TWO TYPES OF BRANCH-AND-BOUND ALGORITHMS
FOR THE SCHEDULING PROBLEM TO MINIMIZE TOTAL TARDINESS
ON IDENTICAL PARALLEL MACHINES

Shunji Tanaka and Mituhiko Araki
Graduate School of Electrical Engineering, Kyoto University
1 Kyoto-daigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan
tanaka@kuee.kyoto-u.ac.jp, araki@kuee.kyoto-u.ac.jp

Abstract
We construct two types of branch-and-bound algorithms to minimize total tardiness on identical parallel machines. The one is an improvement of the existing algorithm, and the other is based on the dynamic programming model for general parallel-machine scheduling problems. Computational experiments show that our algorithms can handle instances with up to 20 jobs while the existing algorithm can only handle instances with up to 12 jobs.

Keywords: parallel machine scheduling, total tardiness, optimization, branch-and-bound algorithm.

1. INTRODUCTION
The single machine total tardiness problem \((1||\sum T_j)\) has been extensively studied since the pioneering research by Emmons (1968), although this problem is shown to be NP-hard (Du and Leung 1990). Emmons proved that some precedence relations of jobs hold in an optimal schedule of the problem, which are called Emmons' dominance conditions. Another important contribution in this field of research is done by Lawler (1977). His theorem, called Lawler's decomposition theorem, enables us to decompose the original problem into two subproblems. By utilizing it, he also offered a pseudopolynomial algorithm to solve \(1||\sum T_j\). After the Lawler's research, the solution algorithm has been improved by several researchers, and the algorithm by Szwarc et al. (1999) was reported to be able to handle instances with up to 300 jobs.

As for the problem to minimize total tardiness on identical parallel machines \((P||\sum T_j)\), most researches focus only on heuristic procedures, and there are few on strict solution algorithms (Root 1965, Elmaghraby and Park 1974, Barnes and Brennan 1977, Azizoglu and Kirca 1998). Among these researches only Azizoglu and Kirca targeted the general \(P||\sum T_j\) problem: Root treated the common due date problem \(P|d_j=d'|\sum T_j\), and Elmaghraby and Park, and Barnes and Brennan treated the problem \(P|p_j=d_j|\sum T_j\). Furthermore, the branch-and-bound algorithm by Azizoglu and Kirca can only handle instances with up to 12 jobs.

In this paper we propose two types of branch-and-bound algorithms to solve the general \(P||\sum T_j\) problem. The one is an improvement of the algorithm by Azizoglu and Kirca, and the other is based on a dynamic programming model for general parallel-machine scheduling problems. Computational experiments show that our algorithms can handle instances with up to 20 jobs.

2. PROBLEM DESCRIPTION AND NOTATION
Consider that \(N\) jobs \(J_1, \ldots, J_N\) are to be processed on \(M\) identical parallel machines \(M_1, \ldots, M_M\). Each job \(J_j\) has the processing time \(p_j\) and the due date \(d_j\). All the jobs are available at time zero, and no job preemption is allowed. The tardiness \(T_j\) of \(J_j\) is given by \(T_j = \max(C_j - d_j, 0)\), where \(C_j\) is the completion time of \(J_j\). Our objective is to obtain a schedule that minimizes the total tardiness \(\sum_{j=1}^{N} T_j\).

Since the total tardiness is a nondecreasing function of \(C_j\) \((1 \leq j \leq N)\), there exists an optimal schedule in which no idle times are inserted between jobs. Thus we restrict the searched schedules to those without idle times. Therefore, a schedule can be determined by fixing the assignment of the jobs to the machines and the processing order of the jobs on each machine.

