Solving Parallel Machine Scheduling Problems by Column Generation

Zhi-Long Chen
Department of Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104-6315

Warren B. Powell
Department of Civil Engineering & Operations Research
Princeton University
Princeton, NJ 08544, USA

July 1996
(Revised April 1997, January 1998)

Abstract

We consider a class of problems of scheduling n jobs on m identical, uniform, or unrelated parallel machines with an objective of minimizing an additive criterion. We propose a decomposition approach for solving these problems exactly. The decomposition approach first formulates these problems as an integer program, and then reformulates the integer program, using Dantzig-Wolfe decomposition, as a set partitioning problem. Based on this set partitioning formulation, branch and bound exact solution algorithms can be designed for these problems. In such a branch and bound tree, each node is the linear relaxation problem of a set partitioning problem. This linear relaxation problem is solved by a column generation approach where each column represents a schedule on one machine and is generated by solving a single machine subproblem. Branching is conducted on variables in the original integer programming formulation instead of variables in the set partitioning formulation such that single machine subproblems are more tractable. We apply this decomposition approach to two particular problems: the total weighted completion time problem and the weighted number of tardy jobs problem. The computational results indicate that the decomposition approach is promising and capable of solving large problems.

Key words: Parallel machine scheduling; Set partitioning; Dantzig-Wolfe decomposition; Column generation; Branch and bound

1 Introduction

We consider a class of problems of scheduling n independent jobs $N = \{1, 2, ..., n\}$ on m identical, uniform, or unrelated parallel machines $M = \{1, 2, ..., m\}$ with an objective of minimizing an additive criterion. For ease of presentation, we denote this class of problems as PMAC. In the PMAC problems, each job $i \in N$ has m processing times p_{ij} ($j \in M$), a weight w_i, a due date d_i, and probably other problem dependent parameters. Here p_{ij} is the actual processing time of job i if it is processed on machine j. As a convention, processing times p_{ij}, for all $i \in N, j \in M$, satisfy the following properties. In the case of identical machines, all the machines have the same speed and hence processing times of a job are identical on different machines, i.e. $p_{ij} \equiv p_i$. In the case of uniform machines, machines may have different speeds and processing times of a job may differ by speed factors, i.e. $p_{ij} = p_i/s_j$, where s_j reflects the speed of machine j. Finally, in the case of unrelated machines, p_{ij} is arbitrary and has no special characteristics. We are interested in classical settings, that is, all the parameters are deterministic; all the jobs are available for processing at time zero; and no preemption is allowed during processing.

Let C_j denote the completion time of job j in a schedule. Then a general additive criterion can be denoted as $\sum_{j \in N} f_j(C_j)$, where $f_j(\cdot)$ is a real-valued function. Using the commonly accepted three field classification terminology for machine scheduling problems (see, e.g. Lawler, Lenstra, Rinnooy Kan and Shmoys [22]), we denote the general PMAC problem by $P||\sum f_j(C_j)$ for the identical machine case, $Q||\sum f_j(C_j)$ for the uniform machine case, and $R||\sum f_j(C_j)$ for the unrelated machine case. In this paper, we mainly focus on the following two particular problems: the total weighted completion time problem, denoted as $P||\sum w_jC_j$, $Q||\sum w_jC_j$, or $R||\sum w_jC_j$, respectively for the identical, uniform, or unrelated machine case; and the weighted number of tardy jobs problem, denoted similarly as $P||\sum w_jU_j$, $Q||\sum w_jU_j$, or $R||\sum w_jU_j$, where $U_j = 1$ if job j is late, i.e. $C_j > d_j$, and 0 otherwise.

Needless to say, the PMAC problems are fundamental to numerous complex real-

world applications. They are widely noted in survey papers (e.g. Cheng and Sin [9], and Lawler, Lenstra, Rinnooy Kan and Shmoys [22]) and books (e.g. Baker [1], and Pinedo [26]). Unfortunately, most of them, including $P||\sum w_jC_j$, the easiest case of the total weighted completion time problem, and $P||\sum w_jU_j$, the easiest case of the weighted number of tardy jobs problem, are NP-hard, and very few exact solution algorithms can be found in the literature. As noted in Lawler, Lenstra, Rinnooy Kan and Shmoys [22], the dynamic programming techniques of Rothkopf [27] and Lawler and Moore [23] can solve some of these problems, including the total weighted completion time problem and the weighted number of tardy jobs problem, in which it is possible to schedule jobs on a single machine in a predetermined order. However, it is impractical to solve even very small sized problems using these algorithms which require prohibitively high order of time: $O(m\min\{3^n, n2^n\})$ for $P||\sum f_j(C_j)$; $O(mnC^{m-1})$ for $Q||\sum w_jC_j$; and $O(mnC^m)$ for $R||\sum w_jC_j$, $Q||\sum w_jU_j$, and $R||\sum w_jU_j$, where C is an upper bound on the completion time of any job in an optimal schedule. To the best of our knowledge, there is no other exact solution algorithms in the literature for the problems: $Q||\sum w_jC_j$, $R||\sum w_jC_j$, $P||\sum w_jU_j$, $Q||\sum w_jU_j$, and $R||\sum w_jU_j$.

For the problem $P||\sum w_jC_j$, however, besides this possible dynamic programming algorithm, there are lower bounding techniques in the literature that can be used to design branch and bound algorithms. Eastman, Even and Isaccs [13] give a lower bound for the optimal objective function. This lower bound has been the basis for the branch and bound algorithms of Elmaghraby and Park [14], Barnes and Brennan [2], and Sarin, Ahn and Bishop [28]. Recently, Webster [33, 34, 35] gives tighter lower bounds based on the idea of considering a job as a collection of subjobs linked together by a group constraint. However, there is no branch and bound algorithm reported in the literature using Webster's lower bounds. Belouadah and Potts [4] formulate the problem $P||\sum w_jC_j$ as an integer program and use the well-known Lagrangian relaxation lower bounding scheme to design a branch and bound algorithm which is capable of solving instances with up to 3 machines and 30 jobs within a reasonable time.

Since we mainly focus on the total weighted completion time problem and the

weighted number of tardy jobs problem, we only give a brief review of algorithms for other PMAC problems. Barnes and Brennan [2] propose a branch and bound exact solution algorithm for the total tardiness problem in the identical machine case. Not surprisingly, their algorithm can only solve problems with up to 20 jobs and 4 identical machines. Besides this, De, Ghosh and Wells [11] give an $O(nm[w(n/m+1)]^{2m})$ dynamic programming exact solution algorithm for a problem involving earliness, tardiness and due date penalties. Obviously, this DP algorithm is only capable of solving problems with a small number of machines and jobs. For possible algorithms for special cases of the PMAC problems, the reader is referred to the survey papers: Cheng and Sin [9], and Lawler, Lenstra, Rinnooy Kan and Shmoys [22].

In this paper, we propose a decomposition approach for solving the PMAC problems exactly. The decomposition approach first formulates these problems as an integer program, and then reformulates the integer program, using Dantzig-Wolfe decomposition, as a set partitioning problem. Based on this set partitioning formulation, branch and bound exact solution algorithms can be designed for the PMAC problems. In such a branch and bound tree, each node is the linear relaxation problem of a set partitioning problem. This linear relaxation problem is solved by a column generation approach where each column represents a schedule on one machine and is generated by solving a single machine subproblem. Branching is conducted on variables in the original integer programming formulation instead of variables in the set partitioning problem such that the resulting subproblems are more tractable.

We apply this decomposition approach to the two particular problems: the total weighted completion time problem and the weighted number of tardy jobs problem. The computational results indicate that the decomposition approach is promising and capable of solving large problems.

The success of our decomposition approach is mainly due to the excellent lower bounding obtained from the linear relaxation of the set partitioning problem. We will see later that in the case of the total weighted completion time problem, for each problem size tested, the average deviation (based on twenty test problems) of the linear relaxation

solution value from the integer solution value is always within 0.1%; and in the case of the weighted number of tardy jobs problem, this average deviation is always within 0.8%. Due to extremely tight lower bounds, very few branch and bound nodes need to be explored in the corresponding branch and bound algorithms.

We note that at the same time when we were conducting this research, van den Akker, Hoogeveen and van de Velde [30] independently suggested a similar approach to the PMAC problems. However, the branching strategy used by them is different from that used by us. Their branching is based on the completion times of jobs appearing in a fractional solution, while ours is based on the ordering relations of jobs. They applied the approach to the problem $P||\sum w_jC_j$ and showed its effectiveness. Quite interestingly, also at the same time, Chan, Kaminsky, Muriel and Simchi-Levi [7] independently proposed and analyzed the column generation approach to the set partitioning formulation of the problem $P||\sum w_jC_j$. Their emphasis is on worst-case and probabilistic analysis. They proved in theory that the worst-case deviation of the linear relaxation solution value from the integer solution value is no more than $\frac{\sqrt{2}-1}{2} \times 100\%$.

This paper is organized as follows. The decomposition approach is described for the general PMAC problem in the next section. Then in Sections 3 and 4, this approach is applied, respectively, to the total weighted completion time problem ($P||\sum w_jC_j$, $Q||\sum w_jC_j$, and $R||\sum w_jC_j$) and to the weighted number of tardy jobs problem ($P||\sum w_jU_j$, $Q||\sum w_jU_j$, and $R||\sum w_jU_j$). The resulting single machine subproblems are NP-hard and solved by pseudo-polynomial dynamic programming algorithms. In Section 5, computational experiments are conducted and their results are reported. Finally, in Section 6, we conclude the paper.

2 Decomposition Approach for the PMAC Problems

In this section, we describe in detail the decomposition approach for the PMAC problems. First, in Section 2.1, we give a general integer programming formulation for the

problems. Then, in Section 2.2, we decompose this formulation, using Dantzig-Wolfe decomposition, into a set partitioning master problem and m single machine subproblems. Then the linear relaxation of the set partitioning problem is solved by a column generation procedure. Finally, in Section 2.3, a branch and bound exact solution algorithm is briefly described.

We note that the decomposition approach is applicable, directly or indirectly, to virtually every individual PMAC problem. However, the efficiency of the approach mainly depends on whether single machine subproblems resulted from the decomposition can be solved efficiently. The formulations we are going to give (e.g. **IP1** and **IP2** of Section 2.1, and **SP1** and **SP2** of Section 2.2) only serve as a representative for numerous PMAC problems. When a particular PMAC problem is concerned, we may not directly apply these general formulations to the problem; instead, it may be necessary to modify these formulations appropriately so that more efficient algorithms for the problem can be designed. As we will see later, for the total weighted completion time problem, we directly apply these formulations, while for the weighted number of tardy jobs problem, we slightly modify these formulations.

