Hybrid backward and forward dynamic programming based Lagrangian relaxation for single machine scheduling

Lixin Tang\textsuperscript{a,*}, Hua Xuan\textsuperscript{a}, Jiyin Liu\textsuperscript{b}

\textsuperscript{a}Department of Systems Engineering, Northeastern University, Shenyang, China

\textsuperscript{b}Business School, Loughborough University, Leicestershire, LE11 3TU, UK

Available online 29 November 2005

Abstract

In this paper we consider the single machine scheduling problem with precedence constraints to minimize the total weighted tardiness of jobs. This problem is known to be strongly NP-hard. A solution methodology based on Lagrangian relaxation is developed to solve it. In this approach, a hybrid backward and forward dynamic programming algorithm is designed for the Lagrangian relaxed problem, which can deal with jobs with multiple immediate predecessors or successors as long as the underlying precedence graph representing the precedence relations between jobs does not contain cycles. All precedence constraints are considered in this dynamic programming algorithm, unlike the previous work using forward dynamic programming where some precedence relations were ignored. Computational experiments are carried out to analyze the performance of our algorithm with respect to different problem sizes and to compare the algorithm with a forward dynamic programming based Lagrangian relaxation method. The results show that the proposed algorithm leads to faster convergence and better Lagrangian lower bounds.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Hybrid backward and forward dynamic programming; Lagrangian relaxation; Single machine scheduling; Weighted tardiness; Precedence constraint

1. Introduction

The single machine environment often arises in many manufacturing industries although it is simple. For example, in the steel-making plant of Baosteel Complex, China, molten steel from converter furnaces, whose basic unit is a charge, continuously solidifies into slabs on a single continuous caster. A group of charges using the same intermediate ladle on the continuous caster is called a cast. The charges in the same cast are sequenced so that their slab widths are in descending order. It results in a single machine problem with precedence constraints. Further examples occur in the flowshop and jobshop scheduling. A single bottleneck in seamless steel tube production, which can be viewed as a flowshop, may give rise to a single machine model. Similarly, many jobshop scheduling problems are often decomposed into subproblems that deal with single machines. Therefore, the research on single machine scheduling not only provides insights into the single machine environment but also provides a basis for heuristics for more complex machine environments.
The total weighted tardiness scheduling problem on a single machine is an important generalization of the single machine total tardiness problem. From a practical point of view, the problem of finishing a job before its acceptable delivery date is significant for schedulers. If a job cannot be finished by its fixed due date it will be tardy and a penalty cost will be incurred for each unit of the job’s tardiness. These penalties are represented by job weights that may be different in order to reflect the relative importance and urgency of the jobs. Using the standard notation in Graham et al. [1], the single-machine scheduling problem with the total weighted tardiness criterion studied in this paper can be denoted as \(1|\text{prec}|\sum w_j T_j\), where \(\text{prec}\) denotes that one or more jobs may have to be completed before another job is allowed to start its processing. Lawler [2] and Lenstra et al. [3] show that \(1|\sum w_j T_j\) is NP-hard in the strong sense. Therefore, the more complicated scheduling problem under our consideration is also strongly NP-hard.

Many researchers have worked on the \(1|\sum w_j T_j\) problem and have experimented with many different approaches including exact algorithms and heuristic approaches. Abdul-Razaq et al. [4] gave a survey of algorithms for this problem, especially dynamic programming and branch and bound algorithms, and compared different methods on test problems with up to 50 jobs. Potts and Van Wassenhove [5] introduced a branch and bound algorithm combined with Lagrangian relaxation (LR). For this problem many researchers have also focused on the development of dispatching rules. For example, Holsenback et al. [6] made use of due date modification in a new heuristic and showed that the algorithm performed well especially when 40% or more of the jobs could be finished before the corresponding due dates. Volgenant and Teerhuis [7] exploited a priority rule to improve four construction heuristics and tested on problems up to 80 jobs. Kanet and Li [8] introduced a new rule, referred to as weighted modified due date (WMDD) rule, and tested it against other rules developed for weighted tardiness. However, existing research shows a lack of discussion of LR on the single machine weighted tardiness problem.

