A Column Generation Based Decomposition Algorithm for a Parallel Machine Just-In-Time Scheduling Problem

Zhi-Long Chen
Department of Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104-6315
Email: zlchen@seas.upenn.edu

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

October 1997

abstract

We propose a column generation based exact decomposition algorithm for the problem of scheduling n jobs with an unrestrictively large common due date on m identical parallel machines to minimize total weighted earliness and tardiness. We first formulate the problem as an integer program, then reformulate it, using Dantzig-Wolfe decomposition, as a set partitioning problem with side constraints. Based on this set partitioning formulation, a branch and bound exact solution algorithm is developed for the problem. In the branch and bound tree, each node is the linear relaxation problem of a set partitioning problem with side constraints. This linear relaxation problem is solved by column generation approach where columns represent partial schedules on single machines and are generated by solving two single machine subproblems. Our computational results show that this decomposition algorithm is capable of solving problems with up to 60 jobs in reasonable cpu time.

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

1 Introduction

The current emphasis on the Just-In-Time (JIT) philosophy in industry has motivated the study of the scheduling models capable of capturing the essence of this idea. Unfortunately, as shown in the review article by Baker and Scudder [1], very little has been done in terms of developing solution procedures for scheduling problems in JIT environments with multiple machines.

In this article, we study the following parallel machine JIT scheduling problem. We are given n independent jobs $N = \{1, 2, ..., n\}$ to be scheduled on m identical parallel machines $M = \{1, 2, ..., m\}$. Associated with each job $j \in N$ are a processing time p_j, an earliness penalty weight u_j, a tardiness penalty weight v_j, and a common due date d that is given and unrestrictively large, i.e. d is not early enough to constrain the scheduling decision. We assume that $d \geq \sum_{j \in N} p_j$. Let C_j, $E_j = \max(0, d - C_j)$, and $T_j = \max(0, C_j - d)$ denote, respectively, the completion time, the earliness, and the tardiness of job j in a schedule. The objective of the problem is to find a schedule that allows idle time while prohibits preemption so as to minimize the total weighted earliness and tardiness, that is,

$$\min \sum_{j \in N} (u_j E_j + v_j T_j) \tag{1}$$

In the case of single machine, Hall and Posner [14] showed that the problem (1) is NP-hard even with $u_j = v_j$ for each $j \in N$. Therefore, our problem is NP-hard as well. Numerous results on other related single machine JIT scheduling problems can be found in the survey paper [1] and the recent papers by Li, Chen and Cheng [18], Davis and Kanet [9], Federgruen and Mosheiov [13], Liman and Lee [20], Li, Cheng and Chen [19], and Kim and Yano [15], among others.

There are only a handful of results published for the parallel machine case. When the common due date is a decision variable to be determined, Cheng and Chen [7] and De, Ghosh and Wells [10] independently showed that the problem of minimizing total earliness and tardiness penalties and a due date penalty with a common earliness penalty

weight and a common tardiness penalty weight, i.e.

$$\min \quad \sum_{j \in N} (uE_j + vT_j + wd) \tag{2}$$

(where u, v and w are the common earliness, tardiness, and due date penalty weights respectively) is NP-hard even with $m = 2$. De, Ghosh and Wells [10] proposed an $O(nm[(\lceil n/m \rceil + 1)w]^{2m})$ dynamic programming exact solution algorithm for this problem. Cheng [6] proposed a heuristic for this problem in the case when idle time is not allowed.

When the common due date is given and unrestrictively large, i.e. $d > \sum_{j \in N} p_j$, as we assume in this article, Emmons [12] investigated the problem (1) with $u_j \equiv u$ and $v_j \equiv v$ in both the identical and the uniform machine cases and shows that it can be easily solved using the so-called smallest-to-largest matching of coefficients. A similar bi-criterion problem is addressed in [12] and [10]. When the due dates of jobs are not common, Laguna and Velarde [21] suggested a heuristic for the parallel machine problem of minimizing total weighted earliness subject to the constraint that each job must be completed no later than its due date.