2.1 Integer Programming Formulation

First define a *partial schedule* on a single machine to be a schedule formed by a subset of jobs of N on that machine. Clearly, a schedule for a PMAC problem (where m machines are involved) consists of m partial schedules, one for each machine.

For a given PMAC problem, there may exist a predetermined job ordering restriction which specifies, for each job i, a set of jobs that must be scheduled before or after job i. The job ordering restriction in a problem could be given both externally by the problem itself (e.g. precedence constraints for jobs imposed by the problem) and internally by optimality properties (e.g. some ordering patterns an optimal schedule must follow). Note that, in the branch and bound algorithm described later, each branch and bound node is a PMAC problem with an additional constraint imposed by

the branching rule which enforces some jobs to be scheduled before or after some other jobs. The job ordering restriction in the problem corresponding to each branch and bound node includes the job ordering constraint imposed by the branching rule as well.

Define a *feasible partial schedule* on a machine as a partial schedule on that machine which satisfies the given job ordering restriction. To explicitly take into account the possible job ordering restriction in a given PMAC problem, we only need to consider those schedules where the partial schedule on each machine is feasible. Let us define the following sets, for $j \in N$ and $k \in M$:

$A_j^k = \{i \in N \mid i \text{ can succeed } j \text{ in a feasible partial schedule on machine } k\}$

$B_j^k = \{i \in N \mid i \text{ can precede } j \text{ in a feasible partial schedule on machine } k\}$

In a problem without precedence constraints, if we do not know any ordering pattern that an optimal schedule must follow, then any partial schedule is feasible and simply $A_j^k = B_j^k = N \setminus \{j\}$ for all j and k. In the total weighted completion time problem and the weighted number of tardy jobs problem, as we will see later, there is some ordering pattern that an optimal schedule must follow and hence the sets A_j^k and B_j^k may be much smaller than the set $N \setminus \{j\}$.

As we will see soon, the sets A_j^k and B_j^k defined here and the sets A_j and B_j defined later are used, both in the master problem level and in the subproblem level, to express mathematically the given job ordering restriction of the problem.

Define the following $0-1$ variables, for $i, j \in N$ and $k \in M$:

$$x_{ij}^k = \begin{cases} 1 & \text{if job } j \text{ is processed immediately after job } i \text{ on machine } k \\ 0 & \text{otherwise} \end{cases}$$

$$x_{0j}^k = \begin{cases} 1 & \text{if job } j \text{ is processed first on machine } k \\ 0 & \text{otherwise} \end{cases}$$

$$x_{j,n+1}^k = \begin{cases} 1 & \text{if job } j \text{ is processed last on machine } k \\ 0 & \text{otherwise} \end{cases}$$

Then we have the following integer programming formulation (**IP 1**) for the problems $P||\sum f_j(C_j)$, $Q||\sum f_j(C_j)$, and $R||\sum f_j(C_j)$.

IP 1:

$$\min \sum_{j \in N} f_j(C_j) \tag{1}$$

subject to

$$\sum_{k \in M} \sum_{i \in B_j^k \cup \{0\}} x_{ij}^k = 1, \quad \forall j \in N \tag{2}$$

$$\sum_{j \in N} x_{0j}^k \leq 1, \quad \forall k \in M \tag{3}$$

$$\sum_{i \in B_j^k \cup \{0\}} x_{ij}^k = \sum_{i \in A_j^k \cup \{n+1\}} x_{ji}^k, \quad \forall j \in N, k \in M \tag{4}$$

$$C_j = \sum_{k \in M} \left(p_{jk} x_{0j}^k + \sum_{i \in B_j^k} (C_i + p_{jk}) \, x_{ij}^k \right), \quad \forall j \in N \tag{5}$$

$$x_{ij}^k \in \{0,1\}, \quad \forall i, j \in N, k \in M \tag{6}$$

The objective function (1) seeks to minimize a given additive criterion. Constraints (2) and (3) ensure that each job is processed exactly once and each machine is utilized at most once, respectively. Constraint (4) guarantees that the assignment of jobs to machines is well-defined. This constraint plays the same role as flow conservation constraints in many network flow problems. Constraint (5) defines completion time C_j. The last constraint (6) represents binary integrality requirement of $0-1$ variables. Constraints (4), (5) and (6) ensure that the partial schedule on each machine is feasible.

For the problem $P||\sum f_j(C_j)$, since all the machines are identical, we do not need to distinguish different machines, and hence the formulation (1)-(6) can be simplified. Define sets A_j and B_j and variables $x_{ij}, x_{0j}, x_{j,n+1}$ similarly to A_j^k, B_j^k, $x_{ij}^k, x_{0j}^k, x_{j,n+1}^k$, respectively, as follows:

$A_j = \{i \in N \mid i$ can succeed j in a feasible partial schedule on a single machine$\}$

$B_j = \{i \in N \mid i$ can precede j in a feasible partial schedule on a single machine$\}$

$$x_{ij} = \begin{cases} 1 & \text{if job } i \text{ is processed immediately before job } j \text{ on some machine} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{0j} = \begin{cases} 1 & \text{if job } j \text{ is processed first on some machine} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{j,n+1} = \begin{cases} 1 & \text{if job } j \text{ is processed last on some machine} \\ 0 & \text{otherwise} \end{cases}$$

Then the simplified integer programming formulation (**IP2**) for the problem $P||\sum f_j(C_j)$ is

IP2:

$$\min \quad \sum_{j \in N} f_j(C_j) \tag{7}$$

subject to

$$\sum_{i \in B_j \cup \{0\}} x_{ij} = 1, \quad \forall j \in N \tag{8}$$

$$\sum_{j \in N} x_{0j} \leq m \tag{9}$$

$$\sum_{i \in B_j \cup \{0\}} x_{ij} = \sum_{i \in A_j \cup \{n+1\}} x_{ji}, \quad \forall j \in N \tag{10}$$

$$C_j = p_j x_{0j} + \sum_{i \in B_j} (C_i + p_j)\, x_{ij}, \quad \forall j \in N \tag{11}$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j \in N \tag{12}$$

where constraints (9) represent that there are at most m machines available. The other constraints have similar meanings to those in the preceding formulation **IP1**.

2.2 Dantzig-Wolfe Decomposition

In this subsection, we decompose the integer programming formulations **IP1** and **IP2** given in Section 2.1 into a master problem with a set partitioning formulation (Section 2.2.1) and some single machine subproblems (Section 2.2.3). We then use the column generation approach to solve the linear relaxation of the set partitioning formulation (Section 2.2.2).

2.2.1 The Set Partitioning Master Problem

Applying Dantzig-Wolfe decomposition [10], we decompose the formulation **IP1** into a master problem consisting of (1), (2), and (3), and m subproblems with feasible regions defined by (4), (5), and (6). It is easy to see that, for any fixed k, any point

$\left(x_{ij}^k : i,j \in N\right)$ in the feasible region of the k-th subproblem, that is, the region given by constraints (4) and (6) with the fixed k, together with C_j's defined in (5), corresponds to a feasible partial schedule on machine k. We will see in Section 2.2.3 that each of the m subproblems is a single machine scheduling problem.

Let Ω^k denote the set of all feasible partial schedules on machine k. Let f_s^k be the total cost of schedule $s \in \Omega^k$. For each job $j \in N$, let $a_{js}^k = 1$ if schedule $s \in \Omega^k$ covers job j, and 0 otherwise. Define $0-1$ variables, for $k \in M$ and $s \in \Omega^k$:

$$y_s^k = \begin{cases} 1 & \text{if schedule } s \in \Omega^k \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

Then the master problem can be formulated as the following set partitioning problem (**SP1**).

SP1:

$$\min \sum_{k \in M} \sum_{s \in \Omega^k} f_s^k y_s^k \tag{13}$$

subject to

$$\sum_{k \in M} \sum_{s \in \Omega^k} a_{js}^k y_s^k = 1, \quad \forall j \in N \tag{14}$$

$$\sum_{s \in \Omega^k} y_s^k \leq 1, \quad \forall k \in M \tag{15}$$

$$y_s^k \in \{0,1\}, \quad \forall s \in \Omega^k, k \in M \tag{16}$$

Constraints (14) and (15) correspond to the original constraints (2) and (3) and mean that each job is covered by exactly one feasible partial schedule and each machine is occupied by at most one feasible partial schedule, respectively.

For the problem $P||\sum f_j(C_j)$, this master problem can be simplified, if we apply Dantzig-Wolfe decomposition to the formulation **IP2**. Let Ω denote the set of all feasible partial schedules on a single machine. For any $s \in \Omega$, define f_s and a_{js} similarly to f_s^k and a_{js}^k respectively. Define variable $y_s = 1$ if schedule $s \in \Omega$ is used and 0 otherwise. Then the set partitioning master problem (**SP2**) for the problem $P||\sum w_j C_j$ is

SP2:

$$\min \sum_{s \in \Omega} f_s y_s \tag{17}$$

subject to

$$\sum_{s \in \Omega} a_{js} y_s = 1, \quad \forall j \in N \tag{18}$$

$$\sum_{s \in \Omega} y_s \leq m \tag{19}$$

$$y_s \in \{0,1\}, \quad \forall s \in \Omega \tag{20}$$

where constraint (18) has the same meaning as (14) in **SP1** and constraint (19) is equivalent to (9) in **IP2**.

Since, the formulations **SP1** and **IP1** (**SP2** and **IP2**) are both valid formulations for the same general PMAC problem, the solution value of the formulation **SP1** (**SP2**) must be equal to that of the formulation **IP1** (**IP2**), respectively. But, **SP1** and **SP2** are not merely reformulations of **IP1** and **IP2**. The difference lies in their linear relaxation problems. Relaxing the integrality constraints (6), (12), (16) and (20), we get the linear relaxation problems, denoted by **LIP1**, **LIP2**, **LSP1** and **LSP2**, respectively corresponding to the integer problems **IP1**, **IP2**, **SP1** and **SP2**. The solution value of **LSP1** is usually greater than that of **LIP1** because the set Ω^k is smaller than the set of points $(x_{ij}^k : i, j \in N)$ in the region given by (4) and the linear relaxation of (6). The same relation is true for the formulations **LSP2** and **LIP2**. This means that the set partitioning formulations can yield tighter lower bounds than the original integer programming formulations. Furthermore, the formulations **LIP1** and **LIP2** are not linear programs and may not be easy to solve because they involve nonlinear constraints (5) and (11), while the formulations **LSP1** and **LSP2** are linear programs and can be solved easily using the column generation approach described later. Therefore, in our branch and bound algorithm, we mainly work on the SP formulations **SP1** and **SP2**, instead of the original IP formulations **IP1** and **IP2**. However, in the algorithm, we branch on the variables in the IP formulations, instead of the variables in the SP formulations, to make the subproblems more tractable.