In this paper we explore the performance of an approach combining LR with a hybrid backward and forward dynamic programming (HBFDP) algorithm and examine the effects of problem attributes such as the number of jobs on its performance. The rest of this paper is organized as follows. Section 2 is devoted to formulating the single machine weighted tardiness problem. Section 3 presents the LR framework. Details of the HBFDP algorithm for solving the Lagrangian relaxed problems are described in Section 4. Section 5 reports the computational experiments comparing two different methods for solving the relaxed problems. This is followed by some concluding remarks in Section 6.

2. Mathematical formulation

2.1. Problem description

Assume that the entire planning horizon is divided into many smaller intervals with identical lengths. A typical method to represent discrete events is to use uniform time discretization and to assume that events only happen at the boundaries of those time intervals. The single machine total weighted tardiness problem considered in this paper can be stated as follows. \(n\) jobs are to be processed without interruption on a single machine. There are some precedence requirements for processing the jobs, but no cycles appear in the graph representing these precedence relations even when viewed as an undirected graph. Fig. 1 shows an example of a cycle which is formed by the relations among jobs 3, 4, 5 and 6 if the directed precedence graph is viewed as an undirected one. Note that feasible schedules can be found satisfying all the precedence relations in Fig. 1. However, there will be no feasible schedule for the problem if the precedence graph contains directed cycles. In the problem studied in this paper, we assume there exist no directed or undirected cycles in the job precedence graph, like the example graphs in Figs. 2 (a) and 3 (a). In such a precedence graph, a job may have multiple immediate predecessors or successors. For example, in the precedence graph in Fig. 2(a), job 8 has three immediate predecessors 5, 6 and 7 and two immediate successors 9 and 10. Note that the term ‘immediate’ is used here to clearly represent the explicit job precedence relations in the precedence graph but not necessarily to indicate the relation in the processing sequence. In fact, only one of the three jobs 5, 6 and 7 will be processed immediately before job 8 on the single machine though job 8 can only be started when jobs 5, 6 and 7 have all been completed.

The machine can handle at most one job at a time. Each job \(j (j = 1, 2, \ldots, n)\) requires a processing time \(p_j\), and has a positive weight \(w_j\) and a due date \(d_j\), all of which are assumed to be integers. Job processing is assumed to be nonpreemptive so that a contiguous block of time of length \(p_j\) is needed to process job \(j\). All jobs are considered available for processing at (the beginning of) time one. The planning horizon \(K\) is set to \(\sum_{j=1}^{n} p_j\) because the processing
Fig. 1. A precedence graph with an undirected cycle.

Fig. 2. A connected precedence graph $G_p$ and its transformed tree structure $G'_p$.

Fig. 3. An unconnected precedence graph $G_p$ and its transformed tree structure $G'_p$. 
is on a single machine and only non-delay schedules need to be considered due to the fact that total weighted tardiness is a regular measure. The objective is to find a schedule, that minimizes the total weighted tardiness of jobs, by computing job completion times \( \{C_j\} \).

2.2. The model

Parameters:

- \( J \): the set of all jobs, \( J = \{1, 2, \ldots, n\} \), where \( n \) is the total number of jobs
- \( G_p \): precedence graph without cycles formed by precedence relations between jobs
- \( K \): total number of time units in the planning horizon
- \( p_j \): processing time of job \( j \)
- \( w_j \): weight of job \( j \)
- \( d_j \): due date of job \( j \)
- \( C_E^j \): the earliest possible completion time of job \( j \)
- \( C_L^j \): the latest possible completion time of job \( j \)

Decision variables:

\[
\delta_{jk} = \begin{cases} 
1 & \text{if job } j \text{ is processed in time unit } k, \\
0 & \text{otherwise} 
\end{cases}, \quad j \in J; \quad k = 1, 2, \ldots, K.
\]

\( C_j \): completion time of job \( j \), \( j \in J \). It is a time point (\( C_j = k \) means that the job completes at the end of time unit \( k \)).