Here, we give the notation used in the following. The set of the jobs to be processed, i.e., \(\{J_1, \ldots, J_N\}\) is denoted by \(J\). The priority list of the jobs belonging to \(J\) sorted by the nonincreasing order of the processing times (ties are broken by the nondecreasing order of the due dates) is denoted by \(\text{SPT}(J)\). The priority list sorted by the nondecreasing order of the due dates (ties are broken by the nonincreasing order of the processing times) is denoted by \(\text{EDD}(J)\). The procedure to assign jobs one by one to earliest available machines (ties are broken arbitrary) according to a priority list \(L\) is denoted by \(\text{D}(L)\). The set of all the schedules that can be constructed by \(\text{D}(\bullet)\) is denoted by \(\mathcal{S}_0\).
3. OUTLINE OF THE ALGORITHMS

Two types of algorithms, Algorithms A and B, are proposed in this paper. In Algorithm A, an priority list is searched by a branch-and-bound algorithm as in the algorithm by Azizoglu and Kirca. Since it is easily checked that an optimal schedule belongs to \( S_0 \), the problem can be solved by obtaining an optimal priority list corresponding to an optimal schedule.

In Algorithm B, the assignment of the jobs to the machines is searched by a branch-and-bound algorithm based on the dynamic programming model by Gupta and Maykut (1973), and the processing order of jobs on each machine is determined by solving the single-machine total tardiness problem \((1 || \sum T_j)\). Since it is not necessary to consider such redundant schedules that all the jobs are assigned to one machine, we restrict the searched schedules to \( S_0 \) as in Algorithm A.

4. PREPROCESSING

For a given problem instance, the following preprocessing procedures are applied before applying Algorithm A or B.

4.1 Removal of non-tardy jobs

First, we reduce the problem size by removing non-tardy jobs.

It is straightforward to check that the completion times \( C_k \) \((1 \leq k \leq N)\) in any schedules belonging to \( S_0 \) satisfy

\[
C_k \leq \frac{1}{M} \sum_{j=1}^{N} p_j + \frac{M-1}{M} p_k.
\]

(1)

Therefore, if

\[
d_k \geq \frac{1}{M} \sum_{j=1}^{N} p_j + \frac{M-1}{M} p_k,
\]

(2)

is satisfied, \( J_k \) is always non-tardy. To remove such non-tardy jobs from \( J \), we apply the following \( O(N^2) \) algorithm.

Procedure REMOVE

0° \( J' := J \), \( R := \phi \).

1° \( k := 1 \).

2° If \( J_k \notin J' \), go to 5°.

3° If \( J_k \) satisfies

\[
d_k \geq \frac{1}{M} \sum_{j \in J'} p_j + \frac{M-1}{M} p_k
\]

(3)

go to 4°. Otherwise, go to 5°.

4° Remove \( J_k \) from \( J' \) and insert \( J_k \) into the first position of \( R \). Go to 1°.

5° \( k := k + 1 \). If \( k \leq N \), go to 2°.

An optimal schedule for \( J \) can be obtained by applying \( D(R) \) to an optimal schedule for \( J' \). In the following, we assume that \( J \) is redefined by \( J' \) and \( N \) by \( |J'| \) after applying this procedure.

4.2 Optimality test of SPT

If all the jobs are tardy \((C_k \geq d_k, 1 \leq k \leq N)\) in the schedule constructed by \( D(\text{SPT}(J')) \), it is known to be optimal (Koulamas 1997). In such a case, neither Algorithm A nor B is applied since an optimal schedule is already obtained.

4.3 Initial schedule

An initial schedule for Algorithms A and B that is used for the calculation of an initial upper bound is constructed as follows.

(1) Construct two schedules by \( D(\text{EDD}(J)) \) and \( D(\text{SPT}(J)) \) (it is already constructed in 4.2).

(2) In these schedules, optimize the processing order of jobs on each machine by solving \( 1 || \sum T_j \). The solution algorithm for \( 1 || \sum T_j \) is given in Appendix A.

(3) Adopt the better one as the initial schedule.

If the initial schedule makes all the jobs non-tardy, neither Algorithm A nor B is applied since the initial schedule is optimal.

