IIE Transactions

ISSN: 0740-817X (Print) 1545-8830 (Online) Journal homepage: http://www.tandfonline.com/loi/uiie20

An efficient algorithm for the single machine total tardiness problem

BARBAROS Ç. TANSEL , BAHAR Y. KARA & IHSAN SABUNCUOGLU

To cite this article: BARBAROS Ç. TANSEL , BAHAR Y. KARA & IHSAN SABUNCUOGLU (2001) An efficient algorithm for the single machine total tardiness problem, IIE Transactions, 33:8, 661-674, DOI: 10.1080/07408170108936862

To link to this article: https://doi.org/10.1080/07408170108936862

IIE Transactions (2001) **33**, 661–674

An efficient algorithm for the single machine total tardiness problem

BARBAROS Ç. TANSEL, BAHAR Y. KARA and IHSAN SABUNCUOGLU

Department of Industrial Engineering, Bilkent University, Bilkent 06533, Ankara, Turkey
E-mail: barbaros@bilkent.edu.tr or bkara@bilkent.edu.tr or sabun@bilkent.edu.tr

Received November 1999 and accepted November 2000

This paper presents an exact algorithm for the single machine total tardiness problem ($1//\sum T_i$). We present a new synthesis of various results from the literature which leads to a compact and concise representation of job precedences, a simple optimality check, new decomposition theory, a new lower bound, and a check for presolved subproblems. These are integrated through the use of an equivalence concept that permits a continuous reformation of the data to permit early detection of optimality at the nodes of an enumeration tree. The overall effect is a significant reduction in the size of the search tree, CPU times, and storage requirements. The algorithm is capable of handling much larger problems (e.g., 500 jobs) than its predecessors in the literature (≤ 150). In addition, a simple modification of the algorithm gives a new heuristic which significantly outperforms the best known heuristics in the literature.

1. Introduction

In this paper, we present an exact algorithm that significantly advances the computational state-of-the-art on the single machine total tardiness scheduling problem, $1//\sum T_i$. In this problem, n jobs with known processing times and due dates must be sequenced on a single continuously available machine. If a job is completed after its due date, it is considered tardy, and is charged a penalty equal to its completion time minus its due date. The object is to find a sequence that minimizes the total tardiness. The weighted version of the problem is strongly NP-hard (Lenstra *et al.*, 1977). The version with unit weights which we study is NP-hard in the ordinary sense (Du and Leung, 1990). A pseudo-polynomial time dynamic programming algorithm is available (Lawler, 1977), whose time bound is $O(n^4 \sum_j p_j)$ or

$$O(n^5 \max_j p_j),$$

where p_j is the processing time of job j and n the number of jobs. The method relies on a decomposition theorem proved by Lawler (1977), and refined and strengthened by Potts and van Wassenhove (1982, 1987).

$1//\sum T_i$ was initially introduced by McNaughton (1959) and the first thorough study done by Emmons (1969), who proves dominance theorems that must be satisfied by at least one optimal schedule. These theorems are later generalized to non-decreasing cost functions (Rinnooy Kan *et al.*, 1975). Exact methods rely either on branch and bound (Schwimer, 1972; Rinnooy Kan *et al.*, 1975; Fisher, 1976; Picard and Queyranne, 1978; Sen *et al.*, 1983; Potts and van Wassenhove, 1985; Sen and Borah, 1991; Szwarc and Mukhopadhyay, 1996) or on dynamic programming (Srinivasan, 1971; Lawler, 1977; Schrage and Baker, 1978; Potts and van Wassenhove, 1982, 1987).

Even though this problem has been around for more than 30 years, relatively little progress seems to have been made in developing effective computational procedures for exact solutions. Early algorithms (e.g., Fisher, 1976) were able to solve problems with about 50 jobs. The dynamic programming-based algorithm of Potts and van Wassenhove (1982) is able to solve problems with 100 jobs and the branch and bound algorithm of Szwarc and Mukhopadhyay (1996) problems with 150 jobs. An important bottleneck in handling larger-sized problems is the core storage requirement (Potts and van Wassenhove, 1982). The dynamic programming algorithms, developed for this problem require $O(2^n)$ storage in the worst case, even though efficient bookkeeping leads to reduced storage in most implementations. Branch and bound algorithms that use job precedences (Emmons, 1969) at every subproblem requires $O(n^2)$ storage per subproblem. For a depth-first search that implements cycle prevention rules (needed for consistent application of Emmons' theorems) this amounts to a total core storage of $O(n^3)$ if one keeps an $O(n^2)$ precedence record at every level of the search tree. In contrast, the algorithm that we propose requires $O(n)$ storage for job precedences, which results in a total

core storage requirement of $O(n^2)$. Hence, computational difficulties related to limited storage begin to arise at much larger n in our branch and bound algorithm than in the existing algorithms. In addition, an efficient and simplified way of handling the data allows us to automatically enforce cycle prevention which is a computationally expensive but unavoidable task for the existing methods.

With these features, the proposed algorithm is capable of handling much larger problems (e.g., 500 jobs) than its predecessors in the literature. The CPU times based on 2800 test problems indicate that problems with 500 jobs can be solved to optimality with a 87.5% success rate while those with 400 and 300 jobs can be solved optimally with a 95 and 100% success rate, respectively. The test sets are generated using the method of Fisher (1976), which has been used in the literature to evaluate the previous state-of-the-art algorithms (e.g., Potts and van Wassenhove, 1982; Szwarc and Mukhopadhyay, 1996). The algorithm and the test sets are available in the web site http :*www.bilkent.edu.tr*\~*bkara*.

The proposed algorithm is a depth-first branch and bound algorithm with three major phases: β-sequence construction and β-test, decompositions, and fathoming tests. The β-sequence constructed in the first phase of the algorithm is a critically important sequence which often solves the problem optimally. The β-test is a simple check for optimality of the β-sequence. If the test is passed the optimal sequence is at hand. Otherwise, the algorithm continues with the decomposition phase that utilizes three different decomposition rules: exact decomposition, key position decomposition, and branch decomposition. Fathoming tests are done by means of a lower bound or by checking if the current job population matches one of the earlier job populations that have already been solved optimally (found-solved). This basic structure is repeated at each node of the search tree.

The proposed algorithm is justified on the basis of four theorems. Theorem 1 (β-Theorem) gives sufficient conditions for optimality. This theorem is proved on the basis of Emmons (1969) and Lawler (1977). The exact decomposition (Theorem 2) is obtained from Emmons' first and second theorems extended via Lawler (1977). The key position decomposition (Theorem 3) and the branch decomposition (Theorem 4) are direct extensions of the decomposition theorems of Chang *et al.* (1995) and also those of Potts and van Wassenhove (1982) via Lawler (1977). The relationship of the proposed theory to the existing theory is summarized in Fig. 1.

The paper is organized as follows. In Section 2, we provide the basic results from the literature that we use in our algorithmic design. In Section 3, we prove the β-optimality. In Section 4, we give an efficient method to construct the β-sequence. In Section 5, we give three decomposition theorems. In Section 6, we give the proposed algorithm. Additionally, we briefly discuss time and storage requirements of the algorithm. In Section 7, we present and discuss the computational results. In Section 8, we present two new heuristics and discuss their computational performance. The paper ends with concluding remarks in Section 9.

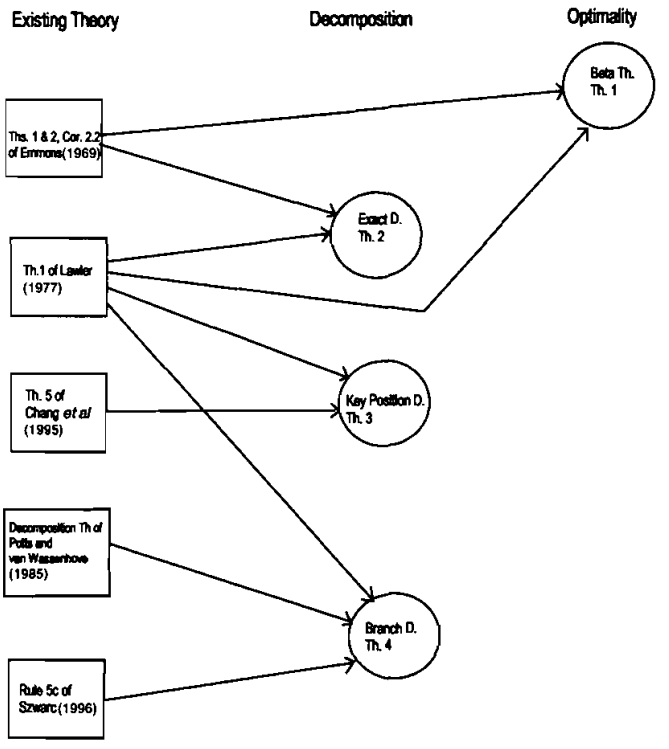

Fig. 1. The relationship of the proposed theory to the existing theory.