\( T_j \): tardiness of job \( j \), equal to \( \max\{C_j - d_j, 0\} \)

Using \( G_p \), the earliest possible completion time \( C_E^j = \sum_{h \in A_j} p_h + p_j \) and the latest possible completion time \( C_L^j = \sum_{h \in J \setminus B_j} p_h \) can be computed, where \( A_j \) denotes a set of jobs preceding job \( j \) and \( B_j \) a set of jobs succeeding job \( j \). They are used to define the possible value range of the variable \( C_j \) in the model so as to reduce the search space. Note that an arc \((i, j)\) in \( G_p \) implies that job \( i \) is an immediate predecessor of job \( j \).

With the help of the above symbols, we can present a model formulation for the 1\textit{prec} \( \sum w_j T_j \) problem that was first introduced in [4]. However, in the model, we give a detailed description of machine occupying constraints and therefore make it become more compact:

\[
\text{Minimize } P, \quad \text{with } P = \sum_{j=1}^{n} w_j T_j \quad (1)
\]

subject to

\[
T_j \geq C_j - d_j, \quad j = 1, \ldots, n. \quad (2)
\]

\[
T_j \geq 0, \quad j = 1, \ldots, n. \quad (3)
\]

\[
C_i + p_j \leq C_j, \quad \text{for each arc } (i, j) \text{ of } G_p. \quad (4)
\]

\[
\sum_{k=1}^{K} \delta_{jk} = p_j, \quad j = 1, \ldots, n. \quad (5)
\]

\[
k \delta_{jk} \leq C_j, \quad j = 1, \ldots, n; \quad k = 1, \ldots, K. \quad (6)
\]

\[
C_j - p_j + 1 \leq k + K(1 - \delta_{jk}), \quad j = 1, \ldots, n; \quad k = 1, \ldots, K. \quad (7)
\]
\[
\sum_{j=1}^{n} \delta_{jk} = 1, \quad k = 1, \ldots, K. \tag{8}
\]
\[
\delta_{jk} \in \{0, 1\}, \quad j = 1, \ldots, n; \quad k = 1, \ldots, K. \tag{9}
\]
\[
C_j \in \{C_j^E, \ldots, C_j^L\}, \quad j = 1, \ldots, n. \tag{10}
\]

In the above formulation the objective (1) is to minimize the sum of weighted tardiness of jobs. Constraint sets (2) and (3) define the tardiness of jobs. Constraint sets (4) ensure the precedence relation between jobs connected by an arc in the precedence relationship graph. Constraint sets (5) state the processing time requirement. Constraint sets (6) and (7) define the time interval for which the machine is occupied by a job. Constraint sets (8) require that the machine can handle at most one job at a time. Finally constraint sets (9) and (10) define the variable ranges.

3. Lagrangian relaxation algorithm

LR is one of the efficient methods for solving integer programming problems. It introduces Lagrangian multipliers to relax coupling constraints to the objective function, therefore leading to a relaxed version of the primal problem. For a given set of multipliers the relaxed problem is minimized in order to compute decision variables. A heuristic approach is then used to convert the infeasible solution obtained from the relaxed problem into a feasible solution. The optimal Lagrangian multipliers are found by solving the Lagrangian dual problem using a subgradient algorithm. A detailed description of the general LR method can be found in [9]. The following describes the specific framework of our LR algorithm.

3.1. Lagrangian relaxation

Based on the above formulation, machine capacity constraints (8) are relaxed to the objective function by using Lagrangian multipliers \(u_k\). The resulting LR problem is given by:

\[
L = \min_{\{C_j\}} \left\{ \sum_{j=1}^{n} w_j T_j + \sum_{k=1}^{K} u_k \left( \sum_{j=1}^{n} \delta_{jk} - 1 \right) \right\}
\]

\[
= \min_{\{C_j\}} \left\{ \sum_{j=1}^{n} w_j T_j + \sum_{k=C_j-p_j+1}^{C_j} \sum_{j=1}^{n} u_k - \sum_{k=1}^{K} u_k \right\}
\]