2.2.2 Column Generation Procedure for Solving LSP1 and LSP2

As we mentioned earlier, each column in **LSP1** and **LSP2** represents a feasible partial schedule on a single machine. As the number of feasible partial schedules on a machine, i.e. $|\Omega^k|$ or $|\Omega|$, can be extremely large, it is impossible to explicitly list all the columns when solving **LSP1** and **LSP2**. So we use the column generation approach (see, e.g. Lasdon [20]) to generate necessary columns only in order to solve **LSP1** and **LSP2** efficiently.

Column generation approach has been successfully applied to many large scale optimization problems, such as cutting stock (Gilmore and Gomory [16], Vance, Barnhart, Johnson and Nemhauser [32]), vehicle routing (Desrochers, Desrosiers and Solomon [12]), air crew scheduling (Lavoie, Minoux and Odier [21], Vance [31]), lot sizing and scheduling (Cattrysse, Salomon, Kuik and Van Wassenhove [6]), and graph coloring (Mehrotra and Trick [24]).

The column generation procedure consists of the following four major steps:

- solving a restricted master problem of **LSP1** or **LSP2**, i.e, the problem **LSP1** or **LSP2** with a restricted number of columns;
- using the dual variable values of the solved restricted master problem to update cost coefficients of the subproblems;
- solving single machine subproblems; and
- getting new columns with negative reduced costs based on the subproblem solutions and adding the new columns to the restricted master problem.

These steps are repeated until no column with negative reduced cost can be generated, upon which we will have solved the problem **LSP1** or **LSP2** to optimality.

The restricted master problem of **LSP1** or **LSP2** is a linear program which can be efficiently solved using a linear programming solver. The efficiency of the column generation procedure mainly depends on how fast a subproblem can be solved. Actually,

it is crucial to find an appropriate formulation (**IP1**, **IP2**, or their modified version) for a particular problem so that the resulting subproblems have desired structures and can be solved efficiently.

It is worth noting some implementation strategies that may be used in the column generation algorithm. First, in order to have an initial feasible restricted master problem of **LSP1** or **LSP2**, we can either use a heuristic to generate a feasible schedule and use the resulting m single machine schedules as the columns in the initial restricted master problem, or slightly modify the formulation **LSP1** or **LSP2** by adding an artificial variable to the left-hand side of each of the n equality constraints and a sufficiently large linear function of each artificial variable to the objective function. Second, if more than one columns with a negative reduced cost are available from the subproblem solution, then it may be beneficial to add multiple such columns, instead of only the column with the most negative reduced cost, to the restricted master problem. Our computational experiments (described in Section 5) seem to indicate that the implementation which adds five to ten columns (if available) in each iteration is more efficient than the implementation which adds only one column or adds more than twenty columns.

2.2.3 Single Machine Subproblems

The goal of solving subproblems is to find the column with the minimum reduced cost to be added to the restricted master problem when solving **LSP1** and **LSP2**. If the minimum reduced cost is nonnegative, then we can terminate the column generation procedure and the problem is solved. In the restricted master problem of **LSP1**, let π_j denote the dual variable value corresponding to job j, for each $j \in N$, in constraint (14), and σ_k denote the dual variable value corresponding to machine k, for each $k \in M$, in constraint (15). Then the reduced cost r_s^k of the column corresponding to $s \in \Omega^k$ is given by:

$$r_s^k = f_s^k - \sum_{j \in N} a_{js}^k \pi_j - \sigma_k \tag{21}$$

Similarly, in the restricted master problem of **LSP2**, let π_j denote the dual variable value corresponding to job j, for each $j \in N$, in constraint (18), and σ denote that corresponding to the constraint (19). Then the the reduced cost r_s of the column corresponding to $s \in \Omega$ is as follows:

$$r_s = f_s - \sum_{j \in N} a_{js}\pi_j - \sigma \tag{22}$$

Hence, when solving **LSP1**, we need to solve m subproblems, one for each machine, in order to find the column with the minimum reduced cost. The k-th subproblem is to find a feasible partial schedule $s \in \Omega^k$ on machine k such that its reduced cost r_s^k is minimized. By contrast, when solving **LSP2**, since all the machines are identical, we only need to solve one subproblem which is to find a feasible partial schedule $s \in \Omega$ on a single machine such that its reduced cost r_s is minimized.

By the reduced cost formula (21) or (22), more precisely, a *single machine subproblem* on some machine is to find a subset of jobs of N and a schedule for these jobs on that machine which satisfies the job ordering restriction of the problem such that the total cost of the jobs in the schedule (i.e. the quantity f_s^k or f_s) minus the total dual variable value of these jobs (i.e. the quantity $\sum_{j \in N} a_{js}^k \pi_j$ or $\sum_{j \in N} a_{js}\pi_j$) is minimized. Here the dual variable values corresponding to the machine, i.e. σ_k and σ, are ignored since they are common to all the feasible partial schedules on that machine. We will see later that single machine subproblems for the total weighted completion time problem and the weighted number of tardy jobs problem are all ordinarily NP-hard and can be solved by pseudopolynomial dynamic programming algorithms.

2.3 Branch and Bound Algorithm

In this section, we describe a branch and bound (b&b) exact solution algorithm for the PMAC problems. Special attention is given to the branching strategy used in the algorithm.

For solving the problems $Q||\sum f_j(C_j)$ and $R||\sum f_j(C_j)$, the b&b algorithm is based on the formulations **IP1**, **SP1**, and their linear relaxations **LIP1** and **LSP1**. While, for

solving the problem $P||\sum f_j(C_j)$, the b&b algorithm uses the corresponding simplified formulations **IP2**, **SP2**, **LIP2**, and **LSP2**. In the b&b tree, each b&b node is a linear relaxation problem **LSP1** or **LSP2** with some additional constraint imposed by the branching rule (described later). These linear relaxation problems are solved by the column generation procedure described earlier.

Usually, in a branch and bound algorithm, two classes of decisions need to be made (see, e.g. Nemhauser and Wolsey [25]) throughout the algorithm. One is called *node selection*, that is, to select an active node in the b&b tree to be explored (solved). The other is called *branching variable selection*, that is, to select a fractional variable to branch on. The node selection strategy we use in our algorithm combines the rule *depth-first-search* (also known as *last-in-first-out* (LIFO)) and the rule *best-lower-bound*. If the current b&b node is not pruned, then the depth-first-search rule is applied such that one of the two son nodes of the current b&b node is selected as the next node to be solved. If the current b&b node is pruned, then the best-lower-bound rule is applied such that an active node in the b&b tree with the smallest lower bound is selected as the next node to be explored.

For solving our problem **SP1** (i.e. the problems $Q||\sum f_j(C_j)$ and $R||\sum f_j(C_j)$) or **SP2** (i.e. the problem $P||\sum f_j(C_j)$), traditional branching on the y-variables in the problem may cause trouble along a branch where a variable has been set to zero. Recall that y_s^k in **SP1** (y_s in **SP2**) represents a feasible partial schedule on some machine k generated by solving a single machine subproblem. The branching $y_s^k = 0$ ($y_s = 0$) means that this partial schedule is excluded and hence no such schedule can be generated in subsequent subproblems on that machine. However, it is not an easy task to exclude a schedule when solving a single machine subproblem.

Fortunately, there is a simple remedy to this difficulty. Instead of branching on the y-variables in the set partitioning formulation **SP1** (**SP2**), we branch on x-variables in the original formulation **IP1** (**IP2**). This branching variable selection strategy, that is, branching on variables in the original formulation, has been proved successful in many branch and bound algorithms for problems that can be reformulated, usually by

Dantzig-Wolfe decomposition, into a master problem and one or some subproblems. See Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance [3] for a class of such problems.

Observing the relation between the set partitioning formulation **SP1** (**SP2**) and the original formulation **IP1** (**IP2**), we can see that for any feasible solution $(y_s^k, s \in \Omega^k, k \in M)$ to **LSP1** $((y_s, s \in \Omega)$ to **LSP2**), there is a corresponding feasible solution $(x_{ij}^k, i, j \in N, k \in M)$ to the problem **LIP1** $((x_{ij}, i, j \in N)$ to the problem **LIP2**), such that

$$x_{ij}^k = \sum_{s \in \Omega^k} e_{ij}^s y_s^k \tag{23}$$

and

$$x_{ij} = \sum_{s \in \Omega} e_{ij}^s y_s \tag{24}$$

where e_{ij}^s is 1 if jobs i and j are both contained in schedule s and job j follows job i immediately in s, and 0 otherwise.

Obviously, if the y-variable solution, i.e. $(y_s^k, s \in \Omega^k, k \in M)$ or $(y_s, s \in \Omega)$, is integral, then the corresponding x-variable solution given by (23) and (24) is integral as well. Actually, the reverse is also true. This is proved in Lemma A1 in the Appendix.

Once we have explored a b&b node (i.e. a problem **LSP1** or **LSP2**), if the solution $(y_s^k, s \in \Omega^k, k \in M)$ or $(y_s, s \in \Omega)$ is fractional and the integer part of its solution value is less than the upper bound of the b&b tree, then the corresponding x-variable solution is computed by (23) or (24). By Lemma A1, this x-variable solution must be fractional. Based on this x-variable solution, an appropriate fractional x-variable is then selected to branch on next. For ease of presentation, we distinguish the two problems: **SP1** and **SP2**, and describe the branching strategy for each of them separately in the following paragraphs.

