New Solution Approaches to the General Single Machine Earliness Tardiness Problem

Hoksung Yau Yunpeng Pan, *member, IEEE* Leyuan Shi, *fellow, IEEE*

***Abstract*— This paper addresses the general single-machine earliness-tardiness problem with distinct release dates, due dates, and unit costs. The aim of this research is to obtain an exact nonpreemptive solution in which machine idle time is allowed. In a hybrid approach, we formulate and then solve the problem using dynamic programming (DP) while incorporating techniques from branch-and-bound (BB). This approach (DP-BB) has been proven to be effective in solving certain types of scheduling problems. We further propose a new adaptation of the approach to a general problem with a nonregular objective function. To address some shortcomings of DP-BB, we also apply a branch-and-bound approach in which partial dynamic programming dominance (BB-PDP) is exploited. Computational experiments were conducted with randomly generated test instances in order to evaluate the effectiveness of the two approaches. The results clearly showed that our new approaches can solve all the instances with up to 40 jobs and most of the instances with 50 jobs, which outperforms those frequently used approaches in scheduling research.**

***Note to Practitioners*— The single-machine scheduling problems with nonregular objective functions are usually harder to solve in comparison to the problems having regular objectives. Most effective algorithms for regular objective problems take advantage of the objective's special feature, which makes it difficult to be applied universally. Sometimes, however, this difficulty can be resolved by changing conventional concepts in the algorithm. This paper addresses a typical nonregular scheduling problem and adopts a technique which has been successfully applied to the regular ones. We extend the traditional concept of "dominance" from one point to piecewise linear so that the technique can be applied to problems with a piecewise linear objective function. Future research can further extend the "linear dominance" to "quadratic dominance" so that this technique can be applied to the single-machine earliness-tardiness problem with a quadratic objective function as well.**

***Index Terms*— single-machine, earliness-tardiness, branch-and-bound, dynamic programming**

I. Introduction

Scheduling problems concerning due dates and earliness-tardiness (E-T) penalties have attracted wide interest in recent years. The importance of these problems is due to a particular dilemma in manufacturing—customers' orders have to be completed in time in order not to lose them, but keeping products too long in inventory will increase production cost. In this situation, schedulers try to complete all the orders as close to their due dates as possible so as to minimize the total cost that might be incurred from either early or late completion. The fundamental model for these scheduling problems contains only one single resource (i.e., a machine) and multiple tasks (i.e., many jobs or orders) with due dates and E-T unit costs.

The single-machine E-T problems have been studied since the 1980's. As indicated by Baker and Scudder [2], these problems can be solved in two stages. The first determines an optimal or near-optimal sequence for all jobs. The second stage computes optimal timing for a given sequence. The optimal timing problem can be easily solved by modeling it as a linear programming (LP) problem (Fry et al. [7]). However, since such optimal timing process is a repetitive task, faster specialized algorithm is more appropriate than solving LP. A better algorithm was proposed by Garey et al. [9], which solved the problem with equal E-T unit costs in polynomial time. Sourd [22] proposed a dynamic programming algorithm to solve the problem with general cost functions. Pan and Shi [17] also presented a procedure, which can be modified to solve the E-T optimal timing problem with arbitrary earliness and tardiness unit costs in polynomial time. Some of these algorithms have been applied in solving single-machine E-T problems as optimal timing procedures.

In comparison to the optimal timing stage, it is much more difficult to find the optimal job sequence. To find such a sequence, the most popular technique that can be used is branch-and-bound. Fujii and Wang [8] proposed a branch-and-bound algorithm for the problem with equal E-T unit costs. Abdul-Razaq and Potts [1] gave a lower bound procedure for the problem with no machine idle time by using dynamic programming state-space relaxation. Li [12] also solved the same problem using branch-and-bound. He developed two multiplier adjustment procedures to solve the Lagrangian subproblems in order to obtain the lower bound. Hoogeveen and Van de Velde [10] tested 5 different lower bound approaches and applied them in a branch-and-bound algorithm with some dominance rules. Note that all of these studies aimed to solve the single-machine E-T problem with some restrictions (e.g., common due date, no idle machine time, equal E-T unit costs). They took advantage of the restricted problems' special structures, thus achieving improved efficiency for the restricted problems. Hence, these approaches usually cannot be applied to such problems without those particular restrictions. Sourd and Kedad-Sidhoum [23] solved the problem with general E-T costs but without release times by using a branch-and-bound approach, where a preemptive lower bound was applied.

Hoksung Yau is a Ph.D. candidate in the ISyE deptartment, University of Wisconsin-Madison, Madison, WI 53706 USA (email: yau@cae.wisc.edu). Dr. Yunpeng Pan is with CombineNet, Inc., Fifteen 27th Street, Pittsburgh, PA 15222 (email: ypan@combinenet.com). Prof. Leyuan Shi is with the ISyE department, University of Wisconsin-Madison, Madison, WI 53706 USA, and also with the Center for Intelligent and Networked Systems, Tsinghua University, Beijing, P. R. China (e-mail: leyuan@engr.wisc.edu).

An early version of this paper has been presented at the IEEE Conference on Automation Science and Engineering 2005.

Because of the complexity of the problem, there have been relatively few publications on the optimal solution of the general single-machine E-T problem. The general problem is NP-hard due to its NP-hard special cases [9]. So, instead of solving the problem for an exact solution, many researchers have turned to searching for near-optimal solutions to large-scale problems. For example, Mazzini and Armentano [13] proposed a constructive heuristic which can handle up to 80 jobs. Yano and Kim [26] developed some heuristic approaches for the problem with earliness and tardiness costs proportional to the processing times. Wan and Yen [25] used a tabu search algorithm to find near-optimal solutions for a more general problem with distinct due windows. These heuristics were quite often employed as upper bound procedures in branch-and-bound algorithms.

The existing body of research indicates that using a technique such as branch-and-bound alone has difficulty to obtain the exact solution to the single-machine E-T problem with arbitrary due dates and E-T unit costs. One solution is to use the combination of dynamic programming with branch-and-bound. This approach has been applied to regular objective scheduling problems. However, to the best of our knowledge, only Bard et al. [3] have applied this method to a nonregular objective scheduling problem. In this paper, we propose new optimization approaches for the general single-machine E-T problem, which combine dynamic programming and branch-and-bound techniques in a graph search. We also apply a transportation problem based lower bound procedure to the nodes in the graph. By exploiting this lower bound procedure and the structure of the E-T problem, we develop some techniques that can help fathom search graph nodes efficiently. The remainder of the paper proceeds as follows: In Section II, the dynamic programming model and branch-and-bound model for the problem are given. Section III describes our new approaches to the problem. Section IV gives some analysis of the proposed approaches and presents computational results. Finally, some concluding remarks are given in Section V.

II. Problem Description and Notation

The general single-machine E-T scheduling problem can be described as follows: Let J be a set of n jobs. Each job j has five attributes: release date r_j, due date d_j, and processing time p_j, unit earliness penalty h_j and unit tardiness penalty g_j. If job j is finished before its due date, inventory costs (such as storage and insurance) are incurred at a rate of h_j per unit time early. On the other hand, if job j is finished after the due date, tardiness costs (such as those due to loss of customer goodwill and/or expediting shipping) accrue at a rate of g_j per unit time tardy. These attributes may vary from job to job. The objective is to find a processing sequence for the jobs in J that minimizes the total cost. If we define C_i as job i's completion time, $E_i = \max\{0, d_i - C_i\}$ and $T_i = \max\{0, C_i - d_i\}$ as earliness and tardiness of job i , respectively, then the general single-machine E-T scheduling problem can be formulated as follows:

$$\min \sum_{i=1}^{n} f_i(C_i)$$

$$\text{s.t.} \quad C_i - C_j \geq p_i \ \ or \ \ C_j - C_i \geq p_j \quad \forall i, j \in J$$
$$C_i \geq r_i + p_i \quad \forall i \in J,$$

where $f_i(C_i) = h_i E_i + g_i T_i$ is a piecewise linear function with two pieces. The main notation used in this paper is listed in Table I.

