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

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: \( Z = \sum_{j=1}^{n} w_j T_j^2 \). 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
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. Rachmadadugu [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 Whitney 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 scheduling 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

\[ Z_j = w_j t_j^2. \]

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 \( D_j(t)=w(t^2)+1 \) if \( t \geq d_j \). Let \( D_j^*(t) \) be the increase of \( D_j(t) \) if \( C_j \) is increased from \( t \) to \( t+1 \). \( D_j^*(t)=0 \) if \( t < d_j \), and \( D_j^*(t)=2w_j \) 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 \( k \). 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 \( S'(M_{jk}(S)=M_{jk}(S')) \).

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 \( t_{lbk} = \sum_{j=1}^{k} \min(p_j, b_j) \) and \( t_{ubk} = \sum_{j=1}^{k} \min(p_j, b_j) \). \( t_{ubk} \) is a lower bound on the start time of job \( k \) and \( t_{lbk} \) is an upper bound on the completion time for job \( j \).

As shown in Fig. 1, \( C_j(S)=C_j(S') \). 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 \( C_j(S) \) it only needs to be shown that the increase in job \( j \)'s weighted squared tardiness when its completion time is increased from \( C_j(S) \) 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 \( D_j(t) \geq D_j(t) \) for \( C_j(S') \leq t \leq C_j(S) \) 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 $M_{k,S}(S)$ 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 $C_{j}(S) = C_{j}(S')$ and $C_{j}(S') = C_{j}(S) + p_{j}$ where $p_{j} < C_{j}(S')$. Then, we have $Z_{j}(S)-Z_{j}(S') = -w_{j}max(C_{j}(S')-d_{j},0)^{2}$ and $Z_{k}(S')-Z_{k}(S) = -w_{k}max(C_{k}(S)-d_{k},0)^{2}$. Since $D_{k}(t) > D_{k}(t)$ for $C_{j}(S') \leq t \leq C_{j}(S)$, then $Z_{j}(S)-w_{j}max(C_{j}(S')-d_{j},0)^{2} \geq Z_{k}(S')-Z_{k}(S)$, 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 $t = C_{j}(S)$. Since $w_{j} \geq w_{k}$ then $D_{j}(t) \geq D_{k}(t)$ for $t > C_{j}(S)$. Therefore, $D_{j}(t) \geq D_{k}(t)$ for $C_{j}(S) \leq t \leq C_{j}(S)$, 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 $p_{j} \leq p_{k}$, $d_{j} < d_{k}$, and $w_{j} > w_{k}$.

Proposition 2. If $p_{j} \leq p_{k}$, $d_{j} < d_{k}$, $w_{j} > w_{k}$, $C_{j}(S') > d_{j}$, and $w_{j}2max((C_{j}(S')-d_{j},0)+1) \geq w_{k}(2max((C_{j}(S')-d_{j},0)+1))$ 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 $t = C_{j}(S)$ (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 $w_{j}2max((C_{j}(S')-d_{j},0)+1) \geq w_{j}2max((C_{j}(S')-d_{j},0)+1)$, which implies $D_{j}(t) \geq D_{j}(t)$ for $t = C_{j}(S)$. Also, since $w_{j} > w_{k}$, we have $D_{j}(t) \geq D_{j}(t)$ for all $t \geq C_{j}(S)$. Therefore, $D_{j}(t) \geq D_{j}(t)$ for $C_{j}(S) \leq t \leq C_{j}(S)$ 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 $i$ 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 $p_{j} < p_{k}$, $d_{j} < d_{k}$, $w_{j} > w_{k}$, then $t_{k}(S)+p_{k} > d_{j}$ and $w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1) \geq w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1)$ then job $j$ can be added to set $B(k)$.

Corollary 2. If $p_{j} < p_{k}$, $d_{j} < d_{k}$, $w_{j} > w_{k}$, then $t_{k}(S)+p_{k} > d_{j}$, $w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1) \geq (w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1)^{2} + w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1)^{2}) \geq (w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1)^{2} + w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1)^{2})$ then job $j$ can be added to set $B(k)$.