\[
= \min_{\{C_j\}} \left\{ \sum_{j=1}^{n} \left( w_j T_j + \sum_{k=C_j-p_j+1}^{C_j} u_k \right) - \sum_{k=1}^{K} u_k \right\}
\]

\[
= \min_{\{C_j\}} \left\{ \sum_{j=1}^{n} L_j(C_j) - \sum_{k=1}^{K} u_k \right\} \tag{11}
\]

subject to constraints (2)–(7) and (10), where

\[
L_j(C_j) = w_j T_j + \sum_{k=C_j-p_j+1}^{C_j} u_k. \tag{12}
\]

Then the Lagrangian dual problem is

\[
\max_{\{u_k\}} L, \quad \text{with } L \equiv \left\{ - \sum_{k=1}^{K} u_k + \min_{\{C_j\}} \sum_{j=1}^{n} L_j(C_j) \right\} \tag{13}
\]

subject to constraints (2)–(7) and (10).
Since our paper focuses primarily on solving Lagrangian relaxed problem, we only briefly describe the methods for updating the multipliers and for constructing a feasible schedule in the following part of this section. In the next section the details of the HBFDP for solving the relaxed problem will be given.

3.2. Updating Lagrangian multipliers and constructing feasible solutions

In our algorithm, the subgradient optimization method is used to update the multipliers \{u_k\} at each iteration given by

\[ u^{s+1} = u^s + \alpha^s g(u^s), \]

where \( u \) is a vector of multipliers \{u_k\}, \( s \) the iteration index and \( g(u) \) the subgradient of \( L \) equal to \( \left( \sum_{j=1}^{n} \delta_{jk} - 1 \right) \) with respect to \( u_k \). \( \alpha^s \) is the step size at the \( s \)th iteration defined as

\[ \alpha^s = \lambda \frac{L^* - L^s}{\|g(u^s)\|^2}, \quad 0 < \lambda < 2, \]

where \( L^* \) is the optimal solution estimated by the best feasible objective value found so far and \( L^s \) is the value of \( L \) at the \( s \)th iteration. The parameter \( \lambda \) is initially set to a value greater than 1 and is multiplied by a factor if the value of \( L^s \) remains approximately the same over several consecutive iterations.

The procedure terminates when the relative duality gap is close to zero or the number of iterations reaches a limit. Since the solution of the relaxed problem at each iteration in the dual problem solution process always corresponds to an infeasible schedule, a heuristic procedure similar to that in [10] is performed to adjust the infeasible solution to construct a feasible schedule. In this heuristic, jobs are processed on the machine, one after another in ascending order of their completion times in the relaxed solution.

4. Hybrid backward and forward dynamic programming for solving the dual problem

It is well known that forward dynamic programming (FDP) can solve deterministic job problems with assembly but it cannot cope with problems where a job has multiple immediate successors. Contrariwise it is also true concerning backward dynamic programming (BDP). To overcome the above difficulties, we design a HBFDP algorithm for solving the relaxed problem of the \( 1|\text{prec}|\sum w_j T_j \) problem. The method takes into account the advantages of FDP and BDP and was first proposed in Zhang et al. [11] for providing high-level planning support for factories with multiple coordinating cells. During the process of algorithm execution, the stages in DP do not necessarily correspond to a progression of time period in a monotone fashion in either a backward or forward direction, but one mixing the two directions as long as the underlying precedence graph does not contain cycles.