A. Dynamic Programming Formulation

Dynamic programming is an approach that is often used in solving combinatorial optimization problems. It is basically a complete enumeration scheme that exploits the solutions to all its subproblems. The optimal solution value is recursively computed through the solution values of subproblems in a bottom-up fashion. In other words, dynamic programming first divides the original problem into subproblems and then obtains an optimal solution to the original problem by recursively solving these subproblems. The rationale of the dynamic programming when applied to the scheduling problem lies in the fact that, if a sequence is optimal for the original problem, then any part of the sequence must be optimal for a related subproblem. According to Morin and Marsten [14], scheduling problems can be described by the following dynamic programming recursion:

$$\begin{aligned} f(x', t') = \min\{\omega(x, t, j) + f(x, t) | (x', t'') = \Gamma(x, t, j), \\ t'' \leq t'\}, for (x', t') \in \Omega \setminus \{(x_0, t_0)\} \quad (1) \\ x, x' \in \Theta, \end{aligned}$$

where Θ is a finite set. $t, t', t'' \in R$ usually represent time. $(x', t'), (x', t'')$, $(x, t) \in \Omega$ are the states of subproblems in dynamic programming, where Ω is the nonempty state space with $\Omega \subseteq \Theta \times R$. (x_0, t_0) is the initial state. $j \in D$ is a decision, where D is the set of possible decisions at state (x, t). ω is the incremental cost function of applying decision j at state (x, t). $\Gamma : \Omega \times D \mapsto \Omega$ is a transition mapping that defines the new state by adding decision j at state (x, t). $f(x, t)$ defines the minimum cost at state (x, t) with the given z_0, where $z_0 = f(x_0, t_0)$ is the initial cost at state (x_0, t_0). From the equation above, the solution to the scheduling problem is equivalent to finding $f^* = f(X, t^*)$, where (X, t^*) is the final state, by recursively solving the subproblem $f(x, t)$.

In the general single-machine E-T problem, x stands for a set of jobs and t is the time at which the last of these jobs is completed. The transition mapping is defined as $\Gamma(x, t, j) = (x \cup \{j\}, \max\{t, r_j\} + p_j)$. The incremental cost of adding job j is $\omega(x, t, j) = \omega_j \cdot |\max\{t, r_j\} + p_j - d_j|$, where ω_j is h_j or g_j. As shown in Proposition 1, $f(x, t)$ is a nonincreasing, piecewise linear function in t. Hence, a 5-tuple label $\mathcal{L} = \langle t, z, l, t_c, z_c \rangle$ is introduced to identify every linear piece of the function. In the label, t and t_c represent the earliest and latest possible finish time of the last job, respectively. Accordingly, z and z_c represent the total costs associated with time t and t_c from the fixed jobs (i.e., $z = f(x, t), z_c = f(x, t_c)$). l stands for the slope of the linear piece, which is the rate of cost decrease after time t. Note that the label has four degrees of freedom. But for the sake of exposition, we include all the 5 components in the label. Keep in mind the fact that we always

TABLE I
SUMMARY OF MAIN NOTATION

Category	Notation	Explanation
Dynamic Programming	Θ	the finite set of all combinations of jobs
	x	$\in \Theta$, the job set
	x_k	k-job set
	(x,t)	state
	π	job sequence
	Ω	$\subseteq \Theta \times R$, the nonempty state space
	D	set of all possible decision jobs
	$\omega(x,t,j)$	the incremental cost function of applying decision j at state (x,t)
	$\Gamma(x,t,j)$	the new state by adding decision j at state (x,t)
	z	$= f(x,t)$, the cost of job set x at start time t on label $\mathcal{L}$
	z_c	$= f(x,t_c)$, the cost of job set x at completion time t_c on label $\mathcal{L}$
	l	the slope of label $\mathcal{L}$
	$\mathcal{L}$	$= \langle t, z, l, t_c, z_c \rangle$, a nonincreasing linear segment; label
	T	the time horizon; longest time that guarantees an optimal solution
Branch-and-Bound	$\ell(x,t)$	a lower bound on the costs of x, where the first job starts no earlier than t
	$\mathscr{L}(\mathcal{L})$	a lower bound on f^*, given label $\mathcal{L}$
	$u(x,t)$	an upper bound on the costs of x, where the first job starts no earlier than t
	μ	the global upper bound on f^*

use the absolute value of any label's slope as l, thus we have $l \geq 0$.

Proposition 1. The function $f(x,t)$ *is nonincreasing, piecewise linear in* t.

Proof: From Eq. (1), it is easy to see that $f(x,t)$ is nonincreasing, since the jobs can complete before t. By mathematical induction, we show the piecewise linear property. First, it is verified that $f(x_0,t) = f(\emptyset,t) = 0$ is piecewise linear. Suppose that $f(x_k,t)$ is also a piecewise linear function in t. A certain job set x_{k+1} can be built from job sets $x_k^1, x_k^2, ..., x_k^{n-k}$ with $(n-k)$ distinct jobs individually. By Eq. (1), at any time t', there exists a state (x_k^j, t) such that $f(x_{k+1},t') = \omega(x_k,t,j) + f(x_k,t)$. Since both $\omega(x_k,t,j)$ and $f(x_k,t)$ are piecewise linear, $f(x_{k+1},t')$ is also piecewise linear in t'. Therefore, $f(x,t)$ is nonincreasing piecewise linear in t. ∎

Dynamic Programming Dominance: A label $\mathcal{L}$ can be dominated by other labels of the same node x, if and only if, for any $t \leq t' \leq t_c$, t' is in an existing label of x and $f(x,t') \leq z - l \cdot (t'-t)$. If $\mathcal{L}$ cannot be dominated, it has to be inserted to the node's label list. Figure 1 shows the case in which the label is dominated and Figure 2 shows how a new label is inserted in the associated node. The drawback of dynamic programming for the general single-machine E-T problem is that when the problem size is large, the number of labels will increase rapidly. In practice, this tendency forces the system to allocate a huge memory space to keep track of all the labels' information. As a result, it is impractical to apply dynamic programming to the single-machine E-T problem without modification.

B. Branch-and-Bound

Branch-and-bound approach was also developed to solve discrete and combinational optimization problems. Different than dynamic programming, branch-and-bound solves a problem by dividing the solution space into a number of subregions. This process, called branching, is in turn applied to subregions as well. It leads to an expanding tree where a node corresponds to a subset of solutions. The growth occurs at the frontier or active nodes of the tree; one active node is selected for branching at a time according to the search strategy. Without loss of generality, in a minimization problem, the upper bound stands for the best objective obtained thus far, and for each node, a lower bound is computed to attain the best possible objective value. A node is fathomed if the lower bound is not strictly less than the upper bound. Different problems or problem classes can employ dominance conditions to quickly fathom a node without computing the lower bound. Branch-and-bound terminates with an optimal solution when all actives nodes are fathomed.

Although the dynamic programming and the branch-and-bound approach achieve optimal solutions in different ways, they are actually based on similar data structures — graph for dynamic programming and tree for branch-and-bound. This similarity makes it possible to combine these two approaches to generate a new efficient algorithm for scheduling problems.

III. Hybrid Dynamic Programming and Branch-and-Bound Approaches

There are two ways to combine dynamic programming and branch-and-bound. One is to do branch-and-bound on the dynamic programming search graph; we refer to it as DP-BB. Conversely, we can incorporate dynamic programming dominance in the branch-and-bound framework, which we denote by BB-DP. These two approaches have their own advantages and disadvantages. In practice, they are applied according to the features of the real problems faced.

A. DP-BB

The DP-BB idea was first proposed by Morin and Marsten [14], who applied it to the Traveling Salesman Problem. Barnes and Vanston [4] used it on a single-machine problem with tardiness penalties. Bard et al. [3] also used this approach to solve a single-machine problem with earliness penalties. Our DP-BB algorithm follows the steps in Pan and Shi [18], where it was applied to the total weighted completion time problem with release dates. However, unlike the total weighted completion time problem, the job cost function of

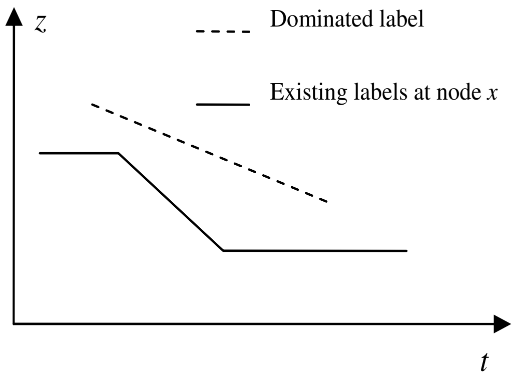


Fig. 1. Dynamic Programming Label Dominance

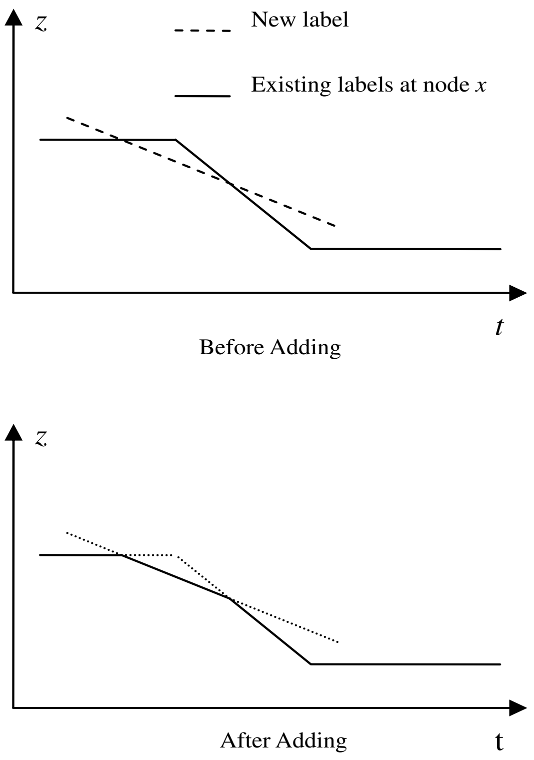


Fig. 2. Add a Label at a Node

E-T problem $f(C_j)$ is not always nondecreasing. This means that ω is not necessarily to be a monotonically nondecreasing function, which violates one of the assumptions in [18]. Hence, the algorithm in [18] cannot be directly used on the E-T problem. Fortunately, the algorithm can still be applied to the E-T problem with the following changes:

1. *Branching*: For a label of node x, attaching an unscheduled job j to x gives a new label in node $x \cup \{j\}$. It is easy to calculate $f(x', t')$ when $\omega(x, t, j)$ is monotonic nondecreasing, where $(x', t') = \Gamma(x, t, j)$. However, if $\omega(x, t, j)$ is not monotonic nondecreasing, $f(x', t')$ cannot be obtained directly. Particularly in the single-machine E-T problem, when there is branching on a label $\langle t, z, l, t_c, z_c\rangle$ of node x, one to three possible new labels may be generated. The number of new labels depends on the difference between job j's due date d_j and t, and the difference between job j's tardiness cost rate g_j and l. Figure 3 shows four cases in which new labels of node x' are generated from those of node x. Note that if $r_j + p_j \geq d_j$, the job j has no earliness penalty indeed and contains the tardiness cost only. The branching process is similar to Case III and Case IV in Figure 3.

2. *Lower Bound*: To apply the branch-and-bound approach in dynamic programming, a strong lower bound for each label is required to curb the size of the search graph. In Section III-B, we will describe how to obtain a strong lower bound for all unscheduled jobs when the latest completion time of scheduled jobs is given. Here, suppose this lower bound function $\ell(\bar{x}, t)$ is known for a particular label $\mathcal{L}$, where $\bar{x} = J \backslash x$ and x is the associated fixed job set of $\mathcal{L}$. If $l = 0$, the lower bound of the whole job set J is $\mathscr{L}(\mathcal{L}) = \ell(\bar{x}, t) + f(x, t)$, where we only need to evaluate the lower bound once. If $l > 0$, then the label is a ranged decreasing linear function in t. Hence, $\mathscr{L}(\mathcal{L}) = \min_{t \leq t' \leq t_c}\{\ell(\bar{x}, t') + f(x, t')\}$. The straightforward way to find $\mathscr{L}(\mathcal{L})$ is to calculate the lower bound for every t' and choose the minimum. However, as will be indicated in Section III-B, the lower bound evaluation is the most time-consuming step in the algorithm. To reduce the number of these evaluations for a single label, we use the following approximation instead of the exact $\mathscr{L}(\mathcal{L})$. Assume that $\ell(\bar{x}, t)$ is a nondecreasing function in t. If we let $\mathscr{L}(\mathcal{L}) = \ell(\bar{x}, t) + f(x, t_c)$, then it is guaranteed that $\mathscr{L}(\mathcal{L}) \leq \min_{t \leq t' \leq t_c}\{\ell(\bar{x}, t') + f(x, t')\}$. Although this approximated lower bound is not as strong as the exhaustive bound, it gives quite good approximation and can be computed much faster in practice, since it entails only one lower bound evaluation for each label.

3. *Dynamic Programming Dominance and Label Updating*: We have discussed this change in Section II. At any time t, we are only interested in the lowest cost of the scheduled jobs. Hence, we compare the new generated label $\mathcal{L}$ with any existing label $\mathcal{L}'$ of the associated node, and keep the one with the lower cost at any common time t of $\mathcal{L}$ and $\mathcal{L}'$. Let μ be the upper bound of J and $u(\bar{x}, t)$ be the upper bound of unscheduled job set $\bar{x}$ starting from time t. The complete algorithm is presented below :

Algorithm DP-BB(μ)

Step 0. *Initialization.* Let $\mu := \max\{\mu, u(J, t_0)\}$. If $\ell(J, t_0) \geq \mu$, STOP. Let $P := \emptyset, Q := \{x_0\}, S := \emptyset$. Node x_0 has single label $\langle t_0, z_0, 0, +\infty, z_0\rangle$.

Step 1. If $Q = \emptyset$, STOP. Let x be the oldest node in Q and $Q := Q \setminus x$. Let $\mathcal{L}^m, m = 1, \ldots, L_x$ be the labels in P_x. Let $m := 0$.

Step 2. Let $m := m + 1$. If $m > L_x$, go to 1.

Step 3. *Branching.* For each available j, $S := S \cup \Gamma(x, t^m, j)$.

Step 4. If S is empty, go to 1. Let (x', t') be the oldest state in S and set $S := S \setminus (x', t')$. Let $\mathcal{L}'$ be the associated label.

Step 5. *DP Dominance.* If node x' already exists in P and for any $\hat{t}$, where $t' \leq \hat{t} \leq t'_c$, there is a label $\mathcal{L}''$ in $P_{x'}$ where $t'' \leq \hat{t} \leq t''_c$ and $z''_{\hat{t}} \leq z'_{\hat{t}}$, then $\mathcal{L}'$ is fathomed. Go to 4.

Step 6. *Upper bounding.* Set $\mu := \min(\mu, z' + u(\bar{x}', t'))$.

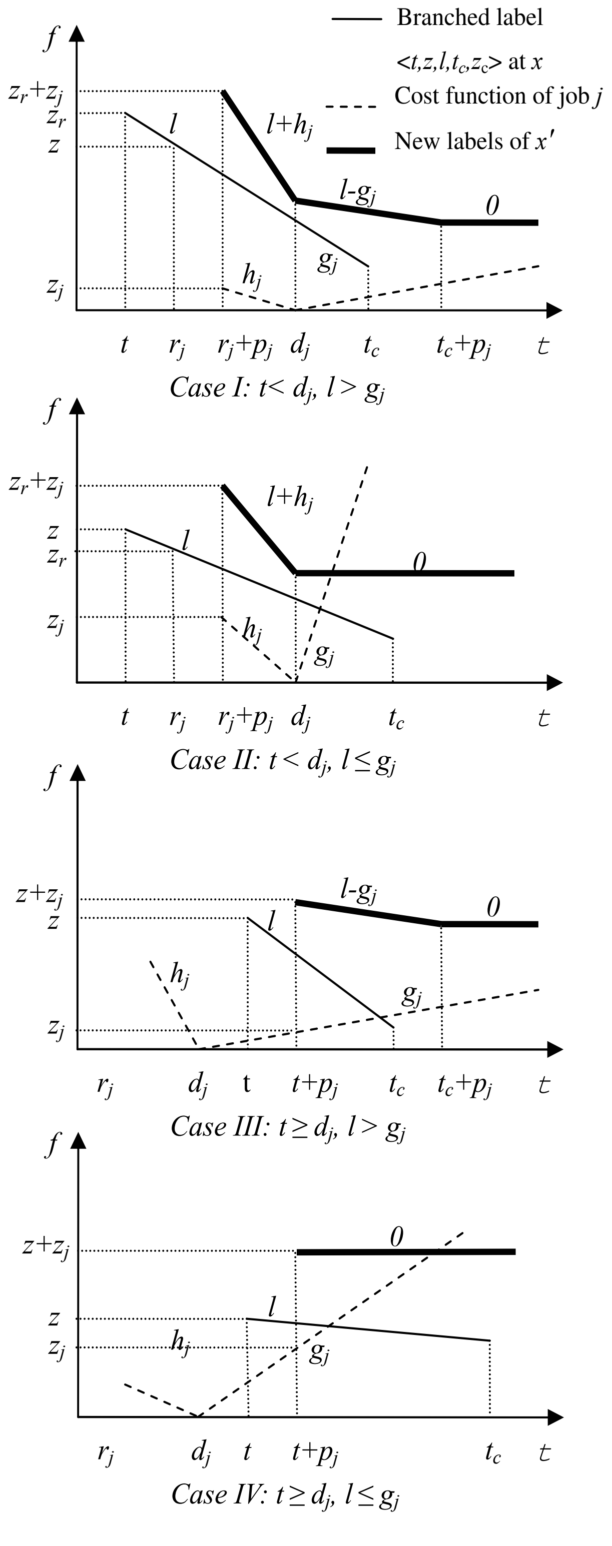


Fig. 3. Branch a Label at Node x

Step 7. *Lower bounding and cut off*. If $l' > 0$ and $\ell(\bar{x}', t') + z'_c \geq \mu$, or $l' = 0$ and $\ell(\bar{x}', t') + z' \geq \mu$, then label $\mathcal{L}'$ is fathomed. Go to 3.