Proof. We have $C_{j}(S) \geq t_{k}(S)+p_{k}$. Also the conditions of the corollary together imply that $w_{j}max((C_{j}(S)-d_{j},0)+1) \geq w_{j}max((C_{j}(S)-d_{j},0)+1)$. 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 3. If $p_{j} < p_{k}$, $d_{j} < d_{k}$, $w_{j} > w_{k}$, then $w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1) \geq w_{j}max((t_{k}(S)+p_{k}+d_{j},0)+1)$ then job $j$ can be added to set $B(k)$.

Proof. We have $C_{j}(S) \geq t_{k}(S)+p_{k}$. Also the conditions of the corollary together imply that $w_{j}max((C_{j}(S)-d_{j},0)+1) \geq w_{j}max((C_{j}(S)-d_{j},0)+1)$.

Proof. We have $C_{j}(S) \geq t_{k}(S)+p_{k}$. Also the conditions of the corollary together imply that $w_{j}max((C_{j}(S)-d_{j},0)+1) \geq w_{j}max((C_{j}(S)-d_{j},0)+1)$. 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 $S'$.
Proof. This problem shows that if the rate of increase in job j's weighted tardiness is greater than or equal to the rate of increase in job k's at time \( t = C_i(S) \) (job k's completion time in sequence S), then the weighted squared tardiness of job k will be less than or equal to that of sequence S. We have

\[ w_\text{j}(2 \max(C_i(S) - d_j, 0) + 1) \geq w_\text{j}(2 \max(\max(C_i(S) - d_j, 0), 0) + 1) \]

where \( w_\text{j} \) is the weighted tardiness of job j.

**Proposition 3 and Corollaries 5 and 6 provide conditions for job j preceding job k when \( p_i \leq p_j \) and \( d_i < d_j \), but \( w_i > w_j \).**

Proof. Let \( t=\max\{t_B(k) + p_k/C_0, d_k, 0\} \geq \max\{(t_B(k) + p_k/C_0, d_k, 0)\}^2 \) and \( w_\text{j}(2 \max(t_B(k) + p_k/C_0, d_k, 0) + 1) \geq w_\text{j}(t_B(k) + p_k/C_0, d_k, 0) + 1 \) then job k can be added to set \( A(j) \).
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. \(\square\)

Proposition 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 ∈ U such that \(d_j \geq t_j\), 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 that 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 \(t_j\) 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 \(d_j \geq t_j\), \(Z(S) = 0\) 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. \(\square\)

**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 \(t_U - p_k\)).

![Fig. 2. Insert-after strategy for proof of Proposition 5.](image)

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 \(t_U + p_k\) 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 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 \(j \leq k\). If \((w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2) > (w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2)\), 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 j 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 \(M_j(S')\) are earlier in the schedule associated with sequence S than under sequence \(S'\), to prove this theorem it only needs to be shown that \(Z(S) + Z(S') \leq Z(S) + Z(S')\). Since \(t_U \leq P_k\) is a lower bound on the completion time of job k, \(C(S') \geq C(S) + P_k\), and therefore \((w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2) > (w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2)\). Also, \((Z(S) - Z(S') = (w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2)\). Since the propositions conditions state that \((w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2) > (w_{\text{max}}(t_U - d_j, 0)^2 - w_{\text{max}}(t_U - d_k, 0)^2)\), have \(Z(S) - Z(S') = (Z(S) - Z(S')) \leq (Z(S) - Z(S'))\). 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. \(\square\)

The following notation is used to develop the condition in Corollary 7. Let \(Z(t_U) = w_{\text{max}}(t_U - d_j, 0)^2\), \(Z_0 = w_{\text{max}}(t_U - p_k)^2\), and \(Z(0) = Z(t_U) - Z(0)\) and \(Z(0) = w_{\text{max}}(t_U - d_j, 0)^2\). 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 ∈ 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 \(C_k \leq \max[Z_4(t_U) - Z_4(t_U)]/w_k + d_k\).