In the case of **SP2**, a pair of jobs (h, l) is selected such that $x_{hl} = \arg\min_{x_{ij}} \{|x_{ij} - 0.5|\}$, i.e. the pair (h, l) has the value x_{hl} with the maximum integer infeasibility. Two son nodes are then created, one along the branch with x_{hl} fixed as 0 and the other along the branch with x_{hl} fixed as 1. If x_{hl} is fixed as 0, then the initial restricted master problem of the corresponding son node consists of all the columns of its father node

except the ones in which job l is scheduled immediately after job h if $h \neq 0$ or job l is scheduled first on a machine if $h = 0$. At the same time, the job ordering restriction is updated such that $B_l := B_l \setminus \{h\}$, which guarantees that no schedule will be generated where job l is processed immediately after job h if $h \neq 0$ or where job l is processed first on a machine. If x_{hl} is fixed as 1, then the initial restricted master problem of the corresponding son node consists of all the columns of its father node except the ones in which job l is scheduled immediately after a job other than h and the ones in which a job other than l is scheduled immediately after job h. The job ordering restriction is also updated accordingly such that $B_l := \{h\}$, which ensures that any schedule that contains job l processes job h immediately before l.

In the case of **SP1**, the branching variable selection strategy is slightly different from the case of **SP2**. First define

$$q_{ij} = \sum_{k \in M} x_{ij}^k \tag{25}$$

The branching variable selection consists of two stages. In the first stage, branching is based on values of the variables q_{ij}. If some q_{ij}'s are fractional, then select a pair of jobs (h, l) with the value q_{hl} having the maximum integer infeasibility, i.e. $q_{hl} = \arg\min_{q_{ij}}\{|q_{ij} - 0.5|\}$. Two son nodes are created similarly as in the case of **SP2**. If q_{hl} is fixed as 0, then x_{hl}^k, for each $k \in M$, is fixed as 0. If q_{hl} is fixed as 1, then all x_{il}^k with $i \neq h$, and x_{hj}^k with $j \neq l$ are fixed as 0. In the son node with q_{hl} fixed as 0, the restricted master problem is constructed in the same way as the son node with $x_{hl} = 0$ in the case of **SP2**. The job ordering restriction for this son node is updated as follows: $B_l^k := B_l^k \setminus \{h\}$ for each $k \in M$. In the son node with q_{hl} fixed as 1, the restricted master problem is also constructed in the same way as the son node with $x_{hl} = 1$ in the case of **SP2**. The job ordering restriction is updated by letting: $B_l^k := B_l^k \bigcap \{h\}$ for each $k \in M$.

When every q_{ij} is integral, it is possible for some x_{ij}^k's to be fractional. In such a case, we use the second stage branching strategy which selects a branching variable x_{hl}^v such that $x_{hl}^v = \arg\min_{x_{ij}^k}\{|x_{ij}^k - 0.5|\}$. Similarly, two son nodes are created, one along

the branch with x_{hl}^v fixed as 0 and the other along the branch with x_{hl}^v fixed as 1. In the first son node, the restricted master problem consists of all the columns of its father node except the ones that schedule job l immediately after job h on machine v; and the job ordering restriction is updated: $B_l^v = B_l^v \setminus \{h\}$. In the second son node, the restricted master problem consists of all the columns of its father node except the ones that contain at least one of the jobs h and l on a machine other than machine v and the ones that schedule job l immediately after a job other than job h or schedule a job other than h immediately before job l; and the job ordering restriction is updated as follows: $B_l^v = \{h\}$, and $B_l^k = \phi, B_j^k = B_j^k \setminus \{h\}$ for $k \in M \setminus \{v\}, j \in N \setminus \{l\}$.

3 The Total Weighted Completion Time Problem

In this section, we apply the decomposition approach to solve all the three cases of the total weighted completion time problem, i.e. problems $P||\sum w_jC_j$, $Q||\sum w_jC_j$ and $R||\sum w_jC_j$. This application is quite straightforward. The entire decomposition approach, including all the formulations (i.e. **IP1**, **IP2**, **SP1**, and **SP2**), and all the solution strategies (i.e. the column generation procedure, and the branching strategy), is directly applied to the total weighted completion time problem. However, some problem dependent parameters and properties and single machine subproblems need to be specified concretely.

This section is organized as follows. In Section 3.1, we point out some problem dependent properties of which we can take advantage. In Section 3.2, we show that the associated single machine subproblem is NP-hard. Then in Section 3.3, we propose a pseudo-polynomial dynamic programming algorithm for solving the subproblem.

3.1 Problem Dependent Properties

First notice that by the well-known Smith's rule [29], in any optimal schedule for the problems $P||\sum w_jC_j$, $Q||\sum w_jC_j$, and $R||\sum w_jC_j$, jobs on each machine must form the shortest weighted processing time ($SWPT$) order, that is, if jobs i and j are both

processed on machine k and i precedes j, then $p_{ik}/w_i \leq p_{jk}/w_j$. So we need only consider those schedules where the partial schedule on each machine forms the $SWPT$ order. Hence, here, a feasible partial schedule (defined in Section 2) is actually a partial schedule in which jobs form the $SWPT$ order.

Let $SWPT^k$ denote the $SWPT$ order formed by all the jobs N on machine k. Without loss of generality, we assume that if two jobs i and j have the same ratios: $p_{ik}/w_i = p_{jk}/w_j$, then the one with the smaller index precedes the other one in the sequence $SWPT^k$. By this assumption, the sequence $SWPT^k$ is unique for all $k \in M$. Also, it is easy to see that $SWPT^1 = SWPT^2 = ... = SWPT^m$ for the problems $P||\sum w_jC_j$ and $Q||\sum w_jC_j$. The sets A_j^k, B_j^k, A_j and B_j used in the formulations **IP1** and **IP2** are actually as follows:

$A_j^k = \{i \in N \mid i \text{ succeeds } j \text{ in the sequence } SWPT^k\}$

$B_j^k = \{i \in N \mid i \text{ precedes } j \text{ in the sequence } SWPT^k\}$

$A_j = \{i \in N \mid i \text{ succeeds } j \text{ in the } SWPT \text{ order of } N\}$

$B_j = \{i \in N \mid i \text{ precedes } j \text{ in the } SWPT \text{ order of } N\}$.

Similarly, the sets Ω^k and Ω used in the formulations **SP1** and **SP2** are actually as follows:

$\Omega^k = \{$ all possible partial schedules on machine k that satisfy the $SWPT$ rule $\}$

$\Omega = \{$ all possible partial schedules on a single machine that satisfy the $SWPT$ rule $\}$.

Finally, we point out that, by the definitions of sets Ω^k and Ω given here and the observation made in Section 2.2.3, a single machine subproblem on some machine is actually to find a subset of jobs such that the total weighted completion time of the jobs under the $SWPT$ order on that machine minus the total dual variable value corresponding to the jobs is minimized. We show the NP-hardness of this problem and give a dynamic programming algorithm for it in the following subsections.

3.2 NP-hardness proof of the Subproblem

As mentioned in the preceding subsection, the single machine subproblem can be stated as follows. Given a set of jobs $N = \{1, 2, ..., n\}$, a processing time $p_i \in Z^+$, a weight $w_i \in Z^+$, and a dual variable value $\pi_i \in Z$, for each $i \in N$, the objective is to find a subset $H \subseteq N$ such that the total weighted completion time of the jobs of H under the $SWPT$ sequence minus the total dual variable values corresponding to the jobs of H is minimized. Note that in the case of $Q||\sum w_j C_j$ or $R||\sum w_j C_j$, the processing time p_i of job i here is replaced by p_{ik}, the processing time of i on some machine k where the subproblem is being considered.

We show the NP-hardness of the single machine subproblem by a reduction from the PARTITION problem, a known NP-complete problem (Garey and Johnson [15]). An instance I of the PARTITION problem can be stated as follows:

Given $n + 1$ integers, a_1, a_2, ..., a_n, and A, such that $2A = \sum_{i=1}^{n} a_i$, does there exist a subset $S \subseteq T = \{1, 2, ..., n\}$ such that $\sum_{j \in S} a_j = \sum_{j \in T \setminus S} a_j = A$?

Given such an instance I of the PARTITION problem, we construct a corresponding instance II for the associated decision problem of the single machine subproblem as follows:

- Jobs: $N = T = \{1, 2, ..., n\}$;
- Processing times: $p_j = a_j$, for all $j \in N$;
- Weights: $w_j = 2a_j$, for all $j \in N$;
- Dual variable values: $\pi_j = 2a_j A + a_j^2$;
- Threshold: $Y = -A^2$.

Clearly, instance II can be constructed from instance I in polynomial time.

Theorem 1 The single machine subproblem is NP-hard in the ordinary sense.

Proof: Obviously, this problem is in the NP class. We prove its NP-hardness by

showing the following statement: there is a subset $H \subseteq N$ for instance II such that the total weighted completion time of the jobs in H under the $SWPT$ sequence minus the total dual variable value corresponding to the jobs in H is no greater than Y, if and only if there is a subset $S \subseteq T = \{1, 2, ..., n\}$ for instance I such that $\sum_{j \in S} a_j = \sum_{j \in T \setminus S} a_j = A$.

Then the existence of a pseudopolynomial dynamic programming algorithm (described in the next subsection) implies that this problem is NP-hard in the ordinary sense.

Given any subset $J \subseteq N$, since $w_j = 2p_j$ for any $j \in N$, it is easy to verify that the total weighted completion time of the jobs of J under any sequence is:

$$f(J) = 2\sum_{j \in J} a_j^2 + 2 \sum_{i<j; i,j \in J} a_i a_j \tag{26}$$

On the other hand, the total dual variable value corresponding to the jobs in J is:

$$g(J) = \sum_{j \in J} \pi_j = 2A \sum_{j \in J} a_j + \sum_{j \in J} a_j^2 \tag{27}$$

Then the objective value of any sequence of the jobs in J is:

$$\begin{aligned} F(J) &= f(J) - g(J) = (\sum_{j \in J} a_j)^2 - 2A \sum_{j \in J} a_j \\ &= (\sum_{j \in J} a_j)^2 - (\sum_{j \in J} a_j + \sum_{j \in N \setminus J} a_j)(\sum_{j \in J} a_j) \\ &= -(\sum_{j \in N \setminus J} a_j)(\sum_{j \in J} a_j) \\ &\geq -A^2 \end{aligned} \tag{28}$$

and the last equality holds if and only if $\sum_{j \in N \setminus J} a_j = \sum_{j \in J} a_j$.

"If part". If such a subset S exists for instance I, let $H = S$. By the above observation, the objective value of any sequence of the jobs in H is $F(H) = -A^2 = Y$ since $\sum_{j \in N \setminus H} a_j = \sum_{j \in H} a_j$. Hence the subset H is a solution to instance II.