Step 8. *Updating Q and P*. If node x' is not in Q, set $Q := Q \cup \{x'\}$. If node x' is not in P, add x' and label $\mathcal{L}'$ to P. Otherwise, insert new labels $\mathcal{L}'$ in $P_{x'}$. Go to 4.

B. Lower Bound and Upper Bound Procedures

The procedure of lower bound estimation plays an important role in our algorithms. The efficiency of the algorithms relies heavily on a fast procedure that produces strong lower bounds. It is straightforward to model and solve the lower bound problem as an LP. Unfortunately, our computational experience has shown that solving the LP of this problem is time-consuming and results in weak bounds for most of the instances. To overcome this difficulty, Sourd and Kedad-Sidhoum [23] and Sourd [21] proposed to calculate the lower bound for discrete and continuous problems, respectively. Both of these two approaches considered a preemptive lower bound of the original problem. In our discrete case, we decompose the jobs into a number of unit-time operations and assign these operations to discrete time slots. Thus, the lower bound problem is converted to the following *transportation problem*:

$$\min \sum_{i \in J} \sum_{t=1}^{T} c_{it} x_{it} \qquad (2)$$

$$\text{s.t.} \sum_{t=1}^{T} x_{it} = p_i \quad \text{for } i \in J$$

$$\sum_{i \in J \cup \{J_0\}} x_{it} = 1 \quad \text{for } t = 1 \ldots, T$$

$$x_{it} \in \{0, 1\} \quad \text{for all } i \in J \cup \{J_0\} \text{ and } t = 1, \ldots, T$$

where J_0 is a dummy job, c_{it} is the transportation cost of pair (i, t) and T is the time horizon of the schedule, where there is at least an optimal schedule finishing before T. The dual problem of the *transportation problem* can be written as:

$$\max \sum_{i \in J} p_i u_i + \sum_{t=1}^{T} v_t \qquad (3)$$

$$\text{s.t. } u_i + v_t \leq c_{it} \text{ for } i \in J, t = 1, \ldots, T$$

As described in Nemhauser and Wolsey [15], the transportation problem can be solved using the Hungarian method in $O(T^3)$ time. In [23], an adaptation of the Hungarian method to this particular transportation problem is suggested with $O(n^2 T)$ complexity. Bülbül et. al [5] also used the same adaptation with better transportation parameters. The resulting lower bound is stronger than the one provided by the LP relaxation of the disjunctive integer programming formulation.

In our implementation, we set $T = \max_{j \in J}(r_j, d_j) + \sum_{j \in J} p_j$ at the root node and reduce it at the others according to the parent nodes' lower bound assignment and the fixing job's occupation. For the upper bound, we used tabu search to find a good solution at the root node. The solutions found by Tabu search are usually very close to the optimal. The average gap by tabu search at the root node is less than 2%. But tabu search usually takes longer time than other heuristic methods, which makes it impractical to be used on all nodes. Hence, for nodes other than the root, we applied Mazzini and Armentano [13]'s algorithm, which provides a reasonably good upper bound in $O(n \log n)$ time.

C. Techniques for Improving Algorithmic Performance

We also implemented the following techniques in the algorithm, which were intended to improve its computational performance.

1) Branch-and-Bound Dominance: Lower bound evaluation usually accounts for 50–90% of the total computation time of the algorithm, according to our computational results. So, the smaller the number of labels that are needed to calculate the lower bound, the shorter the time that the algorithm needs to solve the problem. Here we apply three different kinds of branch-and-bound dominance to reduce the number of labels:

1. Consider branching on a label $\mathcal{L}$. If job i and job j are both unscheduled, and $t_c+p_j \leq r_i$, $r_j+p_j \leq r_i$, and $d_j \leq r_i$, then job i cannot be scheduled before job j. This is because if we fix job j first at its optimal time and run job i later, the cost is guaranteed to be less than the cost of the schedule in which job i runs before j.

2. In the situation where all the unscheduled jobs will surely be late, these jobs should be scheduled in increasing order of $p_i/\ g_i$ [20].

3. Consider a label $\mathcal{L}$. When $\mathcal{L}$ is branched on, if the next job i that is going to be fixed has release date $r_i \geq t_c$, then it is not necessary to create a branch for job i. Recall that the cost function in the dynamic programming node is nonincreasing piecewise linear. $r_i \geq t_c$ implies that job i has to start at later labels than $\mathcal{L}$, and with a smaller fixed job cost.

2) Partial Dominance: A label that cannot be fathomed by dynamic programming dominance rules and branch-and-bound cutoff will be added to the associated node. However, such a label can still be partially dominated by either dominance rules or bounding. As indicated in Section III-A, $\mathscr{L}(\mathcal{L}) = \ell(\bar{x}, t) + z_c$. If $\ell(\bar{x}, t) + z_c < \mu$, this label cannot be discarded. But if meanwhile $\ell(\bar{x}, t)+z > \mu$, then applying $\ell(\bar{x}, t)$ on every label point gives a segment on which the optimal solution is not possible, under the assumption that $\ell(\bar{x}, t)$ is nondecreasing in t. Specifically, let $t'' = t + [\ell(\bar{x}, t) + z - \mu]/l$ and z'' be the fixed job cost at time t''. Then, for all $t \leq t' \leq t''$, $\ell(\bar{x}, t) + z' = \ell(\bar{x}, t) + z - (t' - t) \cdot l \geq \mu$. Hence, the start time of label $\mathcal{L}$ can be increased from t to t''. The label is thus shortened and in turn, has a greater likelihood of being fathomed. This is because even if a label cannot be fathomed based on its lower bound, its associated shortened label could very well be fathomed by dynamic programming dominance. Figure 4 shows an example of how a label is partially fathomed. The label cannot be fathomed by dynamic programming dominance because the time in segment (a,b) is out of the range at node x. It cannot be discarded by bounding either, since $\ell(\bar{x}, t_a) + z_c < \mu$. However, if we know that $\ell(\bar{x}, t_a) + z_b \geq \mu$, then segment (a,b) can be discarded. On the other hand, segment (b,c) is completely above the labels at the node. This means (b,c) can be discarded by dynamic programming dominance rules as well. Hence, the whole label can be fathomed using dominance rules along with bounding. Another benefit of this partial fathoming is in branching. In Section III, we have indicated that 1–3 new labels may be generated from an existing label $\mathcal{L}$. If the start time t of $\mathcal{L}$ increases and exceeds the due date of a job j, the number of new labels after branching will be reduced by one. (Note that the partial fathoming cannot be applied to the labels with $l = 0$.)

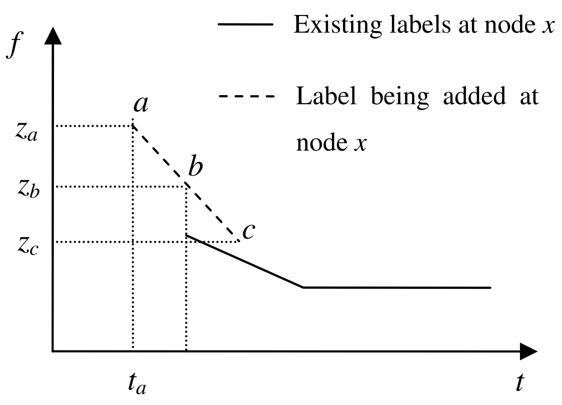


Fig. 4. Partial Fathoming

In our implementation, we still add the original long label at the associated node, and defer partial fathoming until right before the branching step. This is because a long label has a greater chance of dominating other existing labels at the node than a shorter one.

3) Early Termination and Pre-dominance: Besides the branch-and-bound dominance/cutoff used to reduce the number of labels, another effective technique is to speed up the primal-dual procedure of the Hungarian method. Note that the objective value of any feasible solution to the dual problem (3) is a valid lower bound for the primal problem (2). Whenever the objective value of (3) is greater than the difference of the upper bound and the fixed job set cost, the label must have a lower bound that exceeds the upper bound. It is not necessary to continue the procedure in this case, because the label is fathomed at this point. In our implementation, we check the objective value of (3) every time u and v are updated. The complexity of this step is $O(n + T)$. Our computational experience indicates that this extra step is quite helpful.

Another way to improve the performance is to ignore some labels with zero tangent. Suppose label $\mathcal{L}$ is followed by label $\mathcal{L}'$ at node x, and l is positive while l' is equal to 0. Recall that the lower bound of J on $\mathcal{L}$ is $\mathscr{L}(\mathcal{L}) = \ell(\bar{x}, t) + z_c$. If $\ell(\bar{x}, t) + z_c \geq \mu$, then $\mathcal{L}$ is fathomed. $\mathcal{L}'$ can be fathomed as well without calculating the lower bound if $z' = z_c$ (which happens in most cases). This is because $\mathscr{L}(\mathcal{L}') = \ell(\bar{x}', t') + z' \geq \ell(\bar{x}, t) + z_c \geq \mu$.