**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 j 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 \( M_j(S') \) 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 \( C_j = C_k \), \( (S) > \sqrt[m]{\max[Z_j(t_0) - \Delta z_k]} w_k + d_k \) then \( Z_j(S) + Z_j(S') < Z_j(S') + Z_j(S) \) 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. \( Z_j(S) + Z_j(S') < Z_j(S') + Z_j(S) \) implies \( Z_j(S) - Z_j(S') < Z_j(S') - Z_j(S) \). Also, \( Z_j(S) - Z_j(S') = \Delta z_j \) and \( t_j = t_0 \) so \( Z_j(S) - Z_j(S') = \max[Z_j(t_0) - \Delta z_j] \), then \( Z_j(S) - Z_j(S') < Z_j(S') - Z_j(S) \). Since \( Z_j(S) = w_m \max[C_j(S) - d_k, 0]^2 \), \( C_j(S) > \sqrt[m]{\max[Z_j(t_0) - \Delta z_k]} w_k + d_k \) implies \( Z_j(S) - Z_j(S') < Z_j(S') - Z_j(S) \), 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 \( B(k) \) be the set of jobs that are sequenced before the jobs \( j \) and \( k \) in an optimal sequence. Also, let \( t_{B(k)} = \sum_{j \in B(k)} p_j \).

**Proposition 7.** If jobs \( j \) and \( k \) are to be sequenced adjacent to each other after the set of jobs in \( B(k) \), and \( (w_m \max[(B(k) + p_j - d_j - d_k)^2 + w_m \max[b_j + p_j - d_j - d_k]^2] \leq (w_m \max[b_j + p_j - d_j - d_k]^2 + w_m \max[(B(k) + p_j - d_j - d_k)^2]^2) \), 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 \( Z_j(S) + Z_j(S') < Z_j(S') + Z_j(S) \). Therefore, the total weighted tardiness of sequence \( S' \) is less than that of sequence \( S \), and job \( j \) precedes job \( k \).

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

**Proof.** Let \( U \) be a set of unsequenced jobs that are not included in post-subpartial subsequences \( \sigma \) and \( \sigma' \) and will be sequenced at the beginning of a complete sequence. Also, let \( t_{U} = \sum_{j \in U} p_j \), which is the completion time of the jobs in set \( U \) and the start time of the first job in \( \sigma \) and \( \sigma' \). Let \( \pi \) be any initial partial subsequence consisting of the jobs in set \( U \). \( Z(\sigma') + Z(\pi) < Z(\sigma') + Z(\pi) \) for any \( \pi \) if \( Z(\sigma) < Z(\sigma') \) therefore \( \sigma \) will result in a lower total weighted tardiness if it is used instead of \( \sigma' \) as a post-subpartial 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 \( p \)th level in the branch-and-bound tree corresponds to a partial sequence (and the associated sub-problem) for the \( p \) 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 \( q \)th 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 \( \sigma \) be a post-subpartial sequence for a node in the branch-and-bound tree and \( U \) the set of jobs not included in \( \sigma \). 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 \( \sigma \) can be calculated to obtain a lower bound. Let \( Z(\sigma) \) equal the sum of the weighted squared tardiness of the jobs in \( \sigma \). 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 \( \max[(B(j) + p_j - d_j - 0)][w_j] \) 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-subpartial sequence \( \sigma(L_B) \), we add the job based lower bounds for the jobs in set \( U \) to \( Z(\sigma) \): \( L_B = \sum_{j \in U} \max((B(j) + p_j - d_j - 0)]w_j + Z(\sigma) \). 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/~mastajb/jeb/orlib/wtinfor.html) for the linear weighted tardiness problem. These problems were randomly generated as follows. For each job the 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 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
The D procedure is able to solve the 40 job problems for indicate the D procedure could solve larger sized problems when 0.4 very rapidly (an average of less than 0.2 s). These results by the D procedure also increased as the increases for most problem sizes. The number of nodes generated the time required to solve problems increases as the due date tardiness parameter. 