5. ALGORITHM A

In Algorithm A, an optimal priority list is searched by the depth-first branch-and-bound algorithm as the algorithm by Azizoglu and Kirca. Branching is performed by fixing the elements of the priority list from the first to the last. Thus, a subproblem corresponding to a node at level \( l \) is to determine the last \((N-l)\) elements of the priority list. The primary differences between our algorithm and the algorithm by Azizoglu and Kirca are summarized as:

(1) Improvement of lower bounds by introducing dummy jobs.

(2) Improvement of the fathoming test by SPT.

(3) Reduction of branches in the search tree via the Emmons' dominance conditions (Emmons 1968).

In the following, we denote by \( L' \) a partial priority list of length \( l \) and by \( S' \) the partial schedule constructed by \( D(L') \). The set of \((N-l)\) unscheduled jobs is denoted by \( J' \). Let us further denote by \( c_{m}^{j} \) \((1 \leq m \leq M)\) the maximum completion time of the jobs on \( M_n \) in \( S' \) and define \( m^* = \arg \min_{m} c_{m}^{j} \).

5.1 Lower bound

A lower bound for \( D(L') \) is calculated by the following procedure.

(1) Introduce \((M-1)\) dummy jobs \( J')_{1}^{M} \((1 \leq m \leq M, m \neq m^*)\) such that their processing times and due dates are given by \( (c_{m}^{j} - c_{m^*}^{j}) \) and \( d_{m}^{j} \), respectively.

(2) Calculate a lower bound for \( D(L') \) together with the \((M-1)\) dummy jobs by the method described in Appendix B, where the release dates of all these jobs are set to be \( c_{m^*}^{j} \) (or, equivalently, the due dates of all these jobs are decreased by \( c_{m^*}^{j} \)).
5.2 Fathoming test by SPT

By extending the result on the optimally of SPT explained in 4.2, we can show that if all the jobs belonging to \( \mathcal{U}' \) are tardy in the schedule constructed by applying \( \mathcal{D}(\text{SPT}(\mathcal{U}')) \) to \( S' \), this schedule is optimal under the condition that \( S' \) is fixed. Therefore, in such a case the node is fathomed and the incumbent solution is updated if necessary.

In the algorithm by Azizoglu and Kirca, the node is fathomed only when all the due dates \( d_a \) of jobs \( J_a \in \mathcal{U}' \) satisfy
\[
d_a = p_a + \min_a \bar{c}_a,
\]
therefore, their fathoming test is more conservative than our proposed one.

5.3 Reduction of branches in the search tree via the Emmons’ dominance conditions

The Emmons’ dominance conditions offer the conditions that should be satisfied in an optimal schedule for \( 1 \| \sum T_j \).

We restrict the candidates for the \((l+1)\)th job in the priority list to those satisfying them.

Let us assume that the processing order of the jobs on \( M_{l+1} \) in \( S' \) is given by \( J_1, \ldots, J_l \). Since the \((l+1)\)th job in the priority list is assigned to the earliest available machine \( M_{l+1} \), \( J_{l+1} \in \mathcal{U}' \) cannot be a candidate for the \((l+1)\)th job in the priority list if the dominance conditions claim that \( J_{l+1} \) should precede \( J_l \) for some \( j \leq j \leq v \). The detailed procedure is given as follows.

**Reduction of branches via the Emmons’ dominance conditions**

If for some \( j \leq j \leq v \), \( J_{l+1} \in \mathcal{U}' \) satisfies at least one of the following three conditions, \( J_{l+1} \) is removed from the candidates for the \((l+1)\)th job in the priority list.

1. \( p_a < p_j \), \( d_a \leq \sum_{i=1}^{J} p_i \), \( d_j \).
2. \( p_a > p_j \), \( d_a < d_j \), \( p_i < p_j - \sum_{i=1}^{J} p_i \).
3. \( d_a = d_j \), \( p_i < p_j \).