It is necessary to transform \( G_p \) to form a tree in the undirected sense before the method is implemented. The transformation starts from finding a longest directed path in \( G_p \). This path is then placed from left to right and all other branches linking to the path are flipped to the right. As illustrated in Fig. 2, jobs 2 and 3 are flipped to the right-hand side of job 4, and jobs 5 and 7 to the right-hand side of job 8. If \( G_p \) is not connected, e.g., the graph in Fig. 3(a), we do the above for each connected subgraph of \( G_p \). Finally a dummy job 0 with a due date 0 is added with arcs (0, j) to those jobs j that have no other jobs to its left in the graph, resulting in a tree structure \( G_p' \). Figs. 2(b) and 3(b) are the resulting tree structures for the precedence graphs in Figs. 2(a) and 3(a), respectively. Due to the dummy node 0 added in the transformation, the resulting graph \( G_p' \) is always connected. If the graph \( G_p \) is connected, the dummy node 0 is not absolutely necessary. But we still added it so that later discussions will be uniformly applicable for all cases. In the resulting tree \( G_p' \), node 0 is considered as the root node of the tree and is placed at the leftmost position. The job nodes with arcs linking to node 0 are drawn on the right-hand side of it and are called its children (node 0 is their parent). For each of these job nodes, the jobs with arcs linking to it, except its parent node, are placed to its right-hand side and are called its children, and so on. Note that the parent–child relations refer to the links in the tree \( G_p' \) in the undirected sense. A child of a job node may be its immediate predecessor or immediate successor depending on the direction of the arc between them.
After the above transformation, all jobs in \( G_p \) are then ordered in increasing order of the number of arcs on the path from the root node to the job. Ties can be broken arbitrarily without affecting the HBFDP result. Let \( a_0, a_1, a_2, \ldots, a_n \) be the resulting order.

We now show how the relaxed problem can be solved using HBFDP. The weighted job tardiness will be referred as “cost” for short. Let us define \( F_j(x) \) as the minimum cost of completing job \( j \) and all its children and descendants, such that job \( j \) is finished no earlier than time \( x \). Let \( I_j \) denote the set of immediate successors of job \( j \) and \( S_j \) the set of all the children of job \( j \).

The HBFDP begins with the computation of the cost of the \( a_n \)th node. Then we have the recursive equation

\[
F_j(C_j) = L_j(C_j) + \sum_{i \in S_j \cap I_j} \min_{C_i} F_i(C_i) + \sum_{i \in S_j, j \in I_i} \min_{C_i} F_i(C_i). \tag{16}
\]

The second term on the right-hand side of (16) is to select the minimum cost from all possible completion times \( \{C_i\} \) for all \( C_i \geq C_j + p_i \) where job \( i \) belongs to the set of immediate successors of job \( j \). Similarly, the third term is computed for all \( C_i \leq C_j - p_j \) with respect to immediate predecessors of job \( j \).

For \( j \) from \( a_n \) to \( a_0 \), compute (16) in turn. In this way, when we compute \( F_j(C_j) \), the function \( F_i(C_i) \) is already known for all children of job \( j \) in the tree. Then, because job 0 has all other \( n \) jobs as its children or descendants, a lower bound of the relaxed problem objective can be obtained from

\[
L = F_0(0) - \sum_{k=1}^{k} u_k. \tag{17}
\]

### 4.1. An example

Consider a problem with job precedence relations shown in Fig. 3(a). The precedence relation can also be seen from the second column of Table 1. The next three columns of Table 1 provide processing times, weights and due dates of the jobs. At the first iteration in the process of solving the Lagrangian dual problem, the Lagrangian multipliers \( \{u_k\} \) are initialized at 0 in the relaxed problem. The whole process of the HBFDP for solving this initial relaxed problem is described as follows.

**Preparation:** Calculate the value ranges of \( C_j \)’s and order the jobs:

- **Step 1:** Calculate the earliest and latest possible completion times of each job. They define the set of possible completion times for the job. Such sets for the jobs in this example are: \( C_1 \in [2, 3, 4, 5, 6], C_2 \in [3, 4, 5, 6], C_3 \in [6, 7], C_4 \in [8, 9, 10, 11], C_5 \in [8, 9, 10, 11], C_6 \in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] \).

- **Step 2:** Transform the job precedence graph to form a tree structure as shown in Fig. 3(b). The jobs are then ordered based on their positions on the tree: 0, 1, 6, 3, 2, 4, 5.