In this paper, we study the problem (1) with a given and unrestrictively large common due date $d \geq \sum_{j \in N} p_j$ and general earliness and tardiness penalty weights u_j and v_j. As we mentioned earlier, this problem is NP-hard. Hence it is very unlikely that an efficient (i.e. polynomial time) exact solution algorithm exists for the problem if $P \neq NP$. We design a computationally powerful exact solution algorithm that is capable of solving problems with a medium size. The design of our algorithm is based on the decomposition approach independently proposed by Chen and Powell [5], van den Akker, Hoogeveen and van de Velde [24], and Chan, Kaminsky, Muriel and Simchi-Levi [4] for parallel machine scheduling problems with an additive criterion. Existing computational results have shown that the decomposition approach is very effective for the parallel machine total weighted completion time problem ([5, 24]) and the parallel machine weighted number of tardy jobs problem ([5]).

The framework of the decomposition algorithm we are going to present can be out-

lined as follows. we first give an integer programming formulation for our problem, then present a set partitioning formulation (with side constraints) for it by applying Dantzig-Wolfe decomposition to the integer programming formulation. Based on the set partitioning formulation, we design a branch and bound exact solution algorithm. In the branch and bound tree, each node is the linear relaxation problem of a set partitioning problem (with side constraints). This linear relaxation problem is solved by column generation where columns represent schedules on single machines and are generated by solving two single machine subproblems. The single machine subproblems are NP-hard and solved by pseudopolynomial dynamic programming algorithms. The solution of the linear relaxation problem on each branch and bound node is always an extremely tight lower bound on the solution of the original integer problem. This excellent lower bounding is the key factor that leads to the success of the branch and bound algorithm. Branchings are conducted on variables in the original integer programming formulation instead of variables in the set partitioning problem such that subproblem structures are preserved. Computational experiments show that this exact decomposition algorithm is capable of solving problems with up to 60 jobs within reasonable cpu time.

The contribution of this paper is that our algorithm is the first branch and bound and the first computationally effective exact solution algorithm for an NP-hard multi-machine JIT scheduling problem. The remainder of this paper consists of four sections. The set partitioning formulation and the solution procedure for its linear relaxation problem are developed in Section 2. The branch and bound algorithm is designed in Section 3. The computational results are reported in Section 4. Finally, we conclude the paper in Section 5.

2 Decomposition of the Problem

In this section, we first give an integer programming formulation in Section 2.1 for the JIT problem under study. This formulation is then decomposed in Section 2.2, by Dantzig-Wolfe decomposition, into a master problem, which is formulated as a set

partitioning formulation with side constraints, and two single machine subproblems. The linear relaxation of the set partitioning master problem is solved by column generation.

2.1 Integer Programming Formulation

We define a *partial schedule* on a single machine to be a schedule on that machine formed by a subset of jobs of N. We call job j *early* if it is completed at or before the due date (i.e. $C_j \leq d$), and otherwise we say it is *tardy*. By the properties given in [1] for the corresponding single machine problem, we can conclude that any optimal schedule of our problem satisfies the following properties:

Property 1: On each machine, there is no idle time between jobs (though the first job on a machine may start later than time 0).

Property 2: On each machine, one job completes at time d.

Property 3: On each machine, the partial schedule is V-shaped: early jobs are scheduled in nonincreasing order of the ratio p_j/u_j and tardy jobs are scheduled in nondecreasing order of the ratio p_j/v_j.

For any subset $S \subseteq N$, we define the *PU order* of S to be the nonincreasing order of the ratios p_j/u_j for $j \in S$. Similarly, we define the *PV order* of S to be the nondecreasing order of the ratios p_j/v_j for $j \in S$. We prescribe that if two jobs i and j have an identical ratio, i.e. $p_i/u_i = p_j/u_j$ ($p_i/v_i = p_j/v_j$) in a PU (PV order), then the one with a smaller index precedes the other. This guarantees that for any subset of jobs, both the PU order and the PV order are unique. By Property 3, we can assume, without loss of generality, that in an optimal schedule, the early jobs on each machine form their PU order and the tardy jobs their PV order.