"Only if part". If such a subset H exists for instance II, i.e. $F(H) \leq Y = -A^2$. By (28), $F(H) = -A^2$, implying that $\sum_{j \in N \setminus J} a_j = \sum_{j \in J} a_j$. Hence the subset $S = H$ is a solution to instance I. $\square$

3.3 Dynamic Programming Algorithm for the Subproblem

In this subsection, we propose a pseudopolynomial dynamic programming algorithm for solving the subproblem.

First reindex the jobs such that $p_1/w_1 \leq p_2/w_2 \leq ... \leq p_n/w_n$. Then $(1, 2, ..., n)$ forms the $SWPT$ order. Let $P = \sum_{i=1}^{n} p_i$ be the total processing time of the jobs. Define

$$B_j = \{i \in N \mid \ i \text{ precedes } j \text{ in the } SWPT \text{ order}\} = \{1, 2, ..., j-1\}$$

By the job ordering restriction (i.e. jobs one each machine must form the $SWPT$ order), only the jobs in B_j can be scheduled before job j on a machine.

It is easy to show that the following dynamic programming algorithm, consisting of procedures (**dp11**)-(**dp14**), solves the single machine subproblem to optimality.

(dp11) Let $F(j, t)$ denote the minimum objective value (total weighted completion time minus total dual variable value) in a partial schedule consisting of a subset of jobs of $\{1, 2, ..., j\}$, provided that the partial schedule forms the $SWPT$ order and that job j is the last job and completed at time t in the partial schedule.

(dp12) Initial values:

$$F(j, t) = \infty \quad \text{for } t < 0,\ j = 0, ..., n$$

$$F(0, t) = 0 \quad \text{for } t \geq 0$$

(dp13) Recursive relation:

For $j = 1, ..., n$; $t = 0, ..., P$:

$$F(j, t) = \min_{i \in B_j \cup \{0\}} \{F(i, t - p_j) + tw_j - \pi_j\} \tag{29}$$

(dp14) The problem is solved by computing

$$\min_{j \in N, 0 \leq t \leq P} \{F(j, t)\} \tag{30}$$

The worst case complexity of the algorithm is bounded by $O(n^2P)$, since there are a total of nP states in the dynamic program and it takes no more than $O(n)$ time to compute the value for a state.

It is worth noting that this is not the fastest possible algorithm, in terms of the worst case complexity, for the subproblem. A similar but faster dynamic programming algorithm can be given as follows. Let $F(j,t)$ be the minimum objective value of a partial schedule that contains a subset of jobs of $\{1,2,...,j\}$, satisfies the $SWPT$ rule, and is completed at time t. After executing the same initialization step as in the procedure **(dp12)**, we use the following new recursive relation:

$$F(j,t) = \min\{F(j-1,t-p_j) + tw_j - \pi_j, \quad F(j-1,t)\} \tag{31}$$

Then the problem is solved by computing $\min_{0\le t\le P} F(n,t)$. This algorithm has the worst case complexity $O(nP)$ since it only takes constant time to compute the value for each of the total nP states.

Unfortunately, the latter DP algorithm will no longer work after a branching procedure is performed. In the recursion (31), it is implicitly assumed that any job i, for $i = 1,...,j-1$, is eligible to be scheduled immediately before job j. In our branch and bound algorithm, every branching procedure imposes a restriction on which jobs can be scheduled immediately before some job j (i.e. changes the set B_j) and hence the recursion (31) is no longer valid after branching.

However, branching has no effect on the former DP algorithm because it merely updates sets B_j, which has been explicitly taken into account in the former algorithm (see (29)). So in our branch and bound algorithm we use the former DP algorithm instead of the latter one.

4 The Weighted Number of Tardy Jobs Problem

In this section, we apply the decomposition approach to solve all the three cases of the weighted number of tardy jobs problem, i.e. problems $P||\sum w_jU_j$, $Q||\sum w_jU_j$ and

$R||\sum w_jU_j$.

If we apply all the formulations (i.e. **IP1**, **IP2**, **SP1**, and **SP2**) directly to the weighted number of tardy jobs problem, then by the reduced cost formula (21), the resulting subproblem on a machine is to find a partial schedule such that the total weight of tardy jobs in the schedule minus the total dual variable value of the jobs in the schedule is minimized. If there is no job ordering restriction, this subproblem is no more difficult than the well-know single machine weighted number of tardy jobs problem, which can be solved quite efficiently by the pseudo-polynomial dynamic programming algorithm of Lawler and Moore [23]. However, after a branching procedure is performed, some ordering restriction will be imposed on jobs, and no pseudopolynomial algorithm, we believe, can solve the resulting subproblem.

Hence, to get efficient algorithms (particularly, an algorithm for the subproblem), we need to modify the formulations **IP1**, **IP2**, **SP1**, and **SP2**. As we will see soon, the resulting subproblem based on the modified formulations can be solved by a pseudopolynomial dynamic programming algorithm, and fortunately, branching will not affect the capability of the algorithm. Despite the modification, the general solution strategies described in Section 2 (i.e. the column generation procedure, and the branching strategy) can be directly applied.

The organization of this section is as follows. In Section 4.1, we modify the formulations **IP1** and **SP1** (**IP2** and **SP2** can be modified similarly) and point out a result that can be used in the branching procedure. In Section 4.2, we present a pseudo-polynomial dynamic programming algorithm for solving the subproblem.

4.1 Modified Formulations

In this subsection, we modify the formulations **IP1** and **SP1** (for the problems $Q||\sum w_jU_j$ and $R||\sum w_jU_j$). The other two formulations: **IP2** and **SP2** (for the problem $P||\sum w_jU_j$) can be modified similarly, and hence we do not give any detail on them.

It is clear that by the well-known result of Lawler and Moore [23], for each of the

problems $P||\sum w_jU_j$, $Q||\sum w_jU_j$ and $R||\sum w_jU_j$, there exists an optimal schedule where the partial schedule on each machine satisfies the following properties:

Property 1: On-time jobs form the EDD (earliest due date first) order, that is, the nondecreasing order of jobs' due dates;

Property 2: Tardy jobs are in an arbitrary order;

Property 3: On-time jobs are scheduled earlier than any of its tardy jobs.

Define an *on-time EDD partial schedule* on a single machine to be a partial schedule on that machine in which all the jobs are on-time and form the EDD order. By Properties 1, 2 and 3, we can conclude that solving the problems $P||\sum w_jU_j$, $Q||\sum w_jU_j$ and $R||\sum w_jU_j$ is equivalent to finding an on-time EDD partial schedule for each machine such that the total weight of the jobs that are not covered in these partial schedules is minimized. Those uncovered jobs can be considered tardy and scheduled arbitrarily following the on-time jobs. Hence, the concept of "on-time EDD partial schedule" defined here is actually equivalent to that of "feasible partial schedule" defined in Section 2.

Based on the above observation, we modify the formulations **IP1** in the following. Define the following sets and $0-1$ variables:

$A_j = \{i \in N \mid i \text{ succeeds } j \text{ in the } EDD \text{ order of } N\}$

$B_j = \{i \in N \mid i \text{ precedes } j \text{ in the } EDD \text{ order of } N\}$

$z_j = 1$ if job j is scheduled tardy on some machine; 0 otherwise.

$x_{ij}^k = 1$ if job i and j are both scheduled on-time on machine k and i is processed immediately before job j; 0 otherwise.

$x_{0j}^k = 1$ if job j is scheduled first and on-time on machine k; 0 otherwise.

$x_{j,n+1}^k = 1$ if job j is scheduled last and on-time on machine k; 0 otherwise.

Then the following integer programming formulation (**IP1′**) solves the problems $P||\sum w_jU_j$, $Q||\sum w_jU_j$ and $R||\sum w_jU_j$.

IP1′:

$$\min \sum_{j\in N} w_j z_j \tag{32}$$

subject to

$$\sum_{k\in M}\sum_{i\in B_j\cup\{0\}} x_{ij}^k + z_j = 1, \quad \forall j \in N \tag{33}$$

$$\sum_{j\in N} x_{0j}^k \leq 1, \quad \forall k \in M \tag{34}$$

$$\sum_{i\in B_j\cup\{0\}} x_{ij}^k = \sum_{i\in A_j\cup\{n+1\}} x_{ji}^k, \quad \forall k \in M, j \in N \tag{35}$$

$$C_j = \sum_{k\in M}\left(p_{jk}x_{0j}^k + \sum_{i\in B_j}(C_i + p_{jk})x_{ij}^k\right), \quad \forall j \in N \tag{36}$$

$$0 \leq C_j \leq d_j, \quad \forall j \in N \tag{37}$$

$$x_{ij}^k \in \{0,1\}, \quad \forall i,j \in N, k \in M \tag{38}$$

$$z_j \in \{0,1\}, \quad \forall j \in N \tag{39}$$

The objective function (32) seeks to minimize the weighted number of tardy jobs. Constraint (33) ensures that each job j is covered by at most one on-time EDD partial schedule. If $z_j = 0$ then job j is covered by an on-time EDD partial schedule; otherwise it is tardy. Each tardy job j has all x_{ij}^k's$= 0$ and $C_j = 0$. Constraint (34) guarantees that each machine is utilized at most once. Constraint (36) defines the completion time of each on-time job. Constraint (37) means that each on-time job must be completed before or at its due date. Constraints (35), (36), (37) and (38) guarantee the feasibility of the on-time EDD partial schedule on each machine.

Applying Dantzig-Wolfe decomposition to the modified formulations **IP1′**, we can get the following set partitioning formulation **SP1′**.

SP1′:

$$\min \sum_{j\in N} w_j z_j \tag{40}$$

subject to

$$\sum_{k\in M}\sum_{s\in\Omega^k} a_{js}^k y_s^k + z_j = 1, \quad \forall j \in N \tag{41}$$

$$\sum_{s \in \Omega^k} y_s^k \leq 1, \quad \forall k \in M \tag{42}$$

$$y_s^k \in \{0,1\}, \quad \forall s \in \Omega^k, k \in M \tag{43}$$

$$z_j \in \{0,1\}, \quad \forall j \in N \tag{44}$$

Here Ω^k denotes the set of all possible on-time EDD partial schedules on machine k and all other parameters and variables are defined similarly as in **SP1**.