**Recursion:** Calculate \( F_j(C_j) \)’s backwards from the last job in the above order:

- **Step 3:** Since \( S_5 = \emptyset \) and \( I_5 = \emptyset \), we have

\[
F_5(C_5) = L_5(C_5) = w_5 \times \max\{C_5 - d_5, 0\} + \sum_{k=C_5-p_5+1}^{C_5} u_k.
\]

So \( F_5(8) = 4; F_5(9) = 6; F_5(10) = 8; F_5(11) = 10. \)
5.1. The generation of problem instances

In an attempt to compare the algorithms under rigorous experimental conditions, positive job weights and processing times were generated from the discrete uniform distribution \([1,10]\). Integer due dates were established from the uniform distribution \([K(1-t-R/2), K(1-t+R/2)]\) where both \(t\) and \(R\) are selected from \([0.2, 0.4, 0.6, 0.8, 1.0]\). Additionally,
the problem sizes used in the experiment were set at nine different levels, ranging from 20 to 100 jobs. Ten replications for each combination of \( n, t \) and \( R \) were generated, resulting in a total of 2250 problem instances.

The precedence relations in the test problems were generated as follows. A set of precedence relations between jobs was generated by applying the dominance rules mentioned in [4]. A simple procedure was then implemented to randomly break cycles in the precedence graph, leading to the final graph \( G_p \) used in our experiment. In such a graph it is possible for a job \( j \) to have multiple immediate predecessors or successors.

5.2. The comparison of the two LR algorithms

In the previous work in [4], constraints (8) are also relaxed to the objective function by introducing Lagrangian multipliers. In their work, due to the fact that FDP cannot be used to solve a problem where a job has more than one immediate successor, some precedence relations in \( G_p \) are deleted so that each job has at most one immediate successor. FDP is then performed to solve the corresponding Lagrangian relaxed problem in order to compute a lower bound. In the following comparisons, the LR algorithm using FDP is referred to as LRFDP and our LR algorithm using HBFDP as LRHBFDP. It can be observed from Section 4 that none of the arcs are deleted when using LRHBFDP because this algorithm can deal with jobs with multiple immediate predecessors or successors. Theoretically, reducing the number of relaxed or deleted constraints often results in faster convergence and better results. This is also supported by our experiment results.

Each of the problem instances was solved using the two algorithms respectively. Each algorithm gave three outputs: the value of the dual function \( L \) (a lower bound of the total weighted tardiness, it will be called “dual cost” for short hereafter), the total weighted tardiness of the heuristic feasible solution (“feasible cost” for short), and the number of iterations used. For each problem instance, we also calculated the relative improvements of LRHBFDP over LRFDP. On one hand, for the dual cost (lower bound), a higher value means closer to the optimal solution. The relative improvement is thus measured by \( \% \text{Imp} = (\text{LRHBFDP} - \text{LRFDP}) \times 100 / \text{LRFDP} \), where LRHBFDP and LRFDP are the dual costs resulted from the two algorithms, respectively for the problem instance and \( \text{LRFDP} \) is the average dual cost from LRFDP for the group of problem instances with the same size as the instance tested. We use \( \text{LRFDP} \) as the reference for measuring the relative improvement, rather than using the LRFDP for the instance tested, because LRFDP can be zero for some instances and hence \( \% \text{Imp} \) cannot be defined that way. On the other hand, for the feasible cost (heuristic solution), a lower value means closer to the optimal solution. The relative improvement is therefore measured by \( \% \text{Imp} = (\text{LRFDP} - \text{LRHBFDP}) \times 100 / \text{LRHBFDP} \), where LRHBFDP and LRFDP are the feasible costs resulted from the two algorithms, respectively for the problem instance and \( \text{LRHBFDP} \) is the average dual cost from LRHBFDP for the group of problem instances with the same size as the instance tested. The average performance measures for problems with different sizes are presented in Table 2. Each number in the table is an average over 250 problem instances. Table 3 shows the variations of the \( \% \text{Imp} \) measures for different problem sizes: the standard deviations (SD), the worst case (Min), and the best case (Max).