For the problem sizes with 20 or more jobs the procedure was only found by the procedure decreases as the number of jobs increases. The number of optimal solutions with ten jobs to 4.93 percent. The average tends to rise as the very tardy. 

Table 2 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 per problem, and number solved by due date tardiness parameter (T). 

Table 3 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, P2 includes all the dominance conditions except Proposition 2 and Corollaries 1–4, P3 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, P6 includes all the dominance conditions except Proposition 6.
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 $n$. The values of $n$ in the data set are 40, 50 and 100 and the same values of the parameters $T$ 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 4
Average seconds per problem, average number of nodes generated per problem and number solved by due date range parameter ($R$).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>D</th>
<th>Dy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$R$</td>
<td>Sec.</td>
</tr>
<tr>
<td>10</td>
<td>0.2</td>
<td>0.008</td>
</tr>
<tr>
<td>0.4</td>
<td>0.004</td>
<td>25.60</td>
</tr>
<tr>
<td>0.6</td>
<td>0.004</td>
<td>23.20</td>
</tr>
<tr>
<td>0.8</td>
<td>0.008</td>
<td>22.80</td>
</tr>
<tr>
<td>1.0</td>
<td>0.006</td>
<td>21.00</td>
</tr>
</tbody>
</table>

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 | 1,719,504 | 29 |
| 1.0 | 0.038 | 171.2 | 50 | 347 | 9,013,811 | 23 |

25 | 0.2 | 0.202 | 790.2 | 50 | 280 | 5,200,720 | 28 |
| 0.4 | 0.196 | 719.6 | 50 | 270 | 4,884,564 | 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 | 353 | 2,101,939 | 25 |
| 0.6 | 0.758 | 2184 | 50 | 350 | 2,000,803 | 31 |
| 0.8 | 0.864 | 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 | 2,977,280 | 17 |

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

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Solved</th>
<th>Seconds</th>
<th># of Nodes</th>
<th>% inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^1$</td>
<td>248</td>
<td>31.27</td>
<td>149</td>
<td>53,697</td>
</tr>
<tr>
<td>$P2G$</td>
<td>250</td>
<td>14.48</td>
<td>15</td>
<td>29,323</td>
</tr>
<tr>
<td>$P3G$</td>
<td>248</td>
<td>33.68</td>
<td>169</td>
<td>73,686</td>
</tr>
<tr>
<td>$P^4$</td>
<td>247</td>
<td>31.47</td>
<td>151</td>
<td>64,225</td>
</tr>
<tr>
<td>$P5$</td>
<td>247</td>
<td>20.45</td>
<td>63</td>
<td>44,741</td>
</tr>
<tr>
<td>$P6G$</td>
<td>243</td>
<td>52.58</td>
<td>319</td>
<td>307,010</td>
</tr>
<tr>
<td>$P^7$</td>
<td>225</td>
<td>98.60</td>
<td>686</td>
<td>226,126</td>
</tr>
<tr>
<td>$P8$</td>
<td>250</td>
<td>12.11</td>
<td>3.43</td>
<td>24,954</td>
</tr>
<tr>
<td>$D$</td>
<td>250</td>
<td>12.54</td>
<td>0</td>
<td>24,938</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Problem size ($n$)</th>
<th>T</th>
<th>All T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>40</td>
<td>0.01 25</td>
<td>0.15 25</td>
</tr>
<tr>
<td>50</td>
<td>0.03 25</td>
<td>0.49 25</td>
</tr>
<tr>
<td>100</td>
<td>0.43 25</td>
<td>0.25 22</td>
</tr>
</tbody>
</table>
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