Similarly, the column generation procedure can be applied to solve the linear relaxation problem **LSP1′** of the integer problem **SP1′**. Let π_j denote the dual variable value corresponding to job j, for each $j \in N$, in constraint (41). Then the reduced cost r_s^k of the column corresponding to $s \in \Omega^k$ is given by:

$$r_s^k = -\sum_{j \in N} a_{js}^k \pi_j - \sigma_k \tag{45}$$

There are m single machine subproblems. The single machine subproblem on machine k is to find an on-time EDD partial schedule $s \in \Omega^k$ on machine k such that its reduced cost r_s^k is minimized, that is, to find an on-time EDD partial schedule $s \in \Omega^k$ on machine k such that the total dual variable value corresponding to the jobs covered in s, i.e. the quantity $\sum_{j \in N} a_{js}^k \pi_j$, is maximized.

If we consider the dual variable value corresponding to a job as the weight of that job, then the single machine subproblem on a machine is exactly the single machine weighted number of tardy jobs problem. It is NP-hard in the ordinary sense (Karp [19]) and can be solved by the pseudopolynomial dynamic programming algorithm of Lawler and Moore [23]. However, we do not use Lawler and Moore's algorithm because as we will see in the next subsection, their algorithm is hard to implement inside a branch and bound algorithm. Instead, we propose a new dynamic programming algorithm for solving the subproblem. This algorithm is described in the next subsection.

Finally, we note that the result in Lemma A1 still holds with respect to the modified formulations given earlier for the weighted number of tardy jobs problem. Hence, the branching strategy suggested in Section 2 can be directly applied to all the three cases of the weighted number of tardy jobs problem. Furthermore, for the problems $Q||\sum w_j U_j$

and $R||\sum w_jU_j$, if every q_{ij} (defined in Section 2) resulted from solving **LSP1′** is integral, then, even if the x-variable solution is fractional, it is possible to construct an integer solution to the problem **LSP1′** based on the x-variable solution and q-values. This means that as long as q-values are integral, there is no need to conduct the second stage branching (described in Section 2). This is proved in Lemma A2 in the Appendix.

4.2 Dynamic Programming Algorithm for the Subproblem

First reindex the jobs such that $d_1 \leq d_2 \leq ... \leq d_n$. Then $(1, 2, ..., n)$ forms the EDD order. Let $B_j = \{1, 2, ..., j-1\}$. Then only jobs in B_j can be scheduled before job j in a feasible partial schedule on a machine. Let $P = \sum_{i=1}^{n} p_i$ be the total processing time of the jobs. Then it is easy to see that the following dynamic programming algorithm, consisting of procedures (**dp21**)-(**dp24**), solves the subproblem to optimality.

(dp21) Let $F(j,t)$ denote the maximum objective function value (total dual variable value) in an on-time EDD partial schedule consisting of a subset of jobs of $\{1, 2, ..., j\}$, provided that job j is the last job and is completed at time t in the partial schedule.

(dp22) Initial values:

$$F(j,t) = -\infty \quad \text{for } t < 0,\ j = 0, ..., n$$

$$F(0,t) = 0 \quad \text{for } t \geq 0$$

(dp23) Recursive relation:

For $j = 1, ..., n$; $t = 0, ..., P$:

$$F(j,t) = \max_{i \in B_j \cup \{0\}} \{\hat{F}(i,j,t)\}, \tag{46}$$

where

$$\hat{F}(i,j,t) = \begin{cases} F(i, t-p_j) + \pi_j, & \text{if } t \leq d_j \\ -\infty, & \text{otherwise} \end{cases} \tag{47}$$

(dp24) The problem is solved by computing

$$\max_{j \in N, 0 \leq t \leq P} \{F(j,t)\} \tag{48}$$

The worst case complexity of the algorithm is bounded by $O(n^2P)$, since there are a total of nP states in the dynamic program and it takes no more than $O(n)$ time to compute the value for a state.

Lawler and Moore's algorithm is faster than the above algorithm in terms of the worst case complexity. Their algorithm consists of similar procedures as in the above algorithm. The procedure **(dp22)** is not changed. In **(dp21)**, redefine $F(j,t)$ to be the maximum objective function value (total dual variable value) of an on-time EDD partial schedule that contains a subset of jobs of $\{1,2,...,j\}$ and is completed at time t. In **(dp23)**, a new recursive relation is used:

$$F(j,t) = \max\{F(j-1,t), \hat{F}(j,t)\}, \tag{49}$$

where

$$\hat{F}(j,t) = \begin{cases} F(j-1,t-p_j)+\pi_j, & \text{if } t \le d_j \\ -\infty, & \text{otherwise} \end{cases} \tag{50}$$

Then the problem is solved by computing $\max_{0\le t\le P} F(n,t)$. This algorithm has the worst case time complexity $O(nP)$ since it only takes constant time to compute the value for each of the total nP states.

Unfortunately, Lawler and Moore's algorithm will no longer work after a branching procedure is performed. In the recursion (49), it is implicitly assumed that any job i, for $i = 1,...,j-1$, is eligible to be scheduled immediately before job j. In our branch and bound algorithm, every branching procedure imposes a restriction on which jobs can be scheduled immediately before some job j (i.e. changes set B_j), and hence the recursion (49) is no longer valid after branching. This is the reason why we do not use Lawler and Moore's algorithm for solving the subproblem.

However, branching has no effect on the DP algorithm we proposed here because it merely updates sets B_j, which has been explicitly taken into account in the procedure (46) in our algorithm.

5 Computational Experiments

In this section, we describe our computational experiments for both the total weighted completion time problem and the total weighted number of tardy jobs problem. Our algorithms are all coded in C and tested on a Silicon Graphics Iris Workstation with MIPS R4400 Processor. Linear programs inside the branch and bound algorithms are solved by CPLEX, a commercial LP solver.

5.1 Configuration of Test Problems

For the total weighted completion time problem, the test problems are generated as follows:

- Number of machines (m). We use six different numbers: 2, 4, 8, 12, 16, 20.
- Number of jobs (n). We use six different numbers: 20, 30, 40, 60, 80, 100.
- Weights (w_j). The weight for each job is an integer number uniformly drawn from $[1, 100]$.
- Processing times (p_{ij}). For the problem $P||\sum w_jC_j$, the processing time p_j of each job j is an integer number uniformly drawn from $[1, 10]$. For the problem $Q||\sum w_jC_j$, the processing time of each job j on machine i, $p_{ij} = p_is_j$, where p_i and s_j are both integers from the uniform distribution $[1, 10]$. For the problem $R||\sum w_jC_j$, p_{ij} is an integer from the uniform distribution $[1, 30]$.

Note that the processing times and weights in the test problems we use here for the problem $P||\sum w_jC_j$ have the same distributions as the ones used in Belouadah and Potts [4].

For the total weighted number of tardy jobs problem, the test problems are generated as follows:

- Number of machines (m). We use five different numbers: 2, 4, 6, 8, 10.

- Number of jobs (n). We use nine different numbers: 20, 30, 40, 50, 60, 70, 80, 90, 100.

- Weights (w_j). The weight for each job is an integer number uniformly drawn from $[1, 100]$.

- Processing times (p_{ij}). For the problem $P||\sum w_jU_j$, processing times p_i are integers uniformly drawn from $[1, 100]$. For the problem $Q||\sum w_jU_j$, processing times $p_{ij} = p_is_j$ are generated by letting p_i and s_j be integers uniformly distributed in $[1, 40]$ and $[1, 5]$. For the problem $R||\sum w_jU_j$, processing times p_{ij} are integers uniformly distributed in $[1, 100]$.

- Due dates (d_j). We follow the model used in Ho and Chang [17, 18] to generate due dates. The due date of job i is set to be $\max\{\alpha_i, \beta_q\}$, where $\alpha_i = \min_{j \in M}\{p_{ij}\}$, and β_q is an integer uniformly distributed in $[1, 100r/q]$ with $r = n/m$ and q being a controllable parameter. The q value indicates the congestion level of the scheduling system. The larger the q, the more congested the system will be, and the more tardy jobs will result. We consider five possible values for q: 1, 2, 3, 4, and 5.

5.2 Computational Results

Tables 1, 2, 3, 4, 5, and 6 list the computational results for the problems $P||\sum w_jC_j$, $Q||\sum w_jC_j$, $R||\sum w_jC_j$, $P||\sum w_jU_j$, $Q||\sum w_jU_j$, and $R||\sum w_jU_j$, respectively. In these tables, the first two columns "m" and "n" represent, respectively, the number of machines and the number of jobs of a test problem. For each selected pair of values m and n, 20 problems are randomly generated based on the distributions of the associated parameters (for the problems $P||\sum w_jU_j$, $Q||\sum w_jU_j$, and $R||\sum w_jU_j$, exactly 4 randomly generated problems have the same q value). Each of the other entries in these tables represents an averaged performance value based on 20 problems with the same m and n. The column "LP-IP gap" represents the average gap in percentage between the linear relaxation solution value of the root b&b node and the integer solution value. This percentage reflects the tightness of the lower bound obtained by solving the linear

relaxation problem **LSP1** or **LSP2** (with respect to the original integer problem **SP1** or **SP2**). The column "problems solved at root node" indicates the number of problems solved at the root b&b node, i.e. without any branching, out of 20 problems. The column "b&b nodes explored" represents the average number of b&b nodes explored for solving the problem. Note that at least one b&b node (the root node) must be explored for solving any problem. The other two columns "columns generated" and "cpu time" represent, respectively, the average number of columns generated for solving the problem and the average cpu time (in seconds) consumed.

We note that among the 20 problems tested for each problem size, the worst-case of each above mentioned performance measure is always very close to the average, so we do not give the worst-case values of these performance measures.

From these tables, we can make the following observations:

- We can conclude that the lower bound given by the solution value of the linear relaxation problem **LSP1** (or **LSP2**) is extremely close to the solution value of the integer problem **SP1** (or **SP2**). In fact, in the case of the total weighted completion time problem, 52% of the 1380 problems we have tested here are solved at root node without any branching, and for each set of 20 test problems with the same size, the average gap between the lower bound and the integer solution value is less than 0.1%. In the case of the weighted number of tardy jobs problem, this average gap between is less than 0.8%. Due to this excellent lower bounding, an average of only nine b&b nodes are explored for solving an instance of the total weighted completion time problem in our experiments. Compared to the experiments by Belouadah and Potts [4] where at least thousands of b&b nodes have to be explored for solving a problem with a similar size, we can evidently conclude that our lower bound is much tighter than that in Belouadah and Potts [4] using Lagrangian relaxation. We note that, however, to solve a b&b node, our algorithm may take longer time than the algorithm by Belouadah and Potts.