2. Basic results

In this section, we present the existing theory that we use in our algorithmic design, namely the first and second theorems of Emmons (1969) and the theorem due to Lawler (1977).

Let $J = \{1, \ldots, n\}$ and let p_j, d_j denote, respectively, the processing time and due date of job j. For any sequence S of job indices, let $C_j(S)$ be the completion time of job j in sequence S. The total tardiness associated with S is $T(S) = \sum_{j \in J} \max(0, C_j(S) - d_j)$. The problem is to find a sequence S^* such that $T(S^*) \leq T(S) \forall S$. For any subset K of J, let $p(K) = \sum_{j \in K} p_j$.

Theorem 1 of Emmons (1969). *For a pair of jobs j and k with $j < k$, if a set B_k of jobs is known to precede job k in some optimal sequence and if $d_j \leq \max\{p(B_k) + p_k, d_k\}$, then there exists an optimal sequence in which all jobs in $B_k \cup \{j\}$ precede job k.*

Theorem 2 of Emmons (1969). *For a pair of jobs j and k with $j < k$, if sets B_k and A_k of jobs are known to precede and succeed job k, respectively, in some optimal sequence, $d_j > \max\{p(B_k) + p_k, d_k\}$, and $d_j + p_j \geq p(J - A_k)$, then there exists an optimal sequence in which all jobs in $A_k \cup \{j\}$ succeed job k.*

We refer to B_j and A_j as the *predecessor* and *successor* sets of job j, respectively, if jobs in B_j are known to precede and jobs in A_j are known to succeed job j in at least one optimal sequence. In addition, we let $E_j = p(B_j) + p_j$, $L_j = p(J - A_j)$ and refer to these as the *earliest* and *latest* completion times of job j, respectively.

The following corollary to Theorem 2 (Emmons, 1969) gives sufficient conditions for optimality of the Earliest Due Date (EDD) sequence.

Corollary 2.2 of Emmons (1969). *The EDD sequence is optimal if it results in waiting times $W_j \leq d_j$ (i.e., $C_j(\text{EDD}) - p_j \leq d_j$) for all jobs j.*

The next theorem states that the due dates can be modified in appropriate ranges without losing optimality.

Theorem 1 of Lawler (1977). *Let S^* be any sequence which is optimal with respect to the given due dates $d_1, \ldots, d_n$ and let $C_j(S^*)$ be the completion time of job j for this sequence. Let d'_j be chosen such that $\min(d_j, C_j(S^*)) \leq d'_j \leq \max(d_j, C_j(S^*))$. Then any sequence S' which is optimal with respect to the due dates $d'_1, \ldots, d'_n$ is also optimal with respect to the due dates $d_1, \ldots, d_n$ (but not conversely).*

3. β-Optimality

Assume the jobs in J are indexed in non-decreasing order of processing times with ties broken by non-decreasing order of due dates. Given an instance of the problem with due dates d_j, suppose we have applied (without creating cycles) Theorems 1 and 2 of Emmons (1969) and obtained predecessor sets B_j. Let $\underline{C}_j = p(B_j) + p_j$ and $\beta_j = \max(d_j, p(B_j) + p_j)\ \forall j$. Clearly $\underline{C}_j$ is a lower bound for the completion time of job j in any optimal sequence that satisfies the precedence relations with respect to the sets B_j. Define the β-sequence to be the sequence such that the jobs are ordered in non-decreasing order of their β_j values with tied ones ordered in increasing order of job indices. Let $J^* = \{j \in J : \exists\ i > j$ such that $\beta_j > \beta_i\}$.

Theorem 1 (β-theorem). *The β-sequence is optimal if $\beta_j \geq p(F_j)\ \forall j \in J^*$ where F_j is the set of jobs that precede job j in the β-sequence.*

Proof. We first prove that the β-sequence is optimal for the problem with due dates $d'_j = \beta_j$, then prove that it is also optimal for the original problem. Let $j \in J$. If $j \in J^*$, then the assumption of the theorem gives $\beta_j \geq p(F_j) = W_j$ (where W_j is the waiting time of job j in the β-sequence). If $j \notin J^*$, then $F_j = \{i : p_i \leq p_j$ and $\beta_i \leq \beta_j\}$ (due to the indexing convention). It follows from Theorem 1 of Emmons (1969) that every job i in F_j is also in B_j. Hence, $\beta_j \geq \underline{C}_j = p(F_j) + p_j > p(F_j) = W_j$. We have shown that $\beta_j \geq W_j\ \forall j \in J$. Corollary 2.2 (Emmons, 1969) implies that the β-sequence is optimal for the problem with due dates β_j.

Observe that the interval $[d_j, \max(d_j, \underline{C}_j)]$ is a subinterval of the interval $[\min(d_j, C_j(S^*)), \max(d_j, C_j(S^*))]$ where S^* is any optimal sequence that satisfies the precedence relations relative to the sets B_j. Since $d'_j = \beta_j$ is in this interval, the β-sequence is optimal for the problem with due dates d_j as a result of Theorem 1 of Lawler (1977). ■

The proof of Theorem 1 provides an important idea which we term the idea of *equivalence*. By observing that the new due dates $d'_j = \beta_j$ are in the intervals $[d_j, \max(d_j, \underline{C}_j)]$, this idea permits every theoretical statement that is valid for the original problem to be re-stated in terms of the data of the new problem while ensuring that the conclusions obtained from such statements remain valid not only for the new problem but also for the original problem. The idea of equivalence is also used in the proofs of Theorems 2, 3, and 4 in the sequel, thereby greatly increasing the utility of the existing theory by appropriately modifying the due dates during the branch and bound algorithm. Even though the root of the equivalence idea lies in Theorem 1 of Lawler (1977), it should be noted that Lawler's theorem alone does not provide a way of creating the modified problem because it requires knowing an optimal sequence. On the other hand, replacing $C_j(S^*)$ with $\underline{C}_j$ circumvents this difficulty in a simple but strong way by eliminating the need to know an optimal sequence.

4. Sequence construction

Theorem 1 gives sufficient conditions for optimality of the β-sequence which is nothing but the EDD sequence with respect to modified due dates obtained through the use of the dominance theorems of Emmons (1969). In this section, we present a method of constructing this sequence in an extremely efficient way. If the β-sequence satisfies the sufficient conditions of Theorem 1, an optimum is at hand. Otherwise, the problem is decomposed to obtain new subproblems each of which goes through the optimality test again with respect to their own β-sequences.

Let $\alpha_j = \max(p_j, d_j)\ \forall j$. Sequence the jobs in non-decreasing order of their α_j values with tied jobs sequenced in increasing order of their indices. The tie breaker defines the resulting sequence uniquely. We call this sequence the *α-sequence*. Note that the well-known heuristic rule MDD

(Modified Due Date) (Baker and Bertrand, 1982) which replaces a job's due date by the current time t plus that job's processing time whenever $d_j < t + p_j$ gives the α_j values if $t = 0$. However, we use the α-sequence as a seed to begin the construction of the β-sequence, rather than as a heuristic rule. Observe that any optimal solution to the problem with due dates α_j is also an optimal solution for the problem with the original due dates d_j. This follows from the fact that $p_j > d_j$ implies job j will be tardy regardless of where it is placed in the sequence. Hence, shifting its due dates to p_j simply changes the tardiness of job j by a constant amount (see also Tansel and Sabuncuoglu (1997) for additional discussion).

For fixed j, partition $J - \{j\}$ into sets $LD_j = \{i \in J : i < j \text{ and } \alpha_i \leq \alpha_j\}$, $LU_j = \{i \in J : i < j \text{ and } \alpha_i > \alpha_j\}, RD_j = \{i \in J : i > j \text{ and } \alpha_i < \alpha_j\}$, and $RU_j = \{i \in J : i > j \text{ and } \alpha_i \geq \alpha_j\}$. We call $LD_j(RD_j)$ the *left-down* (*right-down*) set and $LU_j(RU_j)$ the *left-up* (*right-up*) set of job j. The terminology is motivated by the fact that if we plot the points $(p_1, \alpha_1), \ldots, (p_n, \alpha_n)$ in the plane and pass a horizontal and a vertical line through the fixed point (p_j, α_j), the plane is divided into four quadrants at (p_j, α_j) each of which contains one of the above-mentioned sets. Note that jobs whose points fall on the horizontal or the vertical line through the point (p_j, α_j) belong either to the left-down set or the right-up set of job j depending on their indices. Note also that if $(p_i, \alpha_i) = (p_j, \alpha_j)$, then i is in LD_j if $i < j$ and i is in RU_j if $i > j$. Let $D_j = LD_j \cup RD_j$ and $U_j = LU_j \cup RU_j$. Call D_j the *down-set* and U_j the *up-set* of job j.