Define the following sets and $0-1$ variables:

- $B_j^E = \{i \in N \mid i \text{ precedes } j \text{ in the PU order of } N\}$
- $A_j^E = \{i \in N \mid i \text{ succeeds } j \text{ in the PU order of } N\}$

- $B_j^T = \{i \in N \mid i \text{ precedes } j \text{ in the PV order of } N\}$
- $A_j^T = \{i \in N \mid i \text{ succeeds } j \text{ in the PV order of } N\}$
- $X_{ij}^E = 1$ if job i and j are both scheduled early on the same machine and i is processed immediately after j; 0 otherwise.
- $X_{0j}^E = 1$ if job j is the last early job (i.e. it completes at the due date d by Property 2) on some machine; 0 otherwise.
- $X_{j,n+1}^E = 1$ if job j is the first early job on some machine; 0 otherwise.
- $X_{ij}^T = 1$ if job i and j are both scheduled tardy on the same machine and i is processed immediately before j; 0 otherwise.
- $X_{0j}^T = 1$ if job j is the first tardy job (i.e. it starts at the due date d) on some machine; 0 otherwise.
- $X_{j,n+1}^T = 1$ if job j is the last tardy job on some machine; 0 otherwise.

Then the scheduling problem we are considering can be formulated by the following integer program (**IP**) where the parameter p_0 is defined to be zero.

IP:

$$\min \quad \sum_{j \in N} (u_j E_j + v_j T_j) \tag{3}$$

subject to

$$\sum_{i \in A_j^E \cup \{0\}} X_{ij}^E + \sum_{i \in B_j^E \cup \{0\}} X_{ij}^T = 1, \quad \forall j \in N \tag{4}$$

$$\sum_{j \in N} X_{0j}^E \leq m \tag{5}$$

$$\sum_{j \in N} X_{0j}^T \leq m \tag{6}$$

$$\sum_{i \in A_j^E \cup \{0\}} X_{ij}^E = \sum_{i \in B_j^E \cup \{n+1\}} X_{ji}^E, \quad \forall j \in N \tag{7}$$

$$\sum_{i \in B_j^T \cup \{0\}} X_{ij}^T = \sum_{i \in A_j^T \cup \{n+1\}} X_{ji}^T, \quad \forall j \in N \tag{8}$$

$$E_0 = 0 \tag{9}$$

$$T_0 = 0 \tag{10}$$

$$E_j = \sum_{i \in A_j^E \cup \{0\}} (E_i + p_i) X_{ij}^E, \quad \forall j \in N \tag{11}$$

$$T_j = \sum_{i \in B_j^T \cup \{0\}} (T_i + p_j) X_{ij}^T, \quad \forall j \in N \tag{12}$$

$$X_{ij}^E \in \{0,1\}, \quad \forall i \in N \cup \{0\}, \quad j \in N \cup \{n+1\} \tag{13}$$

$$X_{ij}^T \in \{0,1\}, \quad \forall i \in N \cup \{0\}, \quad j \in N \cup \{n+1\} \tag{14}$$

The objective function (3) seeks to minimize total weighted earliness and tardiness. Constraint (4) ensures that each job j is scheduled exactly once. If $\sum_{i \in A_j^E \cup \{0\}} X_{ij}^E = 1$ then job j is scheduled early; and if $\sum_{i \in B_j^E \cup \{0\}} X_{ij}^T = 1$ then it is scheduled tardy. Constraints (5)-(6) represent the fact that there are only m machines available. Constraints (7)-(8) are the feasibility requirement for job scheduling. Constraints (9)-(12) define the earliness and tardiness for each job. It is easy to see that given X values, E and T values are uniquely determined by (9)-(12). Finally, constraints (13)-(14) represent the binary integrality requirement for the variables.

2.2 Dantzig-Wolfe Decomposition

In this section, we decompose the integer programming formulation **IP** given in Section 2.1 into a master problem with a set partitioning formulation with side constraints (Section 2.2.1) and two single machine subproblems (Section 2.2.3). The column generation approach is then applied to the linear relaxation of the set partitioning formulation (Section 2.2.2).