Table 2
Average results for different problem sizes

<table>
<thead>
<tr>
<th>( n )</th>
<th>Dual cost</th>
<th>Feasible cost</th>
<th>Iteration no.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRFDP</td>
<td>LRHBFDP</td>
<td>%Imp</td>
</tr>
<tr>
<td>20</td>
<td>1323.42</td>
<td>1330.82</td>
<td>0.56</td>
</tr>
<tr>
<td>30</td>
<td>2827.27</td>
<td>2871.64</td>
<td>1.57</td>
</tr>
<tr>
<td>40</td>
<td>4927.34</td>
<td>5057.11</td>
<td>2.63</td>
</tr>
<tr>
<td>50</td>
<td>7430.36</td>
<td>7718.48</td>
<td>3.88</td>
</tr>
<tr>
<td>60</td>
<td>10488.63</td>
<td>11029.43</td>
<td>5.16</td>
</tr>
<tr>
<td>70</td>
<td>13816.16</td>
<td>14608.06</td>
<td>5.73</td>
</tr>
<tr>
<td>80</td>
<td>17425.29</td>
<td>18594.55</td>
<td>6.71</td>
</tr>
<tr>
<td>90</td>
<td>21716.33</td>
<td>23479.14</td>
<td>7.91</td>
</tr>
<tr>
<td>100</td>
<td>26991.10</td>
<td>29225.52</td>
<td>8.28</td>
</tr>
<tr>
<td>Average</td>
<td>11882.88</td>
<td>12657.19</td>
<td>4.71</td>
</tr>
</tbody>
</table>

Note: LRFDP, LRHBFDP and \( \% \text{Imp} \) correspond to an average on 250 problem instances respectively for each job size.
Table 3
Variations of the relative improvements for different problem sizes

<table>
<thead>
<tr>
<th>n</th>
<th>%Imp of dual cost</th>
<th></th>
<th>%Imp of feasible cost</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SD</td>
<td>Min</td>
<td>Max</td>
<td>SD</td>
</tr>
<tr>
<td>20</td>
<td>0.92</td>
<td>−0.46</td>
<td>7.07</td>
<td>0.52</td>
</tr>
<tr>
<td>30</td>
<td>1.99</td>
<td>−0.25</td>
<td>11.34</td>
<td>0.58</td>
</tr>
<tr>
<td>40</td>
<td>2.92</td>
<td>−0.52</td>
<td>14.03</td>
<td>0.60</td>
</tr>
<tr>
<td>50</td>
<td>5.45</td>
<td>−1.19</td>
<td>28.76</td>
<td>0.85</td>
</tr>
<tr>
<td>60</td>
<td>6.03</td>
<td>−0.73</td>
<td>27.65</td>
<td>0.79</td>
</tr>
<tr>
<td>70</td>
<td>6.79</td>
<td>−0.67</td>
<td>43.05</td>
<td>0.79</td>
</tr>
<tr>
<td>80</td>
<td>7.56</td>
<td>−0.96</td>
<td>32.74</td>
<td>0.78</td>
</tr>
<tr>
<td>90</td>
<td>9.04</td>
<td>−1.76</td>
<td>38.69</td>
<td>0.80</td>
</tr>
<tr>
<td>100</td>
<td>9.54</td>
<td>−0.89</td>
<td>46.79</td>
<td>0.72</td>
</tr>
</tbody>
</table>

The results indicate that LR_{HBFD} gives much better lower bounds as compared to LR_{FDP}. The overall average relative improvement is 4.71% and the improvement increases as the problem size increases. The variation of the improvements also increases with problem size as can be observed from the standard deviations. In the worst case, LR_{HBFD} performs slightly worse than LR_{FDP}, but the difference is very small and such cases only account for a small proportion of the problem instances. In the best case, LR_{HBFD} performs over 40% better than LR_{FDP}. The tighter lower bound given by our algorithm can be very useful for providing a better bounding scheme in a branch and bound algorithm and for more accurately measuring performance of heuristic solutions. On the feasible cost (heuristic solution), both the average improvements and the variations are smaller, with LR_{HBFD} performing 0.5% better overall.