The final β-sequence is constructed from the α-sequence by means of successive transformations of due dates. Observe that if we take the initial β-sequence to be the α-sequence, then the set F_j in Theorem 1 is the same as D_j and $J^* = \{j \in J : RD_j \neq \emptyset\}$. We refer to the checking of sufficient conditions $\beta_j \geq p(D_j)\ \forall j \in J^*$ as the β-test. We initially apply this test to the α-sequence. If the test fails, the data transformation is initiated and continued until either the β-test is passed or the transformation is no longer possible. This transformation of the problem into a new form which is more easily solvable than the initial form is one of the features that distinguishes our algorithm from the existing algorithms.

The following property is a direct consequence of Theorem 1 of Emmons (1969):

Property 1. *There is an optimal sequence such that LD_j is a predecessor set of job j and RU_j is a successor set of job j.*

Property 1 gives an easy way to build the predecessor sets of all jobs. The $O(n^2)$ procedure below accomplishes this by successively expanding left-down sets. This procedure defines the β-sequence recursively. We begin the procedure with $\beta_j = \alpha_j\ \forall j$ (or with the most recent β_j values obtained from the procedure RIGHT-EXPAND which is explained later).

LEFT-EXPAND

Step 1. Select any job k which is not selected yet (if none left, stop).

Step 2. Compute $p(LD_k)$.

Step 3. If $p(LD_k) + p_k > \beta_k$, then increase β_k to $p(LD_k) + p_k$, redefine LD_k with respect to the new point $(p_k, \beta_k^{\text{new}})$ and return to Step 2). Otherwise, add job k to the list of selected jobs and return to Step 1).

Example 1 the procedure is illustrated in Fig. 2 for $k = 10$. Initially, $\beta_{10} = \alpha_{10} = 186$ and $LD_{10} = \{1, 5, 7\}$. With $p(LD_{10}) + p_{10} = (10 + 57 + 66) + 82 = 215$, Step 2 updates β_{10} to 215 and LD_{10} now includes job 8 in addition to 1, 5, 7. The vertical arrow that originates at job 10's point in the figure indicates this expansion. The procedure continues with the second, third, and forth expansions as shown by the vertical arrows. The final β_{10} is 544 and LD_{10} is $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$ be a vector of β_j values of jobs at any time during the operation of algorithm LEFT-EXPAND. Redefine $LD_j, LU_j, RU_j, RD_j, D_j, U_j$, and J^* with respect to the points $(p_1, \beta_1), \ldots, (p_n, \beta_n)$.

The β-test can be conducted at any time during the algorithm LEFT-EXPAND. A positive answer is conclusive (optimum at hand) while a negative answer signals the need to further increase the β_j values (especially those that failed in the test). In case of failure of the optimality test, we can use the information available in the right-down sets to further increase the β_j values based on Theorem 2 of Emmons. For a fixed job j, if we choose a

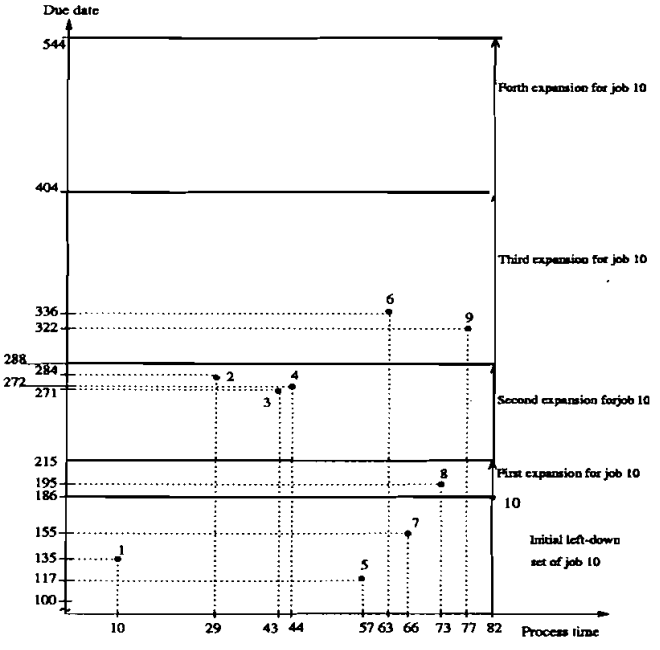

Fig. 2. Illustration of procedure LEFT-EXPAND for job 10.

job k in the most recent right-down set of job j, all conditions of Emmons' second theorem except possibly the last one are automatically satisfied. That is, if we take $B_k = LD_k, A_k = RU_k$, and regard β_j and β_k to be the new due dates, then the only remaining condition that needs to be checked is the inequality $\beta_j + p_j \geq p(J - RU_k)$ (the condition $\beta_j > \max(p(B_k) + p_k, d_k)$ in Theorem 2 of Emmons (1969) is automatically satisfied by all jobs k in RD_j since $\beta_j > \beta_k$ for such jobs due to the definition of RD_j). Whenever this inequality is satisfied, job k qualifies as a predecessor to job j. Such jobs in the right-down set of a given job can be used to further increase the β_j value of that job. This is done by the following $O(n^2)$ procedure, RIGHT-EXPAND.

RIGHT-EXPAND

Step 0. Let $B_j = LD_j$ and $A_j = RU_j \; \forall j$ where LD_j, RU_j are the left-down and right-up sets corresponding to either $\beta_j = \alpha_j \; \forall j$ or the most recent values of $\beta_1, \beta_2, \ldots, \beta_n$ obtained from LEFT-EXPAND. Let $E_j = p(B_j) + p_j$ and $L_j = p(J - A_j) \; \forall j$. Set $j = n + 1$.

Step 1. Set $j \leftarrow j - 1$. If $j = 0$ stop, else go to Step 2.

Step 2. Identify all jobs $k \in RD_j$ (defined by the most recent β_j values) which are not in B_j and which satisfy the inequality $\beta_j + p_j \geq L_k$. Let R be the set of jobs so identified. If R is null, go to Step 1; else go to Step 3.

Step 3. Replace B_j by $B_j \cup R$ and increase E_j by $p(R)$. For each k in R, replace A_k by $A_k \cup \{j\}$ and decrease L_k by p_j. If $E_j^{\text{new}} > \beta_j$, increase β_j to E_j^{new}, and return to Step 2. Otherwise return to Step 1.

Consider the following example:

Example 2. The E_j and L_j values in Table 1 are computed with respect to the LD_j and RD_j sets defined by the α_j values. For example, $E_4 = p(LD_4) + p_4$ where $LD_4 = \{2, 3\}$. For $j = 8, 7$, Step 2 outputs empty RD_j. Continuing with $j = 6$, Step 2 gives $RD_6 = \{7, 8\}$ and $R = \{7\}$.

Table 1. The data for Example 2

j	p_j	d_j	α_j	E_j	L_j
1	57	427	427	57	437
2	80	67	80	80	137
3	91	42	91	171	228
4	94	31	94	265	322
5	103	510	510	425	540
6	104	622	622	529	760
7	115	322	322	380	644
8	116	548	548	656	760

Hence, Step 3 gives $B_6 = LD_6 \cup R = \{1, 2, 3, 4, 5, 7\}$, increases E_6 to 644, updates A_7 to $RU_7 \cup \{6\} = \{8, 6\}$, decreases L_7 to 540, and increases β_6 to 644. Returning to Step 2, with $j = 6$ again, the execution of Steps 2 and 3 yield no change for this job, so the algorithm returns to Step 1 and continues with $j = 5$. Processing job 5 through two cycles of Steps 2 and 3, we obtain $B_5 = \{1, 2, 3, 4, 7\}$, $E_5 = 540, A_7 = \{8, 6, 5\}, L_7 = 437$, and $\beta_5 = 540$. Continuing in like manner with empty RD_j sets for $j = 4, 3, 2$ and with two passes of Steps 2 and 3 for $j = 1$, we obtain the final β-sequence $(2, 3, 4, 7, 1, 5, 6, 8)$ with corresponding β_j values of $(80, 171, 265, 380, 437, 540, 644, 656)$.