4) Memory Management: Recall that DP-BB utilizes the breadth-first search scheme, which implies that all potential labels with k fixed jobs in the search queue have to be branched before any label with $k + 1$ fixed jobs can be explored. Theoretically, each label contains the basic segment information $\langle t, z, l, t_c, z_c \rangle$ only. In practice, we solve the transportation problem for each label, and endeavor to pass parents' solutions onto the children in order to speed up the lower bound evaluation. Hence, besides the basic label data, we also record the dual solutions to the transportation problem, which include u and v with space complexity $O(n)$ and $O(T)$, respectively. The required memory size of a problem instance depends on the maximum number of labels that reside in the search queue simultaneously. It can easily exceed the capacity of the system memory for large or difficult instances.

It is probable that dual solutions are similar for two consecutive labels at a node, since they are from the transportation problems with the same fixed jobs and only differ in the start time for unscheduled jobs. Hence, the dual solutions of the previous label can be used for the next or even all the labels that follow. As a result, we only need to record one set of dual solutions for these labels. Although the dual solutions for other labels are usually not exact, this strategy actually helps reduce the memory requirements. In the extreme case, only the first label needs to contain the parents' dual solutions for the whole node. But, because the first label can be so different than the last, this extreme difference might cause the problem that the dual solutions recorded are too inaccurate to offer any useful hint for the label at hand. To achieve a good tradeoff between accuracy and memory requirement, we always keep the dual solutions that have been in the node and only remove the new added label's if necessary. Specifically, let $\langle t, z, l, t_c, z_c\rangle$ be the label that has been added at the node and $\langle t', z', l', t'_c, z'_c\rangle$ be the label that is going to be added. If $t > t'$, record new label's dual solutions; otherwise, discard the dual solutions. When a label without dual solution data is considered for the lower bound, the available dual solutions of the nearest label will be used instead.

D. An Example of DP-BB

To further explain the DP-BB approach, we use the 3-job instance in Table II as an example. Figure 5 shows the DP-BB procedure on the example. In the figure, each node contains a job set and at least one label. The lower bound $\mathscr{L}(\mathcal{L}) = \ell(x,t)+z_c$ is indicated right after each label $\mathcal{L}$ if $\mathcal{L}$ is not fathomed by dynamic programming dominance rules. In this example, $\ell(x,t)$ is obtained by solving the associated transportation problem and z_c is shown in the label. Suppose that we have attained an upper bound of 12 at the root node. There are three nodes that can be obtained by branching on the root node with different jobs. All the labels in these nodes are eliminated by bounding except $\langle 5,5,5,6,0\rangle$ in node $\{1\}$ and $\langle 5,2,2,6,0\rangle$ in node $\{2\}$. Branching on label $\langle 5,5,5,6,0\rangle$ with job 2 gives two new labels $\langle 7,8,2,8,6\rangle$ and $\langle 8,6,0,+\infty,6\rangle$ in node $\{1,2\}$. (Branching with job 3 generates no new labels because $t_c = 6 < r_3 = 8$ (refer to Section III-C.1.3).) Labels $\langle 8,4,1,9,3\rangle$ and $\langle 9,3,0,+\infty,3\rangle$ that are also in the node $\{1,2\}$ are generated from the label $\langle 5,2,2,6,0\rangle$. Note that $\langle 8,6,0,+\infty,6\rangle$ can be dominated by $\langle 8,4,1,9,3\rangle$ and $\langle 9,3,0,+\infty,3\rangle$. Labels $\langle 7,8,2,8,6\rangle$ and $\langle 9,3,0,+\infty,3\rangle$ are fathomed by bounding. Hence, there is only one remaining label, $\langle 8,4,1,9,3\rangle$. Branching on this label with the only remaining job 3 gives the label $\langle 11,12,0,+\infty,12\rangle$ in the final node. We thus obtain the optimal solution to the instance with job sequence $\pi = 2,1,3$ and objective 12.

TABLE II
A 3-JOB INSTANCE

Job	r	p	d	h	g
1	2	3	6	5	1
2	3	2	6	2	3
3	8	3	9	3	4

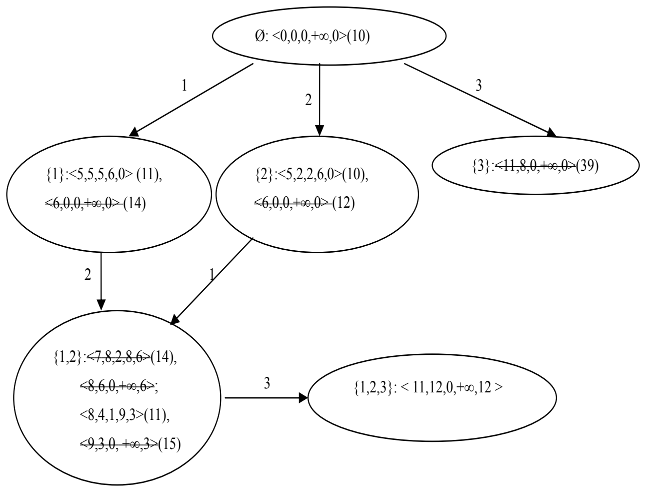


Fig. 5. DP-BB Example

E. BB-PDP

As described above, in the general single-machine E-T problem, DP-BB provides an efficient scheme for taking full advantage of dynamic programming and branch-and-bound approaches. However, the drawback of DP-BB is also apparent: The structure of DP-BB determines that it has to utilize breadth-first search for each label at each node. This search scheme prevents the algorithm from exploring the more in-depth part of the search graph and obtaining high quality solutions early on during the search. Additionally, increasing the number of nodes in the graph can easily cause computer memory overflow, which makes the problem unsolvable. The common solution to this difficulty is to adopt depth-first search, instead of breadth-first search. For this purpose, the problem should be first reformulated in a branch-and-bound framework, in which dynamic programming dominance rules can be applied.

The standard branch-and-bound approach usually contains dominance rules for single-machine scheduling problems. The general dominance rule compares two job sequences of the same job set and discards the one with a higher fixed job cost at any time t. One example of this dominance rule can be found in [20]. However, this dominance rule is weak for the E-T problem because of its nonregular objective function. It is unlikely that one sequence can completely dominate another. Instead of one-to-one comparison, the dynamic programming dominance rule discards a job sequence π by comparing it with multiple job sequences, which dominates π more effectively. Specifically, suppose job sequence π has the same job set with

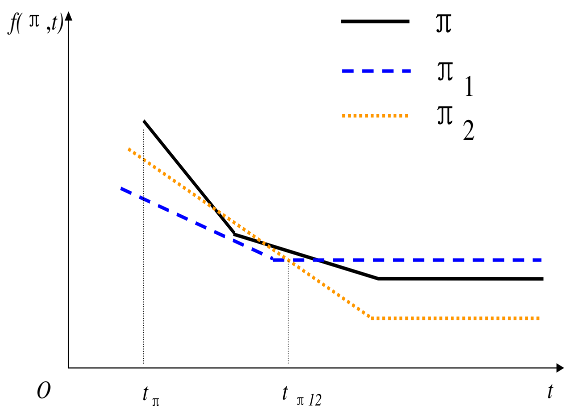


Fig. 6. Dynamic Programming Dominance Rule for Job Sequence π

job sequences $\pi_1, \pi_2, \ldots, \pi_n$. Let $\mathcal{S} = [t_\pi, +\infty)$ (where t_π is the earliest possible finish time of π) and $\mathcal{S}_i$ be the set of times that π_i can dominate π. The dynamic programming dominance rule guarantees that π cannot be part of an optimal sequence as long as $\mathcal{S} \equiv \bigcup_{i=1}^{n} \mathcal{S}_i$. Figure 6 shows an example of using the dynamic programming dominance rule. Sequence π cannot be dominated by either π_1 or π_2 individually. But part $[t_\pi, t_{\pi 12}]$ of π can be dominated by π_1 and $[t_{\pi 12}, +\infty)$ can be dominated by π_2, where $t_{\pi 12}$ is the intersection time point of $f(\pi_1, t)$ and $f(\pi_2, t)$. Therefore, $\mathcal{S} \equiv \mathcal{S}_1 \cup \mathcal{S}_2$ and sequence π is dominated.