2.2.1 Set Partitioning Master Problem

Define an *early partial PU schedule* on a single machine to be a partial schedule on that machine in which (1) all the jobs are early and form their PU order, (2) the last job completes at the due date d, and (3) there is no inserted idle time between jobs. Similarly, define a *tardy partial PV schedule* on a single machine to be a partial schedule on that machine in which (1) all the jobs are tardy and form their PV order, (2) the

first job starts at the due date d, and (3) there is no inserted idle time. Let

- Ω^E = the set of all possible early partial PU schedules on a single machine.
- Ω^T = the set of all possible tardy partial PV schedules on a single machine.

Applying Dantzig-Wolfe decomposition [8], we decompose the formulation **IP** into a master problem consisting of (3)-(6) and (9)-(12), meaning that each job is scheduled exactly once and each machine is utilized at most once, and two subproblems, one with feasible region defined by (7), (9), (11) and (13), and the other with feasible region defined by (8), (10), (12) and (14). We will see in Section 2.2.3 that each of these two subproblems is a single machine scheduling problem.

Define the following coefficients and variables:

- $a_{js} = 1$ if schedule $s \in \Omega^E \cup \Omega^T$ covers job j; 0 otherwise.
- E_{js} = earliness of job j in s if schedule $s \in \Omega^E$ covers job j; 0 otherwise.
- T_{js} = tardiness of job j in s if schedule $s \in \Omega^T$ covers job j; 0 otherwise.
- $Z_s = 1$ if schedule $s \in \Omega^E \cup \Omega^T$ is used; 0 otherwise.

Then the master problem can be reformulated as the following set partitioning problem (**SP**) with two generalized upper bound (GUB) constraints:

SP:

$$\min \quad \sum_{j \in N} \left(\sum_{s \in \Omega^E} (u_j E_{js}) Z_s + \sum_{s \in \Omega^T} (v_j T_{js}) Z_s \right) \tag{15}$$

subject to

$$\sum_{s \in \Omega^E \cup \Omega^T} a_{js} Z_s = 1, \quad \forall j \in N \tag{16}$$

$$\sum_{s \in \Omega^E} Z_s \leq m \tag{17}$$

$$\sum_{s \in \Omega^T} Z_s \leq m \tag{18}$$

$$Z_s \in \{0,1\}, \quad \forall s \in \Omega^E \cup \Omega^T \tag{19}$$

Constraints (16), (17), (18) have exactly the same meanings as constraints (4), (5), (6)

respectively. Note that without the GUB constraints (17) and (18), the above formulation would be a pure set partitioning problem.

The feasible solutions of the formulation **SP** have a one-to-one correspondence to the feasible solutions of the original formulation **IP**. This can be easily justified as follows. Given a feasible solution of **SP**, $(\bar{Z}_s, s \in \Omega^E \bigcup \Omega^T)$. Let $,^E = \{s \in \Omega^E | \bar{Z}_s = 1\}$ and $,^T = \{s \in \Omega^T | \bar{Z}_s = 1\}$. Define

$\bar{X}^E_{ij} = 1$ if some schedule $\omega \in ,^E$ covers both i and j and i is immediately after j in ω; 0 otherwise.

$\bar{X}^T_{ij} = 1$ if some schedule $\omega \in ,^T$ covers both i and j and i is immediately before j in ω; 0 otherwise.