We experimented with three strategies for constructing the β-sequence which we call LEFT, LEFT–RIGHT, and FULL. LEFT uses only LEFT-EXPAND while LEFT–RIGHT first uses LEFT-EXPAND, then switches to RIGHT-EXPAND, then executes LEFT-EXPAND one more time and stops. FULL uses LEFT-EXPAND and RIGHT-EXPAND repeatedly, one after the other, until no further updating occurs in the β_j values. The number of nodes of the enumeration tree generated by LEFT exceeds in general those generated by LEFT–RIGHT which usually marginally exceeds those generated by FULL. Based on our computational tests, we note however that, in terms of average CPU times, LEFT generally outperforms both LEFT–RIGHT and FULL except for the most difficult classes of instances. A more detailed discussion of the efficiency issues is given in Section 6.

5. Decomposition

In this section we give three decomposition theorems: exact, key position, and branch decomposition. All of these decompositions try to identify a position in the sequence to split the problem into subproblems. However, there is a major difference between branch decomposition and the other two. Whereas branch decomposition can only identify a set of candidate positions for splitting the problem, the other two find a specific position for decomposition. Branch decomposition leads to different branches from a given node in a branch and bound procedure and to different states in a dynamic programming procedure. Either way, branch decomposition typically leads to an exponential number of subproblems. In contrast, exact and key position decompositions do not lead to any branching. Hence, the major strength of exact and key position decompositions is their ability to continue with a single branch (or a single state in dynamic programming), but their major weakness is that there is no guarantee that such a decomposition may exist. Consequently, the latter two types of decompositions do not have the ability to lead to a branch and bound or dynamic programming algorithm. However, if they are combined with branch decomposition, the outcome is a faster branch and bound or dynamic programming procedure.

In the proposed algorithm, we use exact decomposition whenever the β-test fails. If no job qualifies for exact decomposition, we continue with key position decomposition which is an enhanced version of Theorem 5 of Chang *et al.* (1995). If this decomposition also fails, we continue with branch decomposition which is an enhanced version of the decomposition proposed by Potts and van Wassenhove (1987). The latter two decomposition theorems are enhanced in that the β_j values are used in these theorems instead of the initial due dates, thereby permitting repeated use of these theorems with continually modified data. Note that the original decomposition of Lawler (1977) is also in the category of branch decomposition. In fact, Potts and van Wassenhove's decomposition strengthens Lawler's by eliminating certain alternative decomposition positions.

The exact decomposition theorem looks for a job q that qualifies for decomposition and assigns it to a specific position in the sequence which splits the problem into two subproblems corresponding to the down-set and up-set of job q. The theorem uses the information that is available at the end of the construction of the β-sequence. This information includes the final sets, LD_j, LU_j, RD_j, RU_j, as well as the final values of the latest times $L_1, \ldots, L_n$ (If LEFT is used, the L_j's are taken to be $p(J) - p(RU_j)$ where the RU_j's are the final right-up sets at the end of LEFT-EXPAND). Note that the time complexity of searching for a job q that qualifies for exact decomposition is $O(n^2)$.

Theorem 2 (Exact decomposition). *Let $q \in J$. If*

(i) *either $RD_q = \emptyset$ or $p_q + \beta_q \geq L_i \forall i \in RD_q$ and*
(ii) *either $LU_q = \emptyset$ or $p_j + \beta_j \geq L_q \forall j \in LU_q$,*

then there is an optimal sequence such that all jobs in D_q precede q and all jobs in U_q succeed q. This implies that the problem decomposes in the form (D_q, q, U_q) where the subproblems corresponding to the sets D_q and U_q are solved independently. The jobs in U_q have a ready time of $p(D_q) + p_q$.

Proof. Consider the problem with modified due dates $\beta_1, \ldots, \beta_n$. For this problem it can be shown, via Theorem 2 of Emmons (1969) that (i) in the theorem implies that RD_q is a predecessor set of q while (ii) implies that LU_q is a successor set of q. Property 1 implies that LD_q is also a predecessor set of q and RU_q is also a successor set of q. Hence, D_q is a predecessor set and U_q is a successor set for job q for the problem with due dates $\beta_1, \ldots, \beta_n$, i.e., there exists a sequence $S = (S_1, q, S_2)$ which is optimal with respect to due dates $\beta_1, \ldots, \beta_n$ such that S_1 is a permutation of D_q and S_2 is a permutation of U_q. Consider now the original problem with due dates $d_1, \ldots, d_n$. If we assign $\underline{C}_j = E_j$ where the E_j's are the earliest times obtained at the end of the construction of the β-sequence, then the due dates $\beta_j \in [d_j, \max(d_j, \underline{C}_j)]$. If S^* is an optimal sequence that obeys the so far obtained precedence relations, then $\underline{C}_j \leq C_j(S^*) \forall j$ so that β_j are in the intervals $[\min(d_j, C_j(S^*)), \max(d_j, C_j(S^*))]$. Theorem 1 of Lawler (1977) implies that (S_1, q, S_2) is an optimal sequence with respect to due dates $d_1, \ldots, d_n$. ■

Given the earliest times $E_1, \ldots, E_n$ and the latest times $L_1, \ldots, L_n$ during or at the end of the construction of the β-sequence, if $E_j = L_j$ for some j then the problem clearly decomposes in the form (D_j, j, U_j) which is the kind of block decomposition given by Szwarc and Mukhopadhyay (1996). In this case, it is straightforward to show that conditions (i) and (ii) of Theorem 2 are satisfied. However, Theorem 2 may yield a decomposition even if $E_j < L_j \forall j$. Example 3 illustrates this.

Example 3. The data for this example are given in Table 2. Taking $B_j = LD_j, A_j = RU_j$ and $\beta_j = \max(E_j, d_j)$, we have the results listed in Table 3. Even though $E_j < L_j \forall j$, there is a decomposition at job 7 since $RD_7 = \emptyset$ and $\beta_1 + p_1 = 99 + 1 > L_7 = 59$, $\beta_2 + p_2 = 193 + 3 > L_7$, $\beta_4 + p_4 = 143 + 9 > L_7$, $\beta_6 + p_6 = 132 + 14 > L_7$. Thus, jobs $D_7 = \{3, 5\}$ and $U_7 = \{1, 2, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$ can be sequenced independently.

Next we give the key position decomposition theorem.

Theorem 3 (Key position decomposition). *Let $S_\beta = ([1], \ldots, [n])$ be the β-sequence and assume job n is in position k in S_β. Let $S_\beta(i)$ be the sequence obtained from S_β by moving job n from position k to position i and let $T(S_\beta(i))$ be the total tardiness of $S_\beta(i)$ computed with respect to the due dates $\beta_1, \ldots, \beta_n$. Define the key position index h to be the smallest index such that*

$$T(S_\beta(h)) = \min_{k \leq i \leq n} T(S_\beta(i)).$$

Table 2. The data for Example 3

j	p_j	d_j
1	1	99
2	3	193
3	4	26
4	9	143
5	13	45
6	14	132
7	15	52
8	15	215
9	16	208
10	17	94
11	17	95
12	17	116
13	17	152
14	17	197
15	18	61
16	18	173

If $h < n$, then there is a sequence $S = (S_1, S_2)$ which is optimal for the problem with due dates $\beta_1, \ldots, \beta_n$ as well as for the original problem such that S_1 is a permutation of the jobs that occupy the first h positions in S_β while S_2 is a permutation of the jobs that occupy the last $n - h$ positions in S_β.

Proof. The theorem is an equivalent statement of Theorem 5 of Chang *et al.* (1995) where the d_i's are replaced by the β_i's and the EDD sequence is replaced by the β-sequence. Hence, the optimality of $S = (S_1, S_2)$ for the problem with due dates $\beta_1, \ldots, \beta_n$ is immediate from the theorem of Chang *et al.* (1995). The optimality of $S = (S_1, S_2)$ for the original problem follows from Lawler's theorem using again the fact that $\beta_j \in [d_j, \max(d_j, \underline{C}_j)] \subseteq [\min(d_j, C_j(S^*)), \max(d_j, C_j(S^*))]$ where S^* is an optimal sequence that obeys the obtained precedence relations. ■

With the above theorem, whenever the key position index $h < n$, we can split the current job population into two subproblems, one consisting of the jobs in positions $1, \ldots, h$ in the β-sequence, the other consisting of those in positions $h + 1, \ldots, n$ in the β-sequence. The ready time of the latter set is redefined to be the sum of the processing times of the first set. Note that checking for key position decomposition takes $O(n^2)$ time.

Both Theorems 2 and 3 are based on the β-sequence. The two theorems differ in two respects:

1. The key position decomposition can be carried out only relative to the longest job whereas the exact decomposition can be carried out relative to any qualifying job q.
2. Key position decomposition decomposes the problem into two sets without specifying the position of any of the jobs (including the longest job) whereas the exact decomposition specifies the position of the qualifying job q.