We have shown that a sequence π can be dominated by comparing it with multiple other job sequences. Now the question is which job sequences π_i should be chosen to do the comparison. Ideally, all possible sequences of a job set should be considered in order not to miss any potential dominance. However, a job set of size k has $k!$ different sequences, making it impossible to enumerate them all for large k. Hence, we only consider a branch-and-bound algorithm with partial dynamic programming dominance (BB-PDP). We tested two ways to select potential job sequences: 1.) For the job sequence with size k, fixed the first k_1 jobs, and enumerate all the job sequences with these fixed jobs. 2.) From the optimal timing solution for π, enumerate the last k_2 jobs where there is no machine idle time between the jobs. If k_2 is too large to be enumerated, swap every two jobs among the k_2 jobs. Empirical evidence shows that the second sequence selection method usually can find all the valid sequences more quickly than the first one. We therefore use this method in all our experiments.

IV. Computational Experiments

The test instances that we used are similar to those in [13]. They were generated as follows. The processing times p_i were drawn from the discrete uniform distribution U[1,100] and the release dates were from U[0,P], where $P = \sum_{i=1}^{n} p_i$. The unit earliness and tardiness costs were generated from U[0,100]. As suggested in [19], the due dates were generated from U[P(max[0,(1-T-R/2)]), P(1-T+R/2)], where T is the tardiness factor and R is the due date range. We chose T=0.2, 0.5, and 0.8 and R=0.4, 0.7, and 1.0. There are a total of 9 different scenarios, which is shown in Table III. We tested 5 instances for each scenario.

There are two main factors that determine the difficulty of a problem. One is the problem size. The other is the parameters T and R, which represent the due date window's position and its range, respectively. Generally, the narrower the due date window, the more easily the problem can be solved (a common due date is the extreme case). The position of the due date window also impacts the difficulty of the problem. The scenarios with due date windows close to time zero are considered to be relatively easy. Notice that the release date is generated in U[0,P]. If the due date range also starts at time zero, it is very possible that some jobs have later release dates than due dates. These jobs have only tardiness cost terms. The problem becomes a weighted tardiness problem if all the jobs have release dates later than due dates, which is much easier to solve. In our research, we consider the problems in scenarios 1 to 4 as *hard* problems and the others as *easy* problems with the same problem size.

A. DP-BB vs. BB-PDP

We use a 30-job *hard* problem instance to analyze the approach of DP-BB. Figure 7 shows the number of nodes generated at each level for the instance using DP-BB. Figure 8 depicts the relationship between the number of labels at each level for the same instance. We can see from the figures that the numbers of nodes and labels reach their peaks near the first 1/3 of the jobs. For the pure dynamic programming approach, the theoretic peak appears in the middle when half of the jobs are fixed. But, because of the dynamic programming dominance and bounding, the actual peak shifts to an earlier time. Furthermore, the numbers fluctuate sharply around the peak, but in other areas, they tend to increase or decrease gradually. This is a common phenomenon in the application of DP-BB to the general single-machine E-T problem. There are about 4.63 labels at each node on average throughout all the levels. In Figure 8, we also give the number of labels that are needed to do the lower bound evaluation based on transportation problem. Obviously, the smaller the number, the more quickly the problem can be solved. The figure shows that strong dominance rules are needed to fathom more labels before computing the lower bound.

The performance of the algorithm is determined by the quality of the lower bounds and upper bounds for the labels. The approach that we used for lower bounding is much better than the LP relaxation of the disjunctive integer programming formulation. For a particular instance of a 30-job problem, the optimal objective value turns out to be 420391. The LP relaxation takes 0.05 seconds to get the root node lower bound 328276 with a gap of 22%. The transportation problem based lower bound is 405811 with a gap of 3.5%, and takes merely 0.08 seconds. We can see that the transportation problem based procedure gives a much stronger lower bound while using almost the same amount of time as LP relaxation. To show the importance of upper bound in this problem, the same instance was tested. The root node upper bound is 436819 with an optimality gap of 3.9%. The total computation time for this

TABLE III

9 DIFFERENT SCENARIOS FOR THE TEST PROBLEMS

Scenario	1	2	3	4	5	6	7	8	9
T	0.2	0.2	0.2	0.5	0.5	0.5	0.8	0.8	0.8
R	0.4	0.7	1.0	0.4	0.7	1.0	0.4	0.7	1.0
Interval	[0.6P, P]	[0.45P, 1.15P]	[0.3P, 1.3P]	[0.3P, 0.7P]	[0.15P, 0.85P]	[0, P]	[0, 0.4P]	[0, 0.55P]	[0, 0.7P]

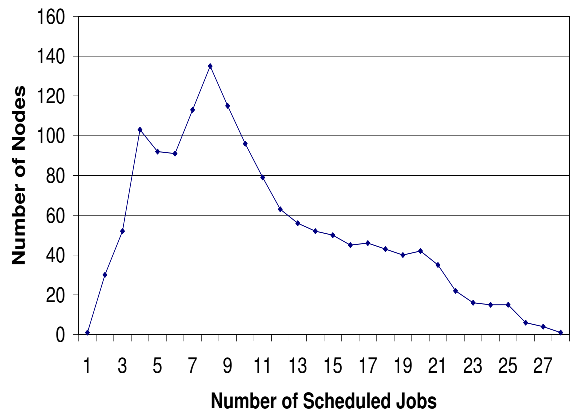


Fig. 7. Number of nodes visited for a 30-job instance

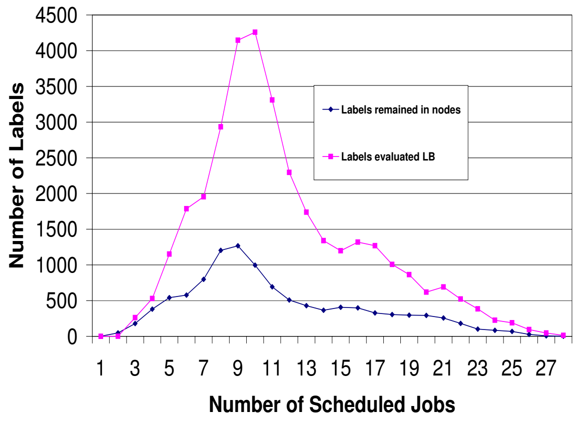


Fig. 8. Number of labels visited for a 30-job instance

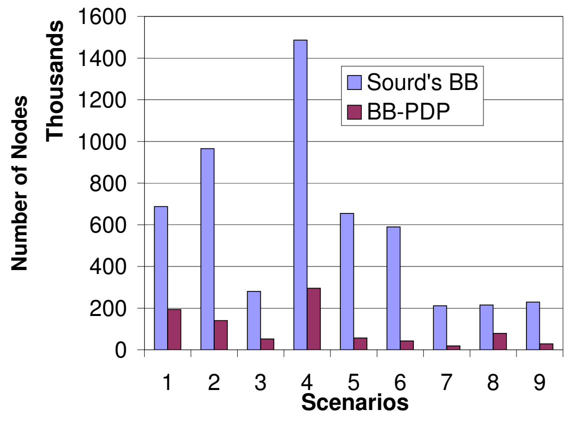


Fig. 9. Number of Nodes by BB-PDP and Sourd's BB on 30-job instances

instance is 11 seconds. But, if we set the initial upper bound to 420391 directly, the total computation time is reduced to 7 seconds. Even a small reduction on the initial upper bound gap can have a significant impact on the overall performance.

In Section III-E, we have shown that BB-PDP can fathom a job sequence more completely than the standard branch-and-bound. The improvement can be seen from Figure 9, which shows the number of nodes generated by [23]'s branch-and-bound and BB-PDP on all the 30-job problem instances. The difference between BB-PDP and Sourd and Kedad-Sidhoum's algorithm lies in the way in which dominance is applied. With the partial dynamic programming dominance, the number of nodes can be reduced by 80% on average from the traditional dominance rule. The associated CPU time is reduced by around 80% as well.

B. Numerical Results

We implemented our algorithm in C++ and compiled it in Visual Studio .NET 2003. All the instances were tested on a Pentium IV 2.8GHz computer. We compared DP-BB and BB-PDP with the time-indexed integer programming formulation (TIF) of the scheduling problem (Formulation E in [6]). The TIF was solved with AMPL/CPLEX9.1. In BB-PDP approach, we used the same upper bound and lower bound procedures as DP-BB. The numerical results are shown in Tables IV and V.