6. **ALGORITHM B**

Let us denote by \( f(S_m, m) \) the minimum total tardiness of jobs belonging to \( S_m \) when they are scheduled on \( m \) identical parallel machines. From the conventional dynamic programming model for general parallel-machine scheduling problems (Gupta and Maykut 1973),
\[
f(S_m, m) = \min_{\mathcal{V}_m} \{f(\mathcal{V}_m, 1) + f(S_{m-1}, m-1)\}, \tag{4}
\]
\[
S_m = S_m - \mathcal{V}_m, \quad S_{m-1} = S_m - \mathcal{V}_m \tag{5}
\]
hold where \( \mathcal{V}_m \quad (1 \leq m \leq M) \) denotes the set of jobs assigned to \( M_m \). In Algorithm B, a branch-and-bound algorithm is constructed based on these relations. More specifically, \( \mathcal{V}_M, \ldots, \mathcal{V}_1 \) are recursively determined by the branch-and-bound algorithm. Thus, a subproblem corresponding to a node at level \( l \) is to determine \( \mathcal{V}_{M-l}, \ldots, \mathcal{V}_1 \) under the condition that \( \mathcal{V}_{M-l-1}, \ldots, \mathcal{V}_M \) are fixed. In other words, this subproblem is to determine the assignment of the jobs in \( S_{M-l} \) to \( M_{M-l}, \ldots, M_1 \) under the condition that the jobs already assigned to \( M_{M-l}, \ldots, M_{M-l-1} \) are fixed. To calculate \( f(\mathcal{V}_m, 1) \) in (4), i.e., to solve \( 1 \| \sum T_j \) for \( \mathcal{V}_m \), the algorithm given in Appendix A is applied.

6.1 Lower bound

Consider a subproblem corresponding to a node at level \( l \). A lower bound for the set of the unassigned jobs \( S_{M-l} \), i.e., a lower bound of \( f(S_{M-l}, M-l) \) is calculated by the method described in Appendix B as in Algorithm A.

6.2 Reduction of branches in the search tree

The candidates for \( \mathcal{V}_{M-l} \subset S_{M-l} \) are reduced by restricting the searched schedules to \( S_{M-l} \). Assume that \( \mathcal{V}_{M-l}, \ldots, \mathcal{V}_{M-1} \) and their processing orders on corresponding machines are fixed, and that the last processed job on \( M_0 \) \((M-l+1 \leq k \leq M)\) is denoted by \( J_{l_k} \) \((J_k \in \mathcal{V}_k)\). Then, \( \mathcal{V}_{M-l} \) should satisfy the following conditions.

For \( 1 \leq l \leq M-1 \):
\[
\sum_{J_j \in \mathcal{V}_{M-l}} p_j \leq \min_{M-l+1 \leq k \leq M} \sum_{J_j \in \mathcal{V}_k} p_j + \max_{J_j \in \mathcal{V}_{M-l}} p_j, \tag{7}
\]
where \( S_{M-l} = S_{M-l} - \mathcal{V}_{M-l} \).

7. COMPUTATIONAL EXPERIMENTS

We compare the efficiency of Algorithms A and B by computational experiments performed on a personal computer with a Pentium 2.4GHz.

Problem instances are generated by the Fisher’s standard method (Fisher 1976). First, the integer processing times \( p_j \) \((1 \leq j \leq N)\) are generated by the uniform distributions in \([1, 100]\). Then, let \( p_j = \sum_{j \in \mathcal{P}} p_j \) and the integer due dates \( d_j \) \((1 \leq j \leq N)\) are generated by the uniform distributions in \([P(1 - \tau - R/2)/M, P(1 - \tau - R/2)/M]\). We consider \( N = 15 \) and 20, \( M = 2 \) and 3, \( \tau = 0.2, 0.5 \) and 0.8, and \( R = 0.2, 0.6 \) and 1.0. For every combination of \( N, M, \tau \) and \( R, 100 \) problem instances are generated.