Then it can be easily shown that $\bar{X}$ defined here is a feasible solution to **IP**. Furthermore, if we let $\bar{E}$ and $\bar{T}$ to be computed by (9)-(12) with X being specified as $\bar{X}$, we can show that the resulting objective function value of **IP** (i.e. $\sum_{j \in N}(u_j \bar{E}_j + v_j \bar{T}_j)$) equals the corresponding objective function value of **SP** (i.e. $\sum_{j \in N}(\sum_{s \in \Gamma^E} u_j E_{js} + \sum_{s \in \Gamma^T} v_j T_{js})$). On the other hand, given a feasible solution of **IP**, $(\bar{X}^E_{ij}, \bar{X}^T_{ij}, i \in N \cup \{0\}, j \in N \cup \{n+1\})$, there must exist q $(= \sum_{j \in N} \bar{X}^E_{0j} \leq m)$ sequences of indices $(l_{h_l}, ..., l_2, l_1)$, for $l = 1, ..., q$ (where h_l is the total number of the indices in the l-th sequence), such that $\bar{X}^E_{0,l_1} = \bar{X}^E_{l_1,l_2} = ... = \bar{X}^E_{l_{h_l-1},l_{h_l}} = \bar{X}^E_{l_{h_l},n+1} = 1$. Clearly, $l_j \in B^E_{l_i}$ for each i, j with $i < j$. Then we can construct q early partial PU schedules with the corresponding job sequences being $(l_{h_l}, ..., l_2, l_1)$, for $l = 1, ..., q$. Let $,^E$ denote the set of these q partial PU schedules. Similarly, we can have r $(= \sum_{j \in N} \bar{X}^T_{0j} \leq m)$ tardy partial PV schedules. Let $,^T$ be the set of these schedules. Define $\bar{Z}_s = 1$ for each $s \in ,^E \bigcup ,^T$ and $\bar{Z}_s = 0$ for any other schedule s. Then it can be easily proved that $\bar{Z}$ solves **SP** and the resulting objective function value of **SP** is the same as the corresponding objective function value of **IP**.

However, **SP** is not merely a reformulation of **IP**. Their difference lies in their linear relaxation problems. Relaxing the binary integrality constraints (13)-(14) and (19) by allowing the variables involved to be any real number between 0 and 1, we get the linear relaxation problems, denoted by **LIP** and **LSP**, respectively for the integer problem

IP and **SP**. It is easy to show that any feasible solution of **LSP**, $(\bar{Z}_s, s \in \Omega^E \bigcup \Omega^T)$, corresponds to a feasible solution of **LIP**, $(\bar{X}^E_{ij}, \bar{X}^T_{ij}, i \in N \cup \{0\}, j \in N \cup \{n+1\})$, given by

$$\bar{X}^E_{ij} = \sum_{s \in \Omega^E} g^s_{ij} \bar{Z}_s \tag{20}$$

$$\bar{X}^T_{ij} = \sum_{s \in \Omega^T} h^s_{ij} \bar{Z}_s \tag{21}$$

where g^s_{ij} is 1 if $s \in \Omega^E$ covers both i and j and i is immediately after j and 0 otherwise, and h^s_{ij} is 1 if $s \in \Omega^T$ covers both i and j and i is immediately before j and 0 otherwise. The reverse is not true because any feasible solution $\bar{X}$ of **LIP** that cannot be expressed as the form (20)-(21) (for any subset of schedules in $\Omega^E \bigcup \Omega^T$ and any possible values $\bar{Z}$ of them) does not correspond to any feasible solution of **LSP**. This means that the feasible solutions of the problem **LSP** correspond to a subset of the feasible solutions of the problem **LIP** and hence **LSP** can yield tighter lower bound than **LIP**. This is the major advantage of using the set partitioning formulation **SP**, instead of the original formulation **IP**, as we know that in a branch and bound algorithm, tight lower bounding is crucial.

2.2.2 Column Generation for Solving LSP

In the branch and bound algorithm (described later), we solve at each iteration one linear relaxation problem **LSP**. Each column in **LSP** represents an early partial PU schedule or a tardy partial PV schedule on a single machine. As the number of early partial PU schedules and tardy partial PV schedules ($|\Omega^E| + |\Omega^T|$) can be extremely large even for a small-size problem, it is impractical to explicitly list all the columns when solving **LSP**. So we use the well-known column generation approach (see, e.g. Lasdon [16]) to generate necessary columns only in order to solve **LSP** efficiently.