To compare the efficiency between DP-BB, BB-PDP and TIF on the 20-job problem group, we set the maximal time limit to 1 minute. The execution time for the 30, 40, and 50-job problem group was limited to 1, 1, and 2 hours, respectively. The instances that required longer time than the limit are not included in the average CPU time and label/node statistics. We can see that over 50% of the 20, 30, and 40 jobs instances could not be solved with TIF, while all the instances were completely solved with DP-BB or BB-PDP. The fundamental methodology that CPLEX uses is also branch-and-bound. But, CPLEX adds cuts at the root node so that in many problem instances, it can usually achieve very strong root node lower bound and consequently does not generate many nodes, as indicated in Table V. Although this strategy is very successful in some instances, it fails in many others, since computing lower bounds with added cuts consumes much of the CPU resources. In the case that a poor lower bound is obtained after a long period of time spent on lower bound computation, CPLEX is usually not able to find the optimal solution within the set limited time. This reduces its overall performance and causes many instances unsolvable in the set time. Compared to BB-PDP, DP-BB's advantage is obvious in both the execution time and the number of labels/nodes generated. Although these two approaches share the same lower bound procedure, DP-

TABLE IV
COMPARISON BETWEEN DP-BB, BB-PDP AND TIF ON CPU TIME

		# of UI**			Avg CPU Time (sec)			Max CPU Time (sec)		
Size	Sc*	DP-BB	BB-PDP	TIF	DP-BB	BB-PDP	TIF	DP-BB	BB-PDP	TIF
20	1	0	0	0	0.9	1.8	13.20	1.2	2.5	21.58
	2	0	0	5	1.7	5.9	N/A	4.2	17.4	60
	3	0	0	3	0.9	4.8	15.18	1.6	11.1	60
	4	0	0	2	0.6	1.7	14.34	0.7	2.1	60
	5	0	0	4	0.8	3.3	11.86	1.5	7.2	60
	6	0	0	3	0.5	1.6	27.35	0.6	2.6	60
	7	0	0	3	0.4	1.3	10.65	0.5	1.8	60
	8	0	0	2	0.4	2.0	10.61	0.8	6.5	60
	9	0	0	2	0.8	2.6	34.60	1.5	5.4	60
30	1	0	0	4	46.7	238	3262.4	110	481.2	3600
	2	0	0	4	24.2	180.9	2423.9	46.7	276.5	3600
	3	0	0	3	9.7	76.8	1518.9	11.5	162.1	3600
	4	0	0	5	32.2	312.8	N/A	67.8	659.3	3600
	5	0	0	2	3.7	74.4	1018.8	4.9	262.3	3600
	6	0	0	1	3.5	50.1	458.6	4.9	135.5	3600
	7	0	0	3	1.6	18.1	10.4	1.8	34.1	3600
	8	0	0	3	2.9	74.4	3145.9	7.1	302.9	3600
	9	0	0	0	3.3	33.8	1318.6	4.4	57.6	3345.8
40	1	0	2	2	497.6	780.7	1221.2	925.3	3600	7200
	2	0	1	4	571.8	2177.5	3726.1	1092.3	3600	7200
	3	0	2	5	299.9	1892.1	N/A	1154.4	3600	7200
	4	0	1	3	323.8	1403.3	2193.7	738.9	3600	7200
	5	0	2	3	78.8	905	1632.1	184.6	3600	7200
	6	0	0	2	60.8	1190.3	4004.7	122.8	3384.6	7200
	7	0	1	4	9.6	348.8	7083.3	12.4	3600	7200
	8	0	1	1	53.3	552.9	3532.4	225.4	3600	7200
	9	0	0	4	13.3	422	855.4	19.8	760.9	7200
50	1	5	4	4	N/A	2731.4	1030.2	7200	7200	7200
	2	3	5	5	2950.4	N/A	N/A	7200	7200	7200
	3	0	5	5	1776.5	N/A	N/A	4043.1	7200	7200
	4	1	4	4	2479.3	914.03	5550	7200	7200	7200
	5	1	3	5	443.7	4308	N/A	7200	7200	7200
	6	0	4	5	1954.7	977.7	N/A	5972.6	7200	7200
	7	0	3	4	89.9	1197.7	991.3	160.8	7200	7200
	8	0	2	5	113	3912.3	N/A	260.5	7200	7200
	9	1	4	5	873.4	5556.3	N/A	7200	7200	7200

* Sc = Scenario
** UI = Unsolved Instances

BB eliminates more nodes. We can also see from Table IV that BB-PDP is worse than DP-BB on most of the problem groups. However, as a backup for DP-BB, BB-PDP can often reach high-quality solutions at earlier stages and consume much less memory. For the 50-job problem group, DP-BB fails in obtaining optimal solutions for all the 5 instances in Scenario 1, due to its running out of memory every time. It does not achieve better solutions than root node upper bounds in these instances. By contrast, BB-PDP can not only solve one of the instances, but also obtain much better final solutions than DP-BB on other instances.

In order to find the largest problem size that DP-BB and BB-PDP can deal with, we tested the 7th scenario of 60 and 70 jobs problem groups. The numerical results for each instance are presented in Table VI. Note that the longest execution time is set to 1 hour and 2 hours for 60 and 70-job instances, respectively. From the table, we can see that DP-BB solves all the 60-job instances while BB-PDP and CPLEX fail. For 70-job instances, only DP-BB can solve some of the instances. The major difficulty that prevents DP-BB from solving the other instances is its excessive memory requirements. The results also show that in case that the problem is unsolvable, BB-PDP can usually get a better solution.

V. CONCLUSIONS AND FUTURE RESEARCH

This paper applies the hybrid approach of dynamic programming and branch-and-bound technique to the general single-machine E-T problem. Since it does not have those particular restrictions, such as common due date and no machine idle time, it can be more generally applied as compared to those approaches that work on restricted problems only. DP-BB can optimally solve up to 40-job problems with any difficulty level and most of the 50-job problems. Furthermore, because the exhaustive searching structure of DP-BB is based on the E-T cost function only, existing techniques for the restricted problems, such as efficient and strong lower bound procedures, can be integrated into the DP-BB approach to generate a highly efficient algorithm for those problems.

BB-PDP is proposed to resolve the memory overflow problem that occurs frequently in DP-BB. This approach allows us to use the depth-first search scheme. It can obtain high-quality solutions during early stages of the search, and has a modest memory requirement. Although BB-PDP is worse than DP-BB in most instances, it actually performs much better than the traditional solver on integer programming model. Besides DP-BB and BB-PDP, [18] also proposes a dichotomy approach that they refer to as Di-DP-BB. Future research

TABLE V
COMPARISON BETWEEN DP-BB, BB-PDP, AND TIF ON LABELS/NODES

		Avg Label/Nodes			Max Labels/Nodes		
Size	Sc	DP-BB	BB-PDP	TIF	DP-BB	BB-PDP	TIF
20	1	318	2558.8	0	490	3690	0
	2	1153.2	9911.2	N/A	3446	30848	0
	3	491	8004.8	15.5	977	18420	225
	4	286.4	2698	26	380	3217	78
	5	563.4	6207.4	0	1210	13300	91
	6	141	2305.2	122	211	4145	244
	7	155	2157.8	0	265	3183	2
	8	204.8	3819.8	0	676	14747	47
	9	514.8	4453.8	10.3	1250	10032	31
30	1	8973	193359	8661	23454	469388	8661
	2	6069.6	140165	3495	9913	176580	6974
	3	2834.6	51985.4	4149	4326	93690	10801
	4	9634.4	295663	N/A	21016	626640	25915
	5	1210	56663.6	4414	1890	181138	24744
	6	1007.6	42471.6	1966	1475	102746	5802
	7	384.6	18950.4	0	711	38217	48707
	8	1175	79025.2	17137.5	3217	322863	29779
	9	995.8	28810.6	2791	1261	41110	6408
40	1	41429.2	340817	380.7	80105	1857882	5909
	2	65671.6	1053740	5228	118102	1991339	5228
	3	31083.2	1051150	N/A	101040	1876092	11502
	4	48613	782085	991.5	95319	2567452	8040
	5	15516.6	554583	2496	28359	2046053	9998
	6	9763	673097	7630.7	24603	2150205	20565
	7	2462	213232	40610	3757	2237794	40610
	8	12031.4	324841	12065	51076	2066980	25800
	9	3001.8	239599	809	4860	430975	13225
50	1	N/A	902500	14	95981	2417919	4355
	2	187086	N/A	N/A	278231	3098244	2526
	3	96791	N/A	N/A	201411	2608494	7711
	4	126100	405007	8151	219786	2833099	8151
	5	48126.3	1547928.5	N/A	194863	3234758	7250
	6	111184	304773	N/A	319521	5560753	8661
	7	16231.4	671404	779	29952	5646919	17247
	8	18018.6	2850661.6	N/A	29695	6647053	14983
	9	65549.8	3600166	N/A	350824	4195557	10728

TABLE VI
COMPARISON BETWEEN DP-BB, BB-PDP AND TIF ON THE 7TH SCENARIO OF 60 AND 70 JOBS

		Best Objective Gap (%)			CPU Time (sec)			# of Labels/Nodes		
Size	Instance	DP-BB	BB-PDP	TIF	DP-BB	BB-PDP	TIF	DP-BB	BB-PDP	TIF
60	1	0	0	0.49	294.6	3600	3600	29154	719877	439
	2	0	0.11	0.11	1643.5	3600	3600	105530	624556	3571
	3	0	0.17	0.01	1386.6	3600	3600	147928	836097	5188
	4	0	0.07	0	992.2	3600	3600	108391	708664	437208
	5	0	0.21	0.66	3509.3	3600	3600	274540	1544047	117539
70	1	1.89	1.78	1.3	7200	7200	7200	129528	2409944	3232
	2	0	1.68	N/A	3389	7200	7200	242553	1978946	30
	3	1.45	1.69	1.88	7200	7200	7200	190611	1628109	682
	4	0	1.44	2.19	4257	7200	7200	281537	951961	65
	5	1.74	1.57	8.24	7200	7200	†572	223514	1975809	0

†: CPLEX aborted due to run-time failure.

could include applying this dichotomy approach to large-scale single-machine E-T scheduling problems.