Our algorithm uses fewer iterations than LR_{FDP} with an average difference of 3 iterations. Fig. 4 illustrates the evolution of the dual cost for example problems with different problem sizes for \( t = 0.6 \) and \( R = 0.2 \). The results and the figures indicate that LR_{HBFD} converges faster than LR_{FDP}. That is, a better dual cost can be obtained in the same number of or fewer iterations using LR_{HBFD} in accordance with the above %Imp results.

From the above analysis of the results, we can conclude that our algorithm LR_{HBFD} performs better on average than LR_{FDP} on all three measures, lower bound, heuristic solution and convergence. The most significant improvement is on the lower bound (dual cost) and the improvement increases with the problem size. We now give a detailed analysis of the results on this performance measure to understand the influences of \( t \) and \( R \). The average %Imp on the dual cost for different combinations of \( t \) and \( R \) are presented in Table 4. Each number in the table, except those in the last row and last column, is an average over 90 problem instances.

From the way of generating the due dates for the problem instances, we know that the due dates become tighter as \( t \) increases. The due dates of the jobs spread out in a wider range when \( R \) is large and are more concentrated when \( R \) is small. Since the job completion times must be different on the single machine, it is easier to meet loose and spreading due dates. When \( t = 0.2 \) and \( R = 0.6, 0.8 \) or 1.0, the due dates are loose and spreading. For most of these problems, even LR_{FDP} found optimal solutions and thus there is no room for improvement. Therefore the %Imp is 0 in these cases. As the due dates become more and more difficult to meet (\( t \) increases and \( R \) decreases), the relative improvement increases, i.e., the improvement on the lower bound by LR_{HBFD} over LR_{FDP} become larger and larger. This can be observed clearly from the average results in the last row and the last column of Table 4.

6. Conclusions

In this paper we discussed the \( 1|\text{prec}||\sum w_j T_j | \) problem without cycles in the job precedence graph \( G_p \). Compared with previous LR method with FDP, the designed LR algorithm with HBFDP performs better with faster convergence and tighter lower bounds for solving both small and large sized problems. This can be attributed to the transformation strategy applied on \( G_p \) and the resulting development of HBFDP for solving Lagrangian relaxed problem. When LR is exploited to provide a lower bound for a branch and bound algorithm, it is critical to obtain a tighter Lagrangian
lower bound because it can help eliminate hopeless branches and therefore reduce the computation time, especially for large-scale problems. Further studies may be done to extend the method to cope with the case with cycles in the precedence graph.

Fig. 4. Evolution of the dual cost for different sized problems.
Table 4
Relative improvements of dual cost (%Imp) for different $t$ and $R$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$R$</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>6.65</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>12.91</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>9.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.22</td>
</tr>
<tr>
<td>0.2</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>0.4</td>
<td>0.68</td>
<td>0.13</td>
</tr>
<tr>
<td>0.6</td>
<td>4.08</td>
<td>2.81</td>
</tr>
<tr>
<td>0.8</td>
<td>11.09</td>
<td>9.21</td>
</tr>
<tr>
<td>1.0</td>
<td>9.44</td>
<td>9.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.43</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>0.68</td>
<td>2.24</td>
</tr>
<tr>
<td></td>
<td>4.08</td>
<td>9.08</td>
</tr>
<tr>
<td></td>
<td>9.44</td>
<td>8.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.99</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>2.81</td>
<td>7.40</td>
</tr>
<tr>
<td></td>
<td>9.21</td>
<td>9.39</td>
</tr>
<tr>
<td></td>
<td>9.98</td>
<td>9.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.87</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>2.53</td>
<td>9.94</td>
</tr>
<tr>
<td></td>
<td>7.40</td>
<td>9.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.71</td>
</tr>
</tbody>
</table>

Acknowledgements

This research is partly supported by National Natural Science Foundation for Distinguished Young Scholars of China (Grant no. 70425003), National Natural Science Foundation of China (Grant no. 60274049 and 70171030), Fok Ying Tung Education Foundation and the Excellent Young Faculty Program and the Ministry of Education, China. The authors also thank the anonymous referees for the valuable suggestions.

References