Column generation approach has been successfully applied to many large scale optimization problems, such as, vehicle routing (Desrochers, Desrosiers and Solomon [11]), air crew scheduling (Lavoie, Minoux and Odier [17], Vance [25]), lot sizing and schedul-

ing (Cattrysse, Salomon, Kuik and Van Wassenhove [3]), graph coloring (Mehrotra and Trick [22]), and cutting stock (Vance, Barnhart, Johnson and Nemhauser [26]).

The column generation procedure for our problem **LSP** consists of the following four major steps, which are repeated until no column with negative reduced cost is found.

- solving a restricted master problem of **LSP**, i.e, the problem **LSP** 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 most negative reduced costs based on the subproblem solutions and adding the new columns to the restricted master problem.

2.2.3 Two Single Machine Subproblems

When solving **LSP** by the column generation procedure described earlier, the goal of solving subproblems is to find the column with the minimum reduced cost. In a restricted master problem of **LSP**, let π_j denote the dual variable value corresponding to job j, for each $j \in N$, in constraint (16), and σ^E and σ^T denote the dual variable values corresponding to (17) and (18) respectively. Then the reduced cost r_s^E of the column corresponding to an early partial PU schedule $s \in \Omega^E$ is given by:

$$r_s^E = \sum_{j \in N} (u_j E_{js} - a_{js}\pi_j) - \sigma^E \tag{22}$$

Similarly, the reduced cost r_s^T of the column corresponding to a tardy partial PV schedule $s \in \Omega^T$ is given by:

$$r_s^T = \sum_{j \in N} (v_j T_{js} - a_{js}\pi_j) - \sigma^T \tag{23}$$

Hence, when solving **LSP** by the column generation approach, we need to solve two subproblems, one for finding an early partial PU schedule $s \in \Omega^E$ on a single machine with the minimum value of r_s^E, and the other for finding a tardy partial PV schedule

$s \in \Omega^T$ on a single machine with the minimum value of r_s^T. We call the two single machine subproblems **SUB1** and **SUB2** respectively. By the formula (22) and the fact that σ^E is common for any early partial PU schedule $s \in \Omega^E$, we can state subproblem **SUB1** precisely as the problem of finding an early partial PU schedule on a single machine so as to minimize the total weighted earliness minus the total dual variable value of the jobs in the schedule. Similarly, subproblem **SUB2** can be precisely stated as the problem of finding a tardy partial PV schedule on a single machine so as to minimize the total weighted tardiness minus the total dual variable value of the jobs in the schedule.

Let s_1 and s_2 be the optimal schedules of the two subproblems **SUB1** and **SUB2** respectively, then the quantity $\min\{r_{s_1}^E, r_{s_2}^T\}$ is the minimum possible reduced cost of a column in the corresponding master problem **LSP**. If this quantity is nonnegative, then the master problem **LSP** has been solved, otherwise, new columns with negative reduced costs generated in the course of solving subproblems are added to the master problem **LSP** and the master problem is then solved again.

It is not difficult to see that both **SUB1** and **SUB2** are reducible to the following problem (**WC**): Find a partial schedule in a single machine that minimizes the total weighted completion time minus the total dual variable value of the jobs in it. Chen and Powell [5] have shown that the problem **WC** is NP-hard in the ordinary sense and can be solved by a pseudopolynomial dynamic programming algorithm. Therefore the problems **SUB1** and **SUB2** are both NP-hard. Furthermore, like the problem **WC**, the problems **SUB1** and **SUB2** can be solved by pseudopolynomial dynamic programming algorithms.

In the following, we give a pseudopolynomial dynamic programming algorithm for each of the two single machine subproblems **SUB1** and **SUB2**.

Algorithm 1 for SUB1

(1a) First reindex the jobs such that $p_1/u_1 \geq p_2/u_2 \geq ... \geq p_n/u_n$. Then $(1, 2, ..., n)$ forms the PU order of N and $A_j^E = \{j+1, j+2, ..., n\}$ for each $j \in N$. Define set $S_j^E = A_j^E \bigcup \{n+1\} = \{j+1, j+2, ..., n+1\}$. Let $P = \sum_{i=1}^n p_i$ be the total processing

time of the jobs.

