheduling with early and quadratic tardy penalties. Computers and Industrial Engineering 2004;46:511–32.
[24] Schild A, Fredman IJ. Scheduling tasks with deadlines and nonlinear loss functions. Management Science 1962;9:73–81.
[25] Sen T, Dileepan P, Lind MR. Minimizing a weighted quadratic function of job lateness in the single machine system. International Journal of Production Economics 1995;42:237–43.
[26] Sen T, Sulek JM, Dileepan P. Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. International Journal of Production Economics 2003;83:1–12.
[27] Srinivasan V. A hybrid algorithm for the one-machine sequencing problem to minimize total tardiness. Naval Research Logistics Quarterly 1971;18:317–27.
[28] Su L-H, Chang P-C. A heuristic to minimize a quadratic function of job lateness on a single machine. International Journal of Production Economics 1998;55:169–75.
[29] Sun X, Noble JS. An approach to job shop scheduling with sequence-dependent setups. Journal of Manufacturing Systems 1999;18:416–30.
[30] Sun X, Noble JS, Klein CM. Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. IIE Transactions 1999;31:113–24.
[31] Szwarc W, Posner ME, Liu JJ. The single machine problem with quadratic cost function of completion times. Management Science 1988;34:1480–8.
[32] Taguchi G. Introduction to Quality Engineering. Tokyo, Japan: Asian Productivity Organization; 1986.
[33] Thomalla CS. Job shop scheduling with alternative process plans. International Journal of Production Economics 2001;74:125–34.
[34] Townsend W. The single machine problem with quadratic penalty function of completion times: a branch and bound solution. Management Science 1978;24:530–4.
[35] Valente JMS. Heuristics for the single machine scheduling problem with early and quadratic tardy penalties. European Journal of Industrial Engineering 2007;1:431–48.
[36] Valente JMS. Beam search heuristics for the single machine scheduling problem with linear earliness and quadratic tardiness costs. Asia-Pacific Journal of Operational Research 2009;26:319–39.
[37] Valente JMS. An exact approach for the single machine scheduling problem with linear early and quadratic tardy penalties. Asia-Pacific Journal of Operational Research 2008;25:169–86.
[38] Valente JMS, Gonçalves JF. A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. Computers & Operations Research 2009;36:2707–15.
[39] Valente JMS, Schaller JE. Improved heuristics for the single machine scheduling problem with linear early and quadratic tardy penalties. European Journal of Industrial Engineering 2010;4:99–129.
[40] Wagner BJ, Davis DJ, Kher H. The production of several items in a single facility with linearly changing demand rates. Decision Sciences 2002;33:317–46.