The efficiency of both DP-BB and BB-PDP relies on the fast lower bound procedure for each label or node. The lower bound procedure often takes up 50–90% of the overall computation time. In the paper, we point out that there are many more labels discarded than kept in the nodes after bounding. This lessens the overall performance of the approach because evaluating lower bound for useless labels wastes the CPU time. This problem requires other strong dominance rules. In addition, we are also trying to improve the lower bound procedure for the labels. The transportation problem based lower bound currently used is appropriate in DP-BB. But, its $O(n^2T)$ complexity prevents DP-BB from solving larger sized problem. New improved branch-and-bound algorithms have recently been proposed by Kedad-Sidhoum et al. [11] and Sourd and Kedad-Sidhoum [24], who exploited a new Lagrange relaxation lower bound and applied it on the parallel machine problem. They were also able to solve 40-job problems and most 50-job problems. It is quite likely that this new bound will help improve our results, should it be incorporated in our algorithms. By the same token, it would be interesting to see how well the stronger version of the transportation problem lower bound proposed in [16] will perform on E-T problems under our proposed framework.

Acknowledgments

This research was supported in part by NSF grants DMI-0100220, DMI-0217924, and DMI-0431227, by AFOSR grant FA9550-04-1-0179, and by John Deere Horicon Works.

References

[1] T. S. Abdul-Razaq and C. N. Potts, "Dynamic programming state-space relaxation for single machine scheduling," *Journal of the Operational Research Society*, vol. 39, pp. 141–152, 1988.

[2] K. R. Baker and G. D. Scudder, "Sequencing with earliness and tardiness penalties," *Operations Research*, vol. 38, pp. 22–36, 1990.

[3] J. F. Bard, K. Venkatraman, and T. A. Feo, "Single machine scheduling with flow time and earliness penalties," *Journal of Global Optimization*, vol. 3, no. 3, pp. 289–309, 1993.

[4] J. W. Barnes and L. K. Vanston, "Scheduling jobs with linear delay penalties and sequence dependent setup costs," *Operations Research*, vol. 29, no. 1, pp. 146–160, 1981.

[5] K. Bülbül, P. Kaminsky, and C. Yano, "Preemption in single machine earliness/tardiness scheduling." *Journal of Scheduling*, 2005, revision under review.

[6] M. E. Dyer and L. A. Wolsey, "Formulating the single machine sequencing problem with release dates as a mixed integer program," *Discrete Applied Mathematics*, vol. 26, pp. 255–270, 1990.

[7] T. D. Fry, R. Armstrong, and J. H. Blackstone, "Minimizing weighted absolute deviation in single machine scheduling," *IIE Transactions*, vol. 19, no. 4, pp. 445–450, 1987.

[8] Fujii and Wang, "Optimal single-machine scheduling for minimizing the sum of earliness and tardiness penalties," *Journal of the Operationa Researchl Society of Japan*, vol. 31, pp. 105–120, 1988.

[9] M. R. Garey, R. Tarjan, and G. Wilfong, "One-processor scheduling with symmetric earliness and tardiness penalties," *Mathematics of Operations Research*, vol. 13, pp. 330–348, 1988.

[10] J. A. Hoogeveen and S. L. van de Velde, "A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 402–412, 1996.

[11] S. Kedad-Sidhoum, Y. R. Solis, and F. Sourd, "Lower bound for the earliness-tardiness scheduling problem on parallel machines with distinct due dates," *Proceedings of the 9th International Conference on Project Management and Scheduling*, pp. 210–213, 2004.

[12] G. Li, "Single machine earliness and tardiness scheduling," *European Journal of Operational Research*, vol. 96, no. 3, pp. 546–558, 1997.

[13] R. Mazzini and V. A. Armentano, "A heuristic for single machine scheduling with early and tardy costs," *European Journal of Operational Research*, vol. 128, pp. 129–146, 2001.

[14] T. L. Morin and R. E. Marsten, "Branch-and-bound strategies for dynamic programming," *Operations Research*, vol. 24, no. 4, pp. 611–627, 1976.

[15] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. NY: John Wiley, 1988.

[16] Y. Pan and L. Shi, "On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems," *Mathematical Programming, Series A*, 2006, forthcoming.

[17] ——, "Dual constrained single machine sequencing to minimize total weighted completion time," *IEEE Transactions on Automation Science & Engineering*, vol. 2, no. 4, pp. 344–357, 2005.

[18] ——, "New hybrid optimization algorithms for machine scheduling problems," *IEEE Transactions on Automation Science & Engineering*, 2006, forthcoming.

[19] C. N. Potts and L. N. van Wassenhove, "A decomposition algorithm for the single machine total tardiness problem," *Operations Research Letters*, vol. 1, no. 5, pp. 177–181, 1982.

[20] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, pp. 59–66, 1956.

[21] F. Sourd, "The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions," *INFORMS Journal on Computing*, vol. 16, no. 2, pp. 198–208, Spring 2004.

[22] ——, "Optimal timing of a sequence of tasks with general completion costs," *European Journal of Operations Research*, pp. 82–96, 2005.

[23] F. Sourd and S. Kedad-Sidhoum, "The one-machine problem with earliness and tardiness penalties," *Journal of Scheduling*, vol. 6, pp. 533–549, 2003.

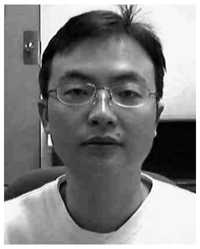
[24] ——, "An efficient algorithm for the earliness-tardiness scheduling problem," 2005, working paper.

[25] G. H. Wan and B.-C. Yen, "Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties," *European Journal of Operational Research*, vol. 142, pp. 271–281, 2002.

[26] C. A. Yano and Y. Kim, "Algorithms for a class of single-machine weighted tardiness and earliness problems," *European Journal of Operational Research*, vol. 52, no. 167-178, 1991.

PLACE PHOTO HERE

Hoksung Yau is a Senior Operations Research Analyst with Harman-Becker Automotive Systems, based in Martinsville, IN. He is also a Ph.D. candidate in Industrial and Systems Engineering at University of Wisconsin-Madison, where he also received his M.S. (2004). He received his B.E. in Computer Science and Technology from Tsighua University, Beijing, China (1999). His research interests include modeling and analyzing business ecosystems, developing techniques for large-scale production planning and scheduling problems and designing optimization algorithms for company supply chain management. He is a member of INFORMS.



Yunpeng Pan is an Algorithms and Formulations Architect with CombinetNet, Inc., based in Pittsburgh, PA. He received a Ph.D. in Industrial Engineering (2003) and an M.S. in Computer Sciences (2001) from University of Wisconsin-Madison, an M.S. in Operations Research from University of Delaware (1998), and a B.S. in Computational Mathematics from Nanjing University, China (1995). His research interests are concerned with developing industrial strength techniques and methods for solving difficult Mixed-Integer Programming problems that arise in E-Commerce, Combinatorial Auctions, Procurement (Reverse) Auctions, and Healthcare Informatics. His work appears in Mathematical Programming, Operations Research Letters, IEEE Trans. on Automation Science and Engineering, European Journal of Operational Research, and Journal of Systems Science and Systems Engineering. Dr. Pan is a member of IEEE, INFORMS, and the Mathematical Programming Society.

Leyuan Shi is a Professor with Department of Industrial and Systems Engineering at University of Wisconsin-Madison. She received her Ph.D. in Applied Mathematics from Harvard University in 1992, her M.S. in Engineering from Harvard University in 1990, her M.S. in Applied Mathematics from Tsinghua University in 1985, and her B.S. in Mathematics from Nanjing Normal University in 1982. Dr. Shi has been involved in undergraduate and graduate teaching, as well as research and professional service. Dr. Shi's research is devoted to the theory and applications of large-scale optimization algorithms, discrete event simulation and modeling and analysis of discrete dynamic systems. She has published many papers in these areas. Her work has appeared in Discrete Event Dynamic Systems, Operations Research, Management Science, IEEE Trans., and, IIE Trans. She is currently a member of the editorial board for Journal of Manufacturing & Service Operations Management, and is an Associate Editor of Journal of Discrete Event Dynamic Systems. Dr. Shi is a fellow of IEEE and a member of INFORMS.