Next, we present the branch decomposition. In what follows, we say job j passes the β-test with *strict inequality* if $\beta_j > p(D_j)$ and with *equality* if $\beta_j = p(D_j)$. If $D_j = \emptyset$, take $p(D_j) = 0$. Theorem 4 below is an equivalent statement of Potts and van Wassenhove (1987) decomposition where the d_j's are replaced by the β_j's. Lawler's theorem implies that the resulting decomposition for the problem with due dates $\beta_1, \ldots, \beta_n$ is also valid for the original problem.

Theorem 4 (Branch decomposition). *Assume job n is in position k in the β-sequence. The problem decomposes with job n in position l for some l satisfying one of the following conditions:*

(i) *$l = k$ and the $(k+1)$st job in the β-sequence passes the β-test with strict inequality.*
(ii) *$l \in \{k+1, \ldots, n-1\}$ and the (l)th job either fails the β-test or passes with equality while the $(l+1)$st job passes the test with strict inequality.*
(iii) *$l = n$ and the (n)th job either fails the test or passes with equality.*

The following example illustrates the Branch decomposition.

Example 4 the data for this example are given in Table 4. The job with the largest processing time is job 16, and it is in the first position of the β-sequence, so $k = 1$. The β-sequence, together with β_j values, down-set totals of the jobs, and passes and fails of the test are given in Table 5. A "$\surd$" stands for a pass with strict inequality, an "=" sign stands for a pass with equality, and an "X" stands

Table 3. The results for Example 3

j	E_j	L_j	β_j
1	1	85	99
2	4	163	193
3	4	8	26
4	14	128	143
5	17	30	45
6	32	128	132
7	32	59	52
8	74	211	215
9	75	211	208
10	49	125	94
11	66	142	95
12	84	159	116
13	124	176	152
14	144	211	197
15	50	193	61
16	160	211	173

Table 4. The data for Example 4

j	p_j	β_j
1	1	456
2	14	432
3	15	246
4	15	385
5	15	490
6	23	325
7	26	519
8	35	418
9	41	390
10	42	440
11	43	417
12	49	475
13	59	506
14	66	432
15	77	292
16	85	241

Table 5. The results for Example 4

β-sequence	β_j	$p(D_j)$	*Test result*
16	241	0	√
3	246	85	√
15	292	100	√
6	325	177	√
4	385	200	√
9	390	215	√
11	417	256	√
8	418	299	√
2	432	334	√
14	432	348	√
10	440	414	√
1	456	456	=
12	475	457	√
5	490	506	X
13	506	521	X
7	519	580	X

for a failure of the test. The test results imply directly that the problem decomposes with job 16 in positions 1, 12, and 16.

Szwarc and Mukhopadhyay (1996) give an additional rule that eliminates some of the positions permitted by the decomposition of Potts and van Wassenhove (1987). This rule is adapted to the β-sequence as follows. The algorithm BETA uses this additional rule in the Branch decomposition.

Rule 5c (Szwarc and Mukhopadhyay (1996)). Let the β-sequence be $[1], \ldots, [n]$ and assume job n is in position k in the β-sequence. Then job n is not in position s, $s \geq k$, if there exists a job i, $i \in \{[k+1], \ldots, [s-1]\}$ such that $p_{[1]} + \cdots + p_{[k-1]} + p_{[k+1]} + \cdots + p_{[s]} + p_{[n]} \leq p_i + d_i$.

Rule 5c additionally eliminates position 12 in Example 4 since $372 + 85 < p_{14} + d_{14} = 498$.

6. The algorithm

The proposed algorithm, which we call BETA, is a depth-first branch and bound algorithm. The algorithm first checks whether the problem on hand is in the list of already solved subproblems (found-solved). If not, the β-sequence is constructed and the β-test applied. If the test is passed, then the β-sequence is optimal. Otherwise, the algorithm looks for a job that qualifies for exact decomposition. If such a job is found, the problem is decomposed and each subproblem is solved via BETA. Otherwise, the algorithm looks for a position h that qualifies for key position decomposition. If such a position is found, decomposition is applied and the subproblems are solved via BETA. Else, the algorithm continues with branch decomposition. During the processing of the branches, BETA is called separately for each branch and two fathoming criteria are applied, one based on found-solved, the other one based on a lower bound. An example that illustrates the algorithm is given in the Appendix.

Of the two fathoming criteria, *found-solved* first checks at each branch if the subproblem under consideration has already been solved during the processing of some other branch. If the answer is yes, just take the solution, otherwise continue with the algorithm in the usual way. Found-solved significantly reduces the number of branches in most instances. As a result, the total CPU time decreases even though additional time is spent per branch for checking the pre-solved problems. The found-solved list contains the most recent k subproblems that have been solved during the algorithm where $k = 1000$ in our implementation.

The second fathoming criterion is based on a lower bound. For the lower bound computation, we use the β-sequence and define a relaxed problem with new due dates $d_j^{\text{new}} = \max\{\beta_j, p(D_j)\}$. With the new enlarged due dates, the β-test will necessarily pass and the resulting β-sequence will be optimal for the modified problem. Since $d_j^{\text{new}} \geq \beta_j \; \forall j$, the total tardiness of the β-sequence with respect to the new due dates gives a lower bound. We proceed with the β-test only if the lower bound fails to fathom that node.

A few words on simplified bookkeeping and computational efficiency of the algorithm are in order. The bookkeeping is greatly simplified by keeping an n-vector which concisely represents all precedence relations. This n-vector contains the job indices in the same order as they appear in the β-sequence (and the associated records $\beta_j, E_j, L_j, p_j, d_j \; \forall j$). From this n-vector, one can easily obtain the sets B_j, A_j and the associated earliest and latest times E_j, L_j. This results in $O(n)$ space for keeping track of precedence relations which lead to $O(n^2)$ space requirement for a depth-first branch and bound tree, whereas existing methods accomplish the same thing in $O(n^3)$ space. As for time requirements, the procedures LEFT and LEFT–RIGHT obtain the β-sequence (and hence the associated precedence relations) in $O(n^2)$ time, whereas the existing methods require $O(n^4)$ time ($O(n^2)$ checking required each time a new relation emerges) for constructing the precedence diagram.

With these considerations, the time and space requirements of the existing methods can easily become prohibitive for $n > 200$ whereas our way of handling the data allows us to solve much larger problems. The time bounds of the different components of the algorithm BETA are given in Table 6. Computational tests indicate that, on the average, about 80% of the total CPU time per branch is spent on the β-sequence computation, about 15% on exact and key position decomposition, and about 5% on branch decomposition, lower bound, and found-solved.

Table 6. Time bounds of the phases of BETA

β-sequence construction	*Optimality check (β-test)*	*Exact decomposition*	*Branch decomposition*	*Key position decomposition*	*Lower bound computation*	*Checking for found-solved*
$O(n^2)$ for LEFT or LEFT–RIGHT $O(n^4)$ for FULL	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n)$	$O(n)$

7. Computational results

The performance of the proposed algorithm is measured on a Sparc Station Classic with a 60 MHz microSPARC 8-CPU and a 384 GB main memory. The code is written in the C language. The data generation scheme initially proposed by Fisher (1976), which has traditionally been used in the literature since then, is used to test the algorithm for different types of instances. These instances of varying degrees of difficulty are generated by means of two factors: tardiness factor, T, and range of due dates, R. For each problem, first the process times are generated from a uniform distribution with parameters (1, 100). Then the due dates are computed from a uniform distribution which depends on $p^* \equiv p(J)$ and on R and T. The due date distribution is uniform over $[p^*(1 - T - R/2), p^*(1 - T + R/2)]$. The values of T and R are selected from the sets $\{0.2, 0.4, 0.6, 0.8\}$ and $\{0.2, 0.4, 0.6, 0.8, 1\}$, respectively. This gives 20 combinations of R, T factors for each problem size. We refer to each (R, T) combination as a problem type. The number of jobs n is chosen from $\{100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 300, 400, 500\}$. We solve 10 different instances for each setting of n, R, T giving a total of 200 instances for each choice of n. The total number of random instances solved is 2800. The time limit to abandon a solution is 80 hours.

A distinctive feature of the proposed algorithm is that it is capable of handling very large populations of jobs whereas this seems to be an impossibility for the existing algorithms in the literature for $n > 200$. Even though the hardest instances could not be solved to optimality for $n = 500$, we still have an 87.5% success rate for this size (i.e., 87.5% of the generated instances with $n = 500$ are solved to optimality).