- For the problem $P||\sum w_jC_j$, our algorithm is capable of solving the problems twice the size of the ones solved in the literature ([28], [4]) in a reasonable time. For the

other problems (on which no computational results are reported in the literature), our algorithm is capable of solving instances with a similar size in a similar time as well.

- Our algorithm works extremely well when the ratio of the number of jobs to the number of machines (n/m) is relatively small (say, less than eight). When this ratio is big (say, more than ten), the column generation procedure suffers from degeneracy (i.e. the dual information provided by solving a restricted master problem is not accurate enough and hence many columns generated are not very useful) and the algorithm becomes relatively slow even for problems with a small number of machines. For example, the average cpu time for a problem with $m = 8$ and $n = 60$ is much lower than that for a problem with $m = 4$ and $n = 60$.

6 Conclusion

We have developed a decomposition approach that can be applied to solve to optimality virtually every classical parallel machine scheduling problem with an additive criterion for which single machine subproblems admit an efficient solution. The applications of this decomposition approach to the total weighted completion time problem and the weighted number of tardy jobs problem have shown that the approach is promising and capable of solving large problems.

However, in order to apply this decomposition approach efficiently to a particular problem, it is important to give proper IP and SP formulations to the problem. As we have seen, for some problems, such as the total weighted completion time problem, the general IP and SP formulations described in Section 2 can be directly followed, while for some other problems, such as the weighted number of tardy jobs problem, it may be necessary to (slightly) modify these general formulations so that the resulting single machine subproblems can be more tractable.

Naturally, an interesting research topic is to apply the decomposition approach to other parallel machine scheduling problems with an additive criterion, such as the total (weighted) tardiness problem, the total earliness-tardiness penalty problem, etc. Actu-

ally, we have just successfully applied this approach to the latter problem with a common due date (Chen and Powell [8]).

Another possible research topic is to design fast heuristics for more complex parallel machine scheduling problems, e.g. the problems considered here with additional constraints, based on the idea of the decomposition approach. One possible way on this line of research is to use a heuristic, instead of an exact algorithm, to solve single machine subproblems.

Appendix

Lemma A1 Given a y-variable solution $(y_s^k, s \in \Omega^k, k \in M)$ to **LSP1** or $(y_s, s \in \Omega)$ to **LSP2**, if the corresponding x-variable solution given by (23) or (24) is integral, then the y-variable solution must be integral.

Proof: We prove the result by contradiction for the case of **LSP1**. For the case of **LSP2**, it can be similarly proved. Let

$$S^k = \left\{ s \in \Omega^k \mid y_s^k > 0 \right\}$$

Since columns generated in the column generation procedure are always distinct, all the schedules in S^k must be distinct. Suppose that there exists some machine k and some schedule $s \in S^k$ such that y_s^k is fractional, i.e. $0 < y_s^k < 1$. Without loss of generality, we assume that jobs i and j are two adjacent jobs in s and i precedes j, where i can be 0, meaning that job j is the first one in s. Thus, by (23), $x_{ij}^k > 0$, which indicates that $x_{ij}^k = 1$ by the fact that the x-variable solution is integral. Since $0 < y_s^k < 1$, there must exist another schedule $u \in S^k$ with $y_u^k > 0$ in which j follows i immediately. Since the two schedules s and u are distinct, there must exist three jobs a, b, c such that b follows a immediately in s while b follows c immediately in u. This implies that $x_{ab}^k > 0$ and $x_{cb}^k > 0$, which further implies that

$$x_{ab}^k = 1, \quad \text{and} \quad x_{cb}^k = 1$$

Hence $\sum_{i \in B_b^k \cup \{0\}} x_{ib}^k \geq 2$. On the other hand, the equations (23) and (14) implies that $\sum_{i \in B_b^k \cup \{0\}} x_{ib}^k \leq 1$. This leads to a contradiction. Therefore, the y-variable solution must be integral. $\Box$

Lemma A2 Given a y-variable solution $(y_s^k, s \in \Omega^k, k \in M)$ to the problem **LSP1′**, if the corresponding q-values given by (23) and (25) are all integral, then an integer solution to the problem **LSP1′** can be constructed from the known values of y and x.

Proof: First we need to give a definition that will be used later. Two single machine partial schedules are said to be *identical* if they contain the same subset of jobs and form the same sequence (but they may be on different machines).

Define

, $= \{s \in \Omega^k$, for some $k \in M \mid y_s^k > 0\}$;

$E = \{j \in N \mid j$ is covered by some schedule in , $\}$;

$H = \{j \in N \mid q_{0j} = 1\} = \{l_1, l_2, ..., l_{|H|}\}$;

, $_h = \{s \in$, $\mid e_{0h}^s = 1$, i.e. job h is scheduled first in schedule $s\}$, where e_{ij}^s is defined to be 1 if job j is scheduled immediately after job i in s.

Let $F(y)$ be the objective function value of **LSP1′** corresponding to the solution $(y_s^k, s \in \Omega^k, k \in M)$. By (23), (25) and (41), the following relations can be easily proved:

$$\sum_{i \in B_j \cup \{0\}} q_{ij} = \sum_{k \in M} \sum_{s \in \Omega^k} a_{js}^k y_s^k = \begin{cases} 1, & \text{if } j \in E \\ 0, & \text{otherwise} \end{cases} \tag{51}$$

and

$$F(y) = \sum_{j \in N \setminus E} w_j \tag{52}$$

Hence, each $q_{ij} \in \{0, 1\}$.

Similarly, by (23), (25) and (41), it is not difficult to show the following results:

(1) $|H| \leq m$

(2) All the partial schedules in , $_h$, for each $h \in H$, are identical, and hence , $_h$ contains only one partial schedule. Denote this schedule as ρ_h.

(3) These $|H|$ partial schedules ρ_1, ..., $\rho_{|H|}$ are disjoint and cover all the jobs in E.

Now we need to show that these $|H|$ partial schedules ρ_1, ..., $\rho_{|H|}$ form an integer solution to the problem **LSP1′**. The key issue we need to resolve here is to find what machine to process ρ_h, for each $h \in H$. Define, for each $h \in H$,

$$M_h = \{k \in M | \rho_h \in \Omega^k, y^k_{\rho_h} > 0\}.$$

The machines in M_h are in fact the candidates for processing schedule ρ_h. Given any subset $L \subseteq H$, let $M(L) = \bigcup_{l \in L} M_l$. It is easy to see that

$$\begin{aligned} \sum_{j \in L} q_{0j} &= \sum_{j \in L} \sum_{k \in M} \sum_{s \in \Omega^k} e^s_{0j} y^k_s \\ &= \sum_{j \in L} \sum_{k \in M_j} y^k_{\rho_j} \\ &\leq \sum_{s \in \Omega} \sum_{k \in M(L)} y^k_s \\ &\leq |M(L)| \end{aligned} \tag{53}$$

On the other hand, $\sum_{j \in L} q_{0j} = |L|$. Thus $|L| \leq |M(L)|$. Define a bipartite graph $G(X, Y)$ with vertex set $X = H \bigcup M$ and edge set $Y = \{(h, k) | h \in H, k \in M_h\}$. Then by the well-known Hall's Theorem (see, e.g. Bondy and Murty [5]), the bipartite graph G contains a matching that saturates every vertex in H. This means that for any $h \in H$, we can find a proper machine $m_h \in M$ to process schedule ρ_h. More precisely, the problem **LSP1′** is solved by the integer solution $(\hat{y}^k_s, s \in \Omega^k, k \in M)$ with $\hat{y}^{m_h}_{\rho_h} = 1$ for any $h \in H$ and $\hat{y}^k_s = 0$ for any other pair (s, k). Since all the jobs in E are covered by the schedules in $\bigcup_{h \in H} \rho_h$, thus the objective function value corresponding to this integer solution is $\sum_{j \in N \setminus E} w_j = F(y)$. This means that this integer solution is optimal too. □

Table 1: Computational results for $P||\sum w_j C_j$

problem m	problem n	LP-IP gap	problems solved at root node	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.0%	20	1	471	2.14
2	30	0.0%	14	1.4	1735	27.32
2	40	0.0%	12	1.3	4396	183.91
4	20	0.0%	20	1	274	0.47
4	30	$< 0.001\%$	13	1.8	794	4.96
4	40	$< 0.001\%$	7	3.0	1566	23.34
4	60	$< 0.001\%$	3	14.2	8712	523.15
4	80	$< 0.001\%$	3	16.3	1427	1723.49
8	20	0.0%	20	1	153	0.13
8	40	0.0%	17	1.2	896	3.09
8	60	$< 0.001\%$	6	10.4	1891	54.72
8	80	$< 0.001\%$	0	25.3	6958	447.29
8	100	$< 0.001\%$	0	48.0	13518	1524.50
12	40	0.0%	20	1	552	0.78
12	60	0.0%	8	7.4	1761	18.71
12	80	0.0%	0	20.5	3544	119.18
12	100	0.0%	0	32.3	7589	467.82
16	40	0.0%	20	1	473	0.41
16	60	0.0%	13	3.0	1297	5.13
16	80	$< 0.001\%$	3	14.4	1917	40.78
16	100	$< 0.001\%$	0	55.2	6006	276.61
20	60	0.0%	20	1	930	1.63
20	80	0.0%	4	9.0	2003	20.12
20	100	0.0%	0	20.4	4903	96.45

Table 2: Computational results for $Q||\sum w_jC_j$

problem m	problem n	LP-IP gap	problems solved at root node	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.0%	20	1	723	3.17
2	30	0.0%	20	1	2159	45.35
2	40	0.0%	20	1	2487	202.74
4	20	0.019%	18	1.3	625	1.47
4	30	0.012%	13	2.6	1540	10.23
4	40	0.025%	5	3.3	3521	87.46
4	60	0.014%	0	10.0	8435	1184.33
8	20	0.0%	20	1	756	0.74
8	40	$< 0.001\%$	15	1.1	1857	17.83
8	60	0.072%	0	3.2	7551	663.87
8	80	0.0152%	0	25.6	11815	2898.49
12	40	0.039%	10	3.3	3317	17.53
12	60	$< 0.001\%$	3	5.1	8497	176.34
12	80	0.013%	0	13.8	13021	746.18
12	100	0.012%	0	22.1	21908	2109.39
16	40	0.0%	20	1	3445	8.65
16	60	0.054%	0	23.5	8032	184.13
16	80	0.004%	0	27.0	6940	512.87
16	100	0.006%	0	30.7	9362	1267.56
20	60	0.033%	2	13.5	7311	35.45
20	80	0.029%	0	38.0	12970	551.78
20	100	0.073%	0	45.8	16375	1320.71