(1b) Let $F(j,t)$ denote the minimum objective function value (total weighted earliness minus total dual variable value) of the early partial PU schedule of a subset of jobs of $\{j, j+1, ..., n\}$, provided that job j is the first job in the schedule and the total processing time of the jobs in the schedule is t.

(1c) Initial values:

$$F(j,t) = \infty \quad \text{for } t < 0,\ j = n+1, n, ..., 1$$

$$F(n+1,t) = 0 \quad \text{for } t \geq 0$$

(1d) Recursive relation:

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

$$F(j,t) = \min_{i \in S_j^E} \{F(i, t-p_j) + u_j(t-p_j) - \pi_j\}, \tag{24}$$

(1e) The problem is solved by computing

$$\min_{j \in N, 0 \leq t \leq P} \{F(j,t)\} \tag{25}$$

Algorithm 2 for SUB2

(2a) First reindex the jobs such that $p_1/v_1 \leq p_2/v_2 \leq ... \leq p_n/v_n$. Then $(1, 2, ..., n)$ forms the PV order of N and $B_j^T = \{1, 2, ..., j-1\}$ for each $j \in N$. Define set $S_j^T = B_j^T \bigcup \{0\} = \{0, 1, ..., j-1\}$. Let $P = \sum_{i=1}^{n} p_i$ be the total processing time of the jobs.

(2b) Let $F(j,t)$ denote the minimum objective function value (total weighted tardiness minus total dual variable value) of the tardy partial PV schedule of a subset of jobs of $\{1, 2, ..., j\}$, provided that job j is the last job in the schedule and the total processing time of the jobs in the schedule is t.

(2c) Initial values:

$$F(j,t) = \infty \quad \text{for } t < 0,\ j = 0, 1, ..., n$$

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

(2d) Recursive relation:

For $j = 1, 2, ..., n$; $t = 0, ..., P$:

$$F(j,t) = \min_{i \in S_j^T} \{F(i, t - p_j) + v_j t - \pi_j\}, \tag{26}$$

(2e) The problem is solved by computing

$$\min_{j \in N, 0 \leq t \leq P} \{F(j,t)\} \tag{27}$$

It is easy to show that the above Algorithm 1 and Algorithm 2 solve the single machine subproblems **SUB1** and **SUB2** to optimality respectively. The worst case complexities of Algorithm 1 and Algorithm 2 are both bounded by $O(n^2 P)$ because there are a total of nP states in each of the dynamic programs and it takes no more than $O(n)$ time to compute the value for a state.

We note that in terms of the worst case complexity, Algorithms 1 and 2 are not the fastest possible algorithms for the single machine subproblems **SUB1** and **SUB2**. A faster algorithm for **SUB1** can be obtained by revising Algorithm 1 as follows. In procedure **(1b)**, let $F(j,t)$ be the minimum objective function value of the early partial PU schedule of a subset of jobs of $\{j, j+1, ..., n\}$, provided that the total processing time of the jobs in the schedule is t. Replace (24) and (25) by

$$F(j,t) = \min\{F(j+1, t - p_j) + u_j(t - p_j) - \pi_j, \quad F(j+1, t)\}, \tag{28}$$

and

$$\min_{0 \leq t \leq P} \{F(j,t)\} \tag{29}$$

respectively. Then the resulting DP algorithm has the worst case time complexity bounded by $O(nP)$, faster than Algorithm 1. Unfortunately, the revised Algorithm 1 will no longer work after a branching procedure is performed. In the recursion (28), it is implicitly assumed that any job i, for $i = j+1, ..., n$, is eligible to be scheduled immediately after job j. In our branch and bound algorithm, every branching procedure

imposes a restriction on which jobs can be scheduled immediately after some job j (i.e. the branching procedure changes set A_j^E), and hence the recursion (28) is no longer valid after branching. Similarly, there is a faster DP algorithm for **SUB2** that will no longer work after a branching procedure is performed. Note that branching will have no effect on Algorithms 1 and 2 because branching merely updates sets A_j^E and B_j^T, which has been explicitly taken into account in the recursive equations in Algorithms 1 and 2.