The results are shown in Tables 1–4. In these tables, average computational times over 100 problem instances are shown. Algorithms are terminated when the computational time exceeds 1000 sec. Italic numbers in Table 4 denote that averages are taken over instances successfully solved within 1000 sec, and the number of problems solved within 1000 sec are shown in parentheses. From these tables, Algorithm B outperforms Algorithm A for almost all problem instances, but when \( \tau \) is large, Algorithm A is faster than...
Table 1  Average computational times for \((N,M) = (15,2)\)

<table>
<thead>
<tr>
<th>(\tau_1/R)</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.2 0.6 1.0</td>
<td>0.2 0.6 1.0</td>
</tr>
<tr>
<td>0.2</td>
<td>1.3 0.1 0.0</td>
<td>0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.4 0.3 0.2</td>
<td>0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.8</td>
<td>0.0 0.1 0.1</td>
<td>0.0 0.0 0.0</td>
</tr>
</tbody>
</table>

Table 2  Average computational times for \((N,M) = (15,3)\)

<table>
<thead>
<tr>
<th>(\tau_1/R)</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.2 0.6 1.0</td>
<td>0.2 0.6 1.0</td>
</tr>
<tr>
<td>0.2</td>
<td>2.2 0.2 0.1</td>
<td>0.4 0.0 0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>1.6 0.1 0.1</td>
<td>0.6 0.5 0.4</td>
</tr>
<tr>
<td>0.8</td>
<td>0.1 0.4 0.1</td>
<td>0.6 0.6 0.6</td>
</tr>
</tbody>
</table>

Table 3  Average computational times for \((N,M) = (20,2)\)

<table>
<thead>
<tr>
<th>(\tau_1/R)</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.2 0.6 1.0</td>
<td>0.2 0.6 1.0</td>
</tr>
<tr>
<td>0.2</td>
<td>248.6 7.4 0.0</td>
<td>0.5 0.0 0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>49.0 24.6 17.0</td>
<td>1.1 0.8 0.5</td>
</tr>
<tr>
<td>0.8</td>
<td>0.9 3.7 8.9</td>
<td>1.0 0.8 0.8</td>
</tr>
</tbody>
</table>

Table 4  Average computational times for \((N,M) = (20,3)\)

<table>
<thead>
<tr>
<th>(\tau_1/R)</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.2 0.6 1.0</td>
<td>0.2 0.6 1.0</td>
</tr>
<tr>
<td>0.5</td>
<td>529.8 646 69</td>
<td>429.8 88</td>
</tr>
<tr>
<td>0.8</td>
<td>9.6 97.3 32.3</td>
<td>97</td>
</tr>
</tbody>
</table>

Algorithm B. It is because the fathoming test by SPT explained in 5.2 effectively works to reduce the computational time for such instances.

References


A SOLUTION ALGORITHM FOR \(1\|\Sigma T_j\)

Since the number of jobs in the single-machine total tardiness problem considered here is up to 20, it is not necessary to apply such complicated solution algorithms that are, in general, even slow when the problem size is small. In this paper we recursively apply the Brucker’s simple algorithm “sequence(1, 1)” (Brucker 2001) that is based on the Lawler’s dynamic programming (Lawler 1977). Here, three elimination rules by Lawler (1977), Potts and Van Vassenhove (1982) and Szwarc (1993) are added.

B ALGORITHM FOR A LOWER BOUND

We calculate a lower bound for \(P|\Sigma T_j|\) as in Azizoglu and Kirca (1998). Assume that \(N'\) jobs \(J_1', \ldots, J_N' \in S'\) are processed on \(M'\) machines \(M_1, \ldots, M_{M'}\) where the processing time and duedate of \(J_j'\) are given by \(p_{j'}\) and \(d_{j'}\), respectively.

First, we relax the constraints of the problem so that jobs are preemptive and can be processed simultaneously on different machines. This relaxed problem reduces to \(1\|\Sigma T_j\)

Let us denote \(S(T') = (J_1', \ldots, J_N')\) and \(EDD(S') = (J_1', \ldots, J_N')\). Then, this lower bound is given by

\[
\sum_{j=1}^{N'} \max \left( \frac{1}{M'} \sum_{k=1}^{M'} p'_{(k)} - d'_{(k)}; 0 \right).
\]