Table 3: Computational results for $R||\sum w_j C_j$

problem m	problem n	LP-IP gap	problems solved at root node	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.0%	20	1	664	2.08
2	30	0.0%	20	1	2021	47.82
2	40	0.0%	20	1	4629	436.39
4	20	0.0%	20	1	489	0.94
4	30	0.0%	18	1.2	1403	8.96
4	40	0.0%	20	1	2539	51.15
4	60	0.0%	10	3.4	7271	623.35
8	20	0.0%	20	1	529	0.94
8	40	0.0%	20	1	1728	10.12
8	60	0.002%	17	1.1	3720	87.97
8	80	$< 0.001\%$	15	1.2	7001	487.41
8	100	$< 0.001\%$	15	1.2	12906	2050.93
12	40	0.0%	20	1	1579	6.56
12	60	0.0%	20	1	3340	48.02
12	80	$< 0.001\%$	10	7.1	5724	203.34
12	100	0.016%	3	6.4	9362	1043.57
16	40	0.0%	20	1	1394	5.40
16	60	$< 0.001\%$	17	2.4	3174	44.85
16	80	0.002%	6	2.0	5394	206.42
16	100	0.0015%	5	3.4	7981	523.59
20	60	0.0%	20	1	2731	27.23
20	80	0.032%	12	4.5	4858	140.92
20	100	$< 0.001\%$	9	5.6	6930	362.80

Table 4: Computational results for $P||\sum w_j U_j$

problem m	problem n	LP-IP gap	problems solved at root node	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.0%	12	1.6	89	0.06
2	30	0.03%	10	2.4	537	3.77
2	40	0.12%	5	3.4	1320	28.12
2	50	0.22%	2	8.3	2689	176.93
4	20	0.0%	20	1.0	99	0.06
4	30	0.49%	9	2.1	505	0.92
4	40	0.56%	3	8.7	993	7.54
4	50	0.56%	0	25.8	3202	114.40
4	60	0.26%	0	81.2	10036	682.35
6	30	0.07%	11	1.4	176	0.21
6	40	0.04%	4	5.0	391	1.70
6	50	0.23%	0	19.7	1747	18.06
6	60	0.31%	0	43.5	4349	93.62
6	70	0.20%	0	61.0	6231	273.78
6	80	0.16%	0	117.5	9407	835.29
8	40	0.15%	7	3.3	379	1.61
8	50	0.10%	2	12.5	1556	8.34
8	60	0.34%	0	43.0	2387	55.22
8	70	0.27%	0	65.9	4021	97.74
8	80	0.33%	0	94.3	7139	305.40
10	40	0.0%	20	1.0	209	0.11
10	50	0.0%	8	4.6	774	1.95
10	60	0.25%	3	15.5	1119	12.50
10	70	0.25%	0	35.6	2031	33.63
10	80	0.23%	0	69.5	4057	95.12
10	90	0.16%	0	81.7	7671	286.95
10	100	0.08%	0	100.4	11456	1206.27

Table 5: Computational results for $Q||\sum w_jU_j$

problem m	problem n	LP-IP gap	problems solved at root node	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.73%	4	5.6	243	0.67
2	30	0.64%	0	17.2	1765	34.38
2	40	0.52%	0	54.0	6022	320.56
2	50	0.45%	0	145.2	21765	1344.31
4	20	0.0%	3	4.8	241	0.41
4	30	0.60%	0	27.0	1211	12.80
4	40	0.59%	0	54.7	3202	135.67
4	50	0.33%	0	72.2	4789	401.20
4	60	0.26%	0	81.2	6634	688.40
6	30	0.33%	1	10.8	667	3.02
6	40	0.36%	0	37.0	2175	29.40
6	50	0.18%	0	55.6	4371	116.53
6	60	0.12%	0	110.7	8073	605.13
6	70	0.07%	0	125.5	11340	1461.86
6	80	0.17%	0	223.2	26914	4230.07
8	40	0.05%	1	17.2	1285	10.8
8	50	0.007%	0	36.0	2057	65.76
8	60	0.13%	0	60.7	6321	340.21
8	70	0.004%	0	91.2	13573	181.20
8	80	0.34%	0	150.8	20886	1450.72
10	40	0.06%	2	13.4	811	5.43
10	50	0.15%	0	35.0	1876	24.65
10	60	0.11%	0	73.7	4657	121.70
10	70	0.26%	0	92.8	6064	453.05
10	80	0.23%	0	117.0	12356	1124.72
10	90	0.14%	0	165.4	34297	3138.90
10	100	0.10%	0	217.6	81102	5729.31

Table 6: Computational results for $R||\sum w_j U_j$

problem m	problem n	LP-IP gap	problems solved at root node	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.09%	7	2.4	271	0.47
2	30	0.11%	7	2.6	854	5.80
2	40	0.09%	6	3.4	1509	38.05
2	50	0.12%	1	10.6	4763	257.25
4	20	0.0%	17	1.2	135	0.13
4	30	0.22%	5	5.4	653	1.68
4	40	0.02%	2	12.2	1460	11.60
4	50	0.15%	0	77.8	8756	273.17
4	60	0.10%	0	132.6	17919	1388.40
6	30	0.0%	9	2.6	203	0.38
6	40	0.01%	2	6.8	444	1.72
6	50	0.01%	0	27.6	2185	30.34
6	60	0.004%	0	63.8	6241	169.20
6	70	0.08%	0	99.7	9073	435.84
6	80	0.09%	0	156.4	14658	1022.43
8	40	0.005%	2	10.8	695	2.81
8	50	$< 0.001\%$	0	36.0	2057	25.42
8	60	$< 0.001\%$	0	52.4	4743	118.37
8	70	0.003%	0	83.2	5439	181.20
8	80	0.001%	0	103.9	11368	849.61
10	40	0.0%	1	9.2	485	2.25
10	50	$< 0.001\%$	0	18.8	963	7.23
10	60	$< 0.001\%$	0	26.6	1738	26.08
10	70	0.006%	0	41.8	3864	123.45
10	80	0.02%	0	103.4	8107	466.81
10	90	0.02%	0	125.1	14297	1221.04
10	100	0.05%	0	178.2	42603	3287.52

Acknowledgement

This research was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research. The authors would like to thank the two anonymous referees for their helpful comments and suggestions on the earlier version of this paper.

References

[1] K.R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[2] J.W. Barnes and J.J. Brennan. An improved algorithm for scheduling jobs on identical machines. *AIIE Transactions*, 9:25–31, 1977.

[3] C. Barnhart, E. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. In J.R. Birge and K.G. Murty, editors, Mathematical Programming: State of the Art 1994, pages 186–207. The University of Michigan, 1994.

[4] H. Belouadah and C.N. Potts. Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Applied Mathematics*, 48:201–218, 1995.

[5] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* Elsevier Science Publishing Co., Inc., North Holland, 1976.

[6] D. Cattrysse, M. Salomon, R. Kuik, and L.N. Van Wassenhove. A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup times. *Management Science*, 39(4):477–486, 1993.

[7] L.M.A. Chan, P. Kaminsky, A. Muriel, and D. Simchi-Levi. Machine scheduling, linear programming and list scheduling heuristic. Technical report, Northwestern University, Chicago, 1995.

[8] Z.-L. Chen and W.B. Powell. A decomposition approach for a parallel machine just-in-time scheduling problem. Technical report, Statistics and Operation Research, Princeton University, 1996.

[9] T.C.E. Cheng and C.C.S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1990.

[10] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[11] P. De, J.B. Ghosh, and C.E. Wells. Due-date assignment and early/tardy scheduling on identical parallel machines. *Naval Research Logistics*, 41:17–32, 1994.

[12] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

[13] W.L. Eastman, S. Even, and I.M. Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 11:268–279, 1974.

[14] S.E. Elmaghraby and S.H. Park. Scheduling jobs on a number of identical machines. *AIIE Transactions*, 6:1–12, 1974.

[15] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[16] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[17] J.C. Ho and Y.-L. Chang. Heuristics for minimizing mean tardiness for m parallel machines. *Naval Research Logistics*, 38:367–381, 1991.

[18] J.C. Ho and Y.-L. Chang. Minimizing the number of tardy jobs for m parallel machines. *European Journal of Operational Research*, 84:343–355, 1995.

[19] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computations*, New York, 1972. Plenum Press.

[20] L.S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, New York, 1970.

[21] S. Lavoie, M. Minoux, and E. Odier. A new approach for crew pairing problems by column generation with an application to air transport. *European Journal of Operational Research*, 35:45–58, 1988.

[22] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, Logistics of Production and Inventory, Amsterdam, 1993. North Holland.

[23] E.L. Lawler and J.M. Moore. A functional equation and its applications to resource allocation and sequencing problems. *Management Science*, 16:77–84, 1969.

[24] A. Mehrotra and M.A. Trick. A column generation approach to graph coloring. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1993.

[25] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., Chichester, 1988.

[26] M. Pinedo. *Scheduling: Theory, Algorithm, and System*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[27] M.H. Rothkopf. Scheduling independent tasks on parallel processors. *Management Science*, 12:437–447, 1966.

[28] S.C. Sarin, S. Ahn, and A.B. Bishop. An improved scheme for the branch and bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime. *International Journal of Production Research*, 26(7):1183–1191, 1988.

[29] W.E. Smith. Various optimizer for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[30] J.M. van den Akker, J.A. Hoogeveen, and S.L. van de Velde. Parallel machine scheduling by column generation. Technical report, Center for Operations Research and Econometrics, Universite Catholique de Louvain, Belgium, 1995.

[31] P.H. Vance. Crew scheduling, cutting stock, and column generation: Solving huge integer programs. Technical report, PhD dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, 1993.

[32] P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

[33] S. Webster. New bounds for the identical parallel processor weighted flow time problem. *Management Science*, 38(1):124–136, 1992.

[34] S. Webster. A priority rule for minimizing weighted flow time in a class of paralle machine scheduling problems. *European Journal of Operational Research*, 70:327–334, 1993.

[35] S. Webster. Weighted flow time bounds for scheduling identical processors. *European Journal of Operational Research*, 80:103–111, 1995.