The detailed results are given in Table 7. Each cell in that table represents the average CPU seconds of 10 instances for each (R, T) pair and n. Of the three versions (LEFT, LEFT–RIGHT, FULL) of the algorithm BETA, the reported numbers except those in parentheses are based on LEFT. The parenthetical numbers are the average CPU seconds for LEFT–RIGHT and are supplied only for those cells where LEFT–RIGHT performed

Table 7. Average CPU seconds for the LEFT version of BETA (numbers in parenthesis are the CPU seconds for LEFT–RIGHT)

R T	*n = 100*	*n = 150*	*n = 200*	*n = 300*	*n = 400*	*n = 500*
0.2 0.2	0.03	0.13	0.70	5.75	31.40	88.50
0.2 0.4	2.17	6.94	125.17	2638.50	29107.4	–
0.2 0.6	4.61	26.04	446.68	33430.90	–	–
0.2 0.8	1.38	15.77	337.99	2269.20	107433.3 (67726.9)	–
0.4 0.2	0.00	0.00	0.00	0.00	0.00	0.00
0.4 0.4	0.35	1.11	14.27	134.60	711.90	2685.80
0.4 0.6	1.96	6.54	92.53	1429.10	6602.90	74286.20 (67317.9)
0.4 0.8	0.08	0.27	0.57	15.20	17.00	42.30
0.6 0.2	0.00	0.00	0.00	0.00	0.00	0.00
0.6 0.4	0.14	0.19	0.60	19.90	24.40	56.40
0.6 0.6	1.43	4.56	34.88	386.30	1562.80	7328.90
0.6 0.8	0.04	0.09	0.26	1.20	1.80	3.90
0.8 0.2	0.00	0.00	0.00	0.00	0.00	0.00
0.8 0.4	0.02	0.04	0.08	0.06	0.20	0.50
0.8 0.6	0.46	0.80	0.96	4.10	22.90	91.90
0.8 0.8	0.03	0.13	0.35	1.03	2.70	5.00
1.0 0.2	0.00	0.00	0.00	0.00	0.00	0.00
1.0 0.4	0.00	0.00	0.00	0.00	0.00	0.00
1.0 0.6	0.11	0.30	0.76	2.05	7.10	10.50
1.0 0.8	0.04	0.11	0.32	1.17	2.70	5.20
Avg. CPU	0.65 sec.	3.16 sec.	52.8 sec.	33.62 min.	–	–
Max. CPU	11.80 sec.	66.72 sec.	21.7 min.	19 (17.1) h.	–	–

considerably faster than LEFT. As remarked earlier, even though FULL gives the best overall performance in terms of the number of branches, its performance in terms of the average CPU seconds is inferior because FULL spends considerably more time per branch to compute the β-sequence than either LEFT–RIGHT or LEFT. The number of branches generated by LEFT–RIGHT is generally quite close to those generated by FULL, but LEFT–RIGHT spends much less time per branch than FULL. Hence, we recommend the use of LEFT–RIGHT for $n = 500$ for the first four most difficult classes of instances corresponding to (R, T) pairs of $(0.2, 0.6), (0.2, 0.8), (0.2, 0.4), (0.4, 0.6)$, where the trade-off between the number of branches and the CPU time per branch works in favor of the reduced number of branches. The same recommendation is valid for categories $(0.2, 0.6), (0.2, 0.8)$ for $n = 400$. In all the remaining cases (i.e., the 16 categories of $n = 500$, the 18 categories of $n = 400$, and all 20 categories of $n \leq 300$), we recommend the use of LEFT as it performs considerably better than LEFT–RIGHT in terms of the average CPU time. Note, however, that the maximum CPU time of LEFT–RIGHT is considerably better for large n ($n \geq 300$) than that of LEFT, e.g., 17.1 hours versus 19 hours for $n = 300$. The general pattern is that it becomes more advantageous to switch-over to LEFT–RIGHT from LEFT as n increases where the switch-over point is earlier for more difficult classes of instances.

As can be seen from Table 7, the average solution time for $n = 100$ is less than 1 sec. The average solution times for $n = 150, 200$, and 300 are 3.16 secs, 52.8 secs, and 33.6 minutes, respectively (the averages for $n = 400$ and 500 are not available because not all instances are solved for those sizes). The worst encountered solution times are 11.8 seconds, 1.11 minutes, 21.7 minutes, 17.1 hours for $n = 100, 150, 200, 300$, respectively (more than 80 hours for sizes 400 and 500). Thus, our algorithm handles problems of size 150 very successfully with an average solution time of about 3 seconds while the maximum solution time for 150 job problems is about 1 minute. 200 job problems are also successfully handled with an average CPU time of less than 1 minute and a maximum CPU time of about 22 minutes.

Table 7 shows that the computation times are highly dependent on the R, T factors. Of the three hard problem types $(0.2, 0.4)$, $(0.2, 0.6)$, and $(0.2, 0.8)$, the most difficult one is the pair $(0.2, 0.6)$. For the blank cells in Table 7, the algorithm obtained suboptimal solutions for $n = 400$, and 500 whose average deviation from optimality is approximately 5%. In all other (R, T) pairs and for all values of n, the algorithm obtained the exact solution for all generated instances.

Observe that five out of 20 (R, T) pairs have a solution time of zero for all n values. Hence, the optimum is immediately found in 25% of all instances. These are the problem types corresponding to (R, T) pairs with $T = 0.2$ with $R = 0.4, 0.6, 0.8, 1$ and $T = 0.4$ with $R = 1$. The optimum is immediately found for these problems because either the β-sequence for the initial job population happens to be the optimum one or a few branches lead to the optimum immediately.

Figure 3 shows the average solution times in CPU seconds as a function of n, ranging from 100 to 200. Observe that, up to $n = 130$, the solution times are less than 2 seconds. At $n = 170$ and 180, the solution times are slightly more than 10 sec, and at $n = 200$, it is about 53 seconds. Hence, the slope increases rather slowly, showing almost a linear trend, from $n = 100$ to 180. After $n = 180$, exponential behavior begins to take over. Even though our algorithm uses reduced memory ($O(n)$ versus $O(n^2)$ in existing algorithms), this exponential behaviour is partly due to the hard disk and memory requirements. However, our computational experiments up to $n = 500$ indicate that, the prohibitive nature of the exponential behavior makes itself felt at much larger n ($n > 300$).

Statistics on the performance of the algorithm in terms of solution times, number of nodes and the depth of the tree are given in Table 8 in the range of $n = 100$ to 200 with increments of 10. A comparison in terms of relative increase in CPU times indicates that the average solution time of the algorithm of Szwarc and Mukhopadhyay (1996) increases 23.69 times as n increases from 100 to 150 while the average CPU time of BETA increases 4.86 times over the same range. Potts and van Wassenhove (1982) do not report any computational times for this range of n, but their reported results indicate that the CPU time increases 35.61 times as n goes from 50 to 100.

Szwarc and Mukhopadhyay (1996) report that, for $n = 150$, the average number of branches and the average depth of the tree are 9005 and 10.7, respectively. For the algorithm BETA and for $n = 150$, the average number of nodes is 811(634) and the average depth of the tree is 8.2(7.6) if LEFT(LEFT–RIGHT) is used which indicates that there is a reduction of about 11(14) times in the average number of branches and a reduction of about 2.5(3) levels in the average depth of the tree depending on which of LEFT or LEFT–RIGHT is used.

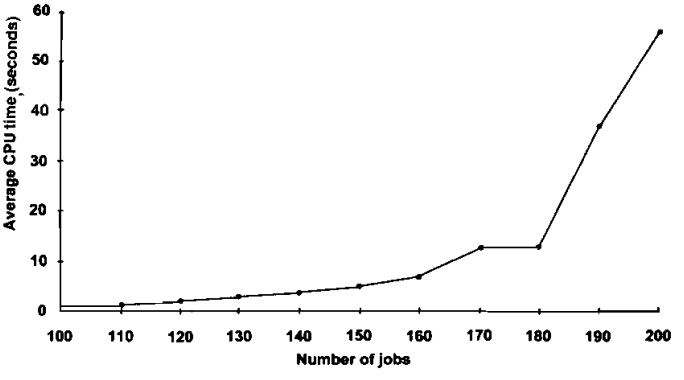

Fig. 3. Solution times of BETA.

Table 8. Average and worst case statistics for n values ranging from 100 to 200

n	*LEFT*		*LEFT*				*LEFT–RIGHT*			
	Solution time		*Number of nodes*		*Depth of tree*		*Number of nodes*		*Depth of tree*	
	Average	*Max*	*Average*	*Max*	*Average*	*Max*	*Average*	*Max*	*Average*	*Max*
100	0.65	11.80	257	4281	6.4	23	229	4001	6.02	22
110	0.76	11.48	332.3	4540	6.5	25	294	3773	6.3	25
120	1.29	19.78	438.4	5915	6.97	25	369.7	4972	6.6	26
130	1.72	32.83	539.9	8560	7.4	27	462.7	6559	7.01	27
140	2.51	40.60	727.5	13 327	7.7	28	588.9	7015	7.3	28
150	3.16	66.72	810.8	15 178	8.2	31	634	11 206	7.60	31
160	5.05	65.65	1092.3	13 105	8.98	29	873.3	7601	8.3	29
170	10.39	252.27	1966.7	37 491	9.3	33	1382.8	21 498	8.69	33
180	10.13	188.50	1971.8	29 236	9.7	35	1547.8	19 954	9.0	35
190	34.10	668.02	4267.2	74 246	10.5	37	2896	40 707	9.9	37
200	52.80	1301.82	6173.9	10 5904	11.5	42	4019	56 139	10.7	42

Szwarc and Mukhopadhyay (1996) also report that, for $n = 150$, the maximum number of nodes and the maximum depth of the tree are 455 233 and 36, respectively. For BETA, the maximum number of nodes is 15 178 and the maximum depth of the tree is 31 for $n = 150$ if LEFT is used. The corresponding numbers are 11 206 and 31 if LEFT–RIGHT is used. The significant difference in the worst case performance may come from the use of different test sets. For this reason, we generated 50 additional random instances for the same n for the most difficult category $(R, T) = (0.2, 0.6)$. This caused the maximum number of branches and the maximum depth of the tree to go up to 49 360 and 38, respectively. The main reason for the reduction in the number of branches is the combined use of various tools (optimality check, exact, key position, and branch decomposition with 5c, β-bound, found-solved) whose effects are magnified due to the continuous reformation of the data at the nodes of the search tree. A particularly effective tool among these seems to be found-solved as we observed in many runs that the same set of subproblems have been encountered in different branches of the tree.

Based on these comparisons, it is reasonable to conclude that BETA performs considerably better than the previous state-of-the-art algorithms in terms of the average CPU times, the average rate of increase in CPU times, the average number of branches, and the average depth of the tree as well as in the maximum number of nodes and the maximum depth of the tree.

8. Beta-based heuristics

In this section, we give two new constructive heuristics based on the algorithm BETA. The first heuristic, HEURBETA, first computes the β-sequence using LEFT–RIGHT. If the optimality check fails, it looks for exact or key position decomposition (Theorems 2 and 3). If no job qualifies for exact decomposition, it decomposes the problem by assigning the longest job to the first (smallest) position which qualifies for branch decomposition (Theorem 4). The resulting subproblems are handled in the same way. HEURBETA is an $O(n^3)$ procedure since, in each pass, at least one job is fixed followed by the β-sequence computation which takes $O(n^2)$. The second heuristic, INITBETA, simply computes the β-sequence using LEFT–RIGHT and takes it as the solution. INITBETA is an $O(n^2)$ procedure.

In Table 9, we give a comparison of these heuristics, in terms of the average and maximum observed deviations from optimality, with three well known heuristics: the PSK heuristic of Panwalkar *et al.* (1993), the ATC (Apparent Tardiness Cost), heuristic of Rachamadugu and Morton, (1981) and the MDD (Modified Due Date),

Table 9. Average (maximum) percent deviations of heuristics from optimality

	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$	*Overall*
HEURBETA	1.6 (11)	1.6 (11)	1.3 (12)	1.2 (12)	0.9 (5)	1.32 (12)
PSK	7.7 (72)	6.1 (∞)	3.3 (16)	5.8 (222)	9.8 (585)	6.54 (∞)
INITBETA	22.4 (88)	21 (86)	19.1 (92)	20.6 (104)	17.4 (93)	20.1 (104)
ATC	45.7 (110)	46.9 (107)	47.3 (96)	47.6 (104)	41.2 (100)	45.7 (110)
MDD	47.8 (110)	48.7 (110)	49.3 (99)	49.2 (104)	42.5 (104)	47.5 (110)

heuristic of Baker and Bertrand (1982). The statistics are based on the same set of 970 test problems that are solved to optimality by the algorithm BETA for $n = 100, 200, 300, 400, 500$. Even though there are various other heuristics in the single machine literature (Wilkerson and Irwin, 1971; Fry *et al.*, 1989; Holsenback and Russel, 1992), we choose the above three heuristics for comparison because PSK is the most successful one reported in the literature for $1//\sum T_i$, while MDD and ATC are the two most frequently used heuristics in the general scheduling literature.

In Table 9, the deviations are computed via the formula

$$\frac{Z_H - Z}{Z} \times 100\%,$$

where Z_H and Z are the total tardiness values of the heuristic and the optimal solutions, respectively. In terms of the average deviation from optimality, HEURBETA shows the best performance with an average deviation of about 1.3% while the closest competitor PSK yields an average deviation of about 6.5%. The superiority of HEURBETA is much more pronounced in terms of maximum deviations. The maximum observed deviation of HEURBETA from optimality is 12% whereas PSK yields a maximum deviation of 585% (barring one instance which yielded ∞% due to the optimal objective value being zero). INITBETA is more primitive than HEURBETA, but it performs reasonably well with an average deviation of about 20% which is poorer than PSK but significantly better than ATC and MDD whose average deviations are close to 50%. In terms of maximum deviations, INITBETA, ATC and MDD perform nearly the same with a maximum deviation of about 100% which is significantly better than that of PSK.

In terms of CPU seconds, the average running time for INITBETA is essentially the same as that of the others while the average running time of HEURBETA is about five or six times the others. Since all of these heuristics run in the order of seconds (e.g., a few seconds for $n = 500$ for HEURBETA), the difference in average CPU seconds seems insignificant relative to the substantial improvement obtained by HEURBETA in terms of the average and maximum deviations from optimality. We note also that all of these heuristics require $O(n)$ storage. In particular, the remarkable success of HEURBETA in terms of the maximum deviation is noteworthy. This heuristic provides a new practical tool for solving very large problems in a few seconds with a maximum expected deviation of only 12%.

9. Concluding remarks

This paper presents a new algorithm for $1//\sum T_i$. The proposed algorithm is based on an integration and enhancement of the existing theory. Its main components consist of a simple optimality check, new decomposition theory, a new lower bound, a check for presolved subproblems, and modified versions of existing theorems that are enhanced through the use of an equivalence concept which permits a repeated modification of the data. The use of these techniques provides substantial savings in bookkeeping and solution time. The combined effect is the ability to solve much larger size problems (e.g., $n = 500$) than previously available in the literature. Extensive computational tests with the new exact algorithm BETA and the heuristic algorithm HEURBETA indicate a significant performance superiority over the existing exact and heuristic algorithms.

References

Baker, K.R. and Bertrand, J.W.M. (1982) A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management*, **3**, 37–42.

Chang, S., Lu, Q., Tang, G. and Yu, W. (1995) On decomposition of the total tardiness problem. *Operations Research Letters*, **17**, 221–229.

Du, J. and Leung, T. (1990) Minimizing total tardiness on one machine is NP-hard. *Operations Research*, **15**, 483–495.

Emmons, H. (1969) One machine sequencing to minimize certain functions of job tardiness. *Operations Research*, **17**, 701–715.

Fisher, M.L. (1976) A dual algorithm for the one machine scheduling problem. *Mathematical Programming*, **11**, 229–251.

Fry, T.D., Vicens, L., Macleod, K. and Fernandez, S. (1989) A heuristic solution procedure to minimize T-bar on a single machine. *Journal of the Operational Research Society*, **40**, 293–297.

Holsenback, J.E. and Russell, R.M. (1992) A heuristic algorithm for sequencing on one machine to minimize total tardiness. *Journal of the Operational Research Society*, **43**, 53–62.

Lawler, E.L. (1977) A 'pseudo polynomial' algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, **1**, 331–342.

Lenstra, J.K., Rinnooy Kan, A.H.G. and Brucker, P. (1977) Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, **1**, 343–362.

McNaughton, R. (1959) Scheduling with deadlines and loss functions. *Management Science*, **6**, 1–12.

Panwalkar, S.S., Smith, M.L. and Koulamas, C.P. (1993) A heuristic for the single machine tardiness problem. *European Journal of Operations Research*, **70**, 304–310.

Picard, J.C. and Queyranne, M. (1978) The time dependent traveling salesman problem and its application to the tardiness problem in one machine scheduling. *Operations Research*, **26**, 86–110.

Potts, C.N. and van Wassenhove, L.N. (1982) A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, **11**, 177–181.

Potts, C.N. and van Wassenhove, L.N. (1985) A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, **33**, 363–377.

Potts, C.N. and van Wassenhove, L.N. (1987) Dynamic programming and decomposition approaches for the single machine total tardiness problem. *European Journal of Operational Research*, **32**, 405–414.

Rachamadugu, R.V. and Morton, T.E. (1981) Myopic heuristics for the single machine weighted tardiness problem. Working Paper 28-81-82, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

Rinnooy Kan, A.H.G., Lageweg, B.J. and Lenstra, J.K. (1975) Minimizing total costs in one machine scheduling. *Operations Research*, **23**, 908–927.

Schrage, L. and Baker, K.R. (1978) Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, **26**, 444–449.

Schwimer, J. (1972) On the n job one machine sequence independent scheduling with tardiness penalties: a branch and bound solution. *Management Science*, **18**, 301–313.

Sen, T.T., Austin, L.M. and Ghandforoush, P. (1983) An algorithm for the single machine sequencing problem to minimize total tardiness. *IIE Transactions*, **15**, 363–366.

Sen, T.T. and Borah, B.N. (1991) On the single machine scheduling problem with tardiness penalties. *Journal of the Operational Research Society*, **42**, 695–702.

Srinivasan, V. (1971) A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. *Naval Research Logistics Quarterly*, **18**, 317–327.

Szwarc, W. and Mukhopadhyay, S.K. (1996) Decomposition of the single machine total tardiness problem. *Operations Research Letters*, **19**, 243–250.

Tansel, B.C. and Sabuncuoglu, I. (1997) New insights on the single machine total tardiness problem. *Journal of the Operational Research Society*, **48**, 82–89.

Wilkerson, J.I. and Irwin, J.D. (1971) An improved method for scheduling independent tasks. *AIIE Transactions*, **3**, 239–245.

Appendix

The example below illustrates the steps of the algorithm BETA.

Suppose we have 10 jobs to schedule corresponding to the data plotted in Fig. A1. According to the steps of the algorithm, we first apply LEFT to obtain the β-sequence (5, 1, 7, 8, 3, 4, 2, 6, 9, 10) shown in Fig. A2.

Jobs 5, 1, 7, 8, 3, 4 pass the β-test, but job 2 fails. Since we find at least one job failing, we cannot conclude that the β-sequence is optimum. Then we look at the exact decomposition and try to decompose if possible. From Fig. A1 we see that $RD_7 = \emptyset$. This means that the first condition of the exact decomposition theorem is satisfied. Since $LU_7 = \{2, 3, 4, 6\}$ the second condition to check is if $p_i + \beta_i \geq L_7 \ \forall i \in LU_7$ and we find that the inequality is valid for all the jobs in LU_7. Thus the problem decomposes with job 7 at position 3 forming two subproblems. One problem has only two jobs $\{1, 5\}$ and the other problem has 7 jobs $\{2, 3, 4, 6, 8, 9, 10\}$.

For the two job case we can easily find the optimum sequence. It happens to be (5, 1). For the second problem we apply the algorithm once more. The β-sequence constructed for this subproblem is (8, 3, 4, 2, 6, 9, 10). We try the β-test first. The test fails so we try exact decomposition. Note that the right-down and left-up sets of job 9 are empty. Hence, the problem decomposes with job 9 at position 6, with job 10 in the up-set and the others in the down-set. After the decomposition with job 9, the down-set $\{2, 3, 4, 6, 8\}$ will be solved. It decomposes with job 6 in the last position. At this stage, jobs 7, 9, 6 are already in fixed positions (these are the jobs that qualified for exact

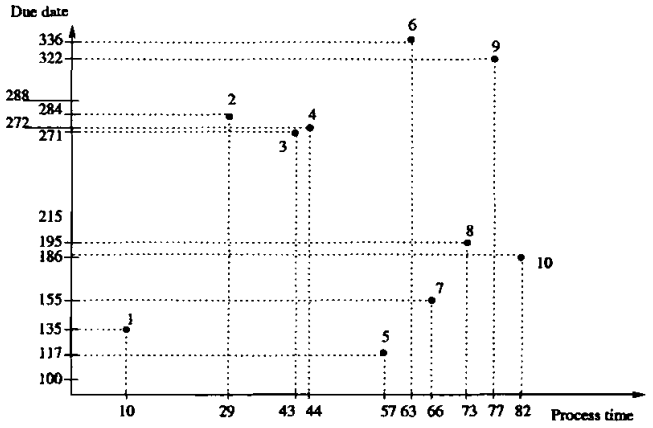

Fig. A1. Data plot of example.

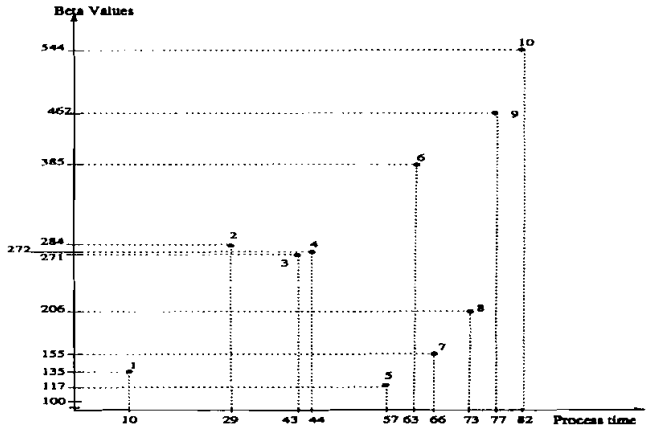

Fig. A2. Graph of β-values.

decomposition). With fixed jobs shown in bold, we have the following partial sequence where jobs in the brackets define independent subproblems.

$$\{5\ 1\}\,\mathbf{7}\,\{2, 3, 4, 8\}\,\mathbf{6}\,\mathbf{9}\,\{10\}.$$

The only non-trivial subproblem is the one corresponding to the job population $\{2, 3, 4, 8\}$ in the middle. Thus the problem is now to schedule four jobs $\{2, 3, 4, 8\}$. For these jobs, the β-sequence is (8, 3, 4, 2) and the β-test, the exact decomposition, and key position decomposition fail. Hence, we apply branch decomposition using the longest job (job 8).

According to Theorem 4 and Rule 5c we find two alternative decompositions for job 8: the position it stays in the β-sequence (from (i) of Theorem 4 since the second job in the β-sequence gets a strict pass in the β-test) and the last position (from (iii) of Theorem 4 since the last job gets a fail in the β-test). So, we continue with two branches. In the first branch we fix job 8 at the first position and then schedule the set $\{2, 3, 4\}$. In this branch, the β-sequence is (3, 2, 4) and this happens to be the optimum sequence for this job population. In the second branch, job 8 is placed in the last position and the other

jobs 2, 3, 4 are to be scheduled before it. The second branch yields an optimum for its subproblem which is no better than the first branch. Hence, the final optimum sequence is (5, 1, 7, 8, 3, 2, 4, 6, 9, 10).

Biographies

Barbaros C. Tansel is the Chairman of the Department of Industrial Engineering at Bilkent University, Ankara, Turkey. Dr. Tansel earned his Ph.D. degree in 1979 from the University of Florida, Department of Industrial and Systems Engineering. Prior to his appointment at Bilkent University in 1991, he was a visiting faculty member at the Georgia Institute of Technology and a faculty member at the University of Southern California. Dr. Tansel's primary research interests are in location theory, combinatorial optimization, and optimization with imprecise data. He has published in various journals including *Operations Research, Management Science, Transportation Science, European Journal of Operational Research, Journal of the Operational Research Society, International Journal of Production Research*, and *Journal of Manufacturing Systems.*

Bahar Y. Kara received her Ph.D. degree in 1999 from Bilkent University, Department of Industrial Engineering. Since then, Dr. Kara has worked as a Post-Doctoral Research Fellow at McGill University, Faculty of Management. Her primary research interests are in hub location, hazardous materials transportation, discrete optimization, and bilevel programming.

Ihsan Sabuncuoglu is an Associate Professor of Industrial Engineering at Bilkent University. He received B.S. and M.S. degrees in Industrial Engineering from the Middle East Technical University and a Ph.D. degree in Industrial Engineering from Wichita State University. Dr. Sabuncuoglu teaches and conducts research in the areas of scheduling, production management, simulation, and manufacturing systems. He has published papers in *IIE Transactions, Decision Sciences, Simulation, International Journal of Production Research, International Journal of Flexible Manufacturing Systems, International Journal of Computer Integrated Manufacturing, European Journal of Operational Research, Production Planning and Control, Journal of the Operational Research Society, Computers and Operations Research, Computers and Industrial Engineering, OMEGA, Journal of Intelligent Manufacturing* and *International Journal of Production Economics.* He is on the Editorial board of *International Journal of Operations and Quantitative Management* and also the *Journal of Operations Management.* He is an associate member of Institute of Industrial Engineering, Institute of Simulation, and Institute of Operations Research and Management Science.

Contributed by the Scheduling Department