3 Branch and Bound Algorithm

In this section, we give the details of our branch and bound (b&b) algorithm for solving the entire problem **SP**. Special attention is given to the branching strategy used in the algorithm.

3.1 Description of the Algorithm

In the b&b tree, each b&b node is a linear relaxation problem **LSP** with some restriction on partial schedule sets Ω^E and Ω^T imposed by branching rules (described later). An upper bound (UB), that is the minimum integer solution value obtained so far, is associated with the b&b tree. By contrast, a lower bound (LB_i) is associated with each b&b node i in the b&b tree. LB_i is the smallest integer greater than or equal to the solution value of the father node of node i. At each b&b iteration, one b&b node is solved using the column generation approach described in Section 2 after its restricted master problem is initialized using all the columns of its father node except the ones that must be deleted based on branching rules. There are three possible cases for the solution to a b&b node.

Case 1: If the solution is integral, then we first prune this node from the b&b tree since none of its offsprings will produce better integer solution. Then the solution value (SV) is compared with the current upper bound UB of the b&b tree. If $SV < UB$, then this node becomes the best integer node and the upper bound of the b&b tree is updated: $UB \Leftarrow SV$, and the lower bound LB_i of each active (or unpruned) b&b node

in the tree is compared with this new upper bound UB. If $LB_i > UB$ then it is not necessary to consider node i any more and node i is thus pruned from the tree.

Case 2: If the solution is fractional and the integer part of its solution value is greater than or equal to the upper bound UB of the b&b tree, then this node is pruned from the tree since the integer solutions of its offsprings will be no better than that of the current best integer node.

Case 3: If the solution is fractional and the integer part of its solution value is less than the upper bound UB of the b&b tree, then in most cases, a branching decision is made to create two son nodes of this node based on this fractional solution. However, in some cases, as we show later, an integer solution to the b&b node can be constructed based on the already obtained fractional solution and hence it becomes Case 1 and no branching is necessary on this node.

Usually, in a branch and bound algorithm, two classes of decisions need to be made (see, e.g. Nemhauser and Wolsey [23]) throughout the algorithm. One is called *node selection* which is to select an active b&b node in the b&b tree to be explored (solved) next. The other is called *branching variable selection* which 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, i.e. its solution satisfies Case 3, 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, i.e. its solution satisfies Case 1 or Case 2, 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.

3.2 Branching Variable Selection

For solving our problem **SP**, traditional branching on the Z-variables in the problem may cause trouble along a branch where a variable has been set to zero. Recall that Z_s in **SP** represents a partial schedule on a single machine generated by solving a single machine subproblem. The branching $Z_s = 0$ means that this partial schedule is excluded and hence no such schedule can be generated in subsequent subproblems. However, it is very hard to exclude a schedule when solving a single machine subproblem.

Fortunately, there is a simple remedy to this difficulty. Instead of branching on the Z-variables in the set partitioning formulation **SP**, we branch on the X-variables in the original formulation **IP**. This branching variable selection strategy, that is, branching on variables in the original formulation, has been proven successful in many branch and bound algorithms for problems that can be reformulated, usually by Dantzig-Wolfe decomposition, as a master problem and one or some subproblems. See Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance [2] for a class of such problems.

As we mentioned in Section 2.2.1, any feasible solution $(\bar{Z}_s, s \in \Omega^E \bigcup \Omega^T)$ of the problem **LSP** corresponds to a feasible solution $(\bar{X}^E_{ij}, \bar{X}^T_{ij}, i \in N \cup \{0\}, j \in N \cup \{n+1\})$ of the problem **LIP** such that (20) and (20) are satisfied. Consider a b&b node, that is, a linear programming problem **LSP**. Suppose that its solution, denoted as $(\bar{Z}_s, s \in \Omega^E \bigcup \Omega^T)$, is fractional and satisfies Case 3 described in Section 3.1. Compute the corresponding $\bar{X}^E_{ij}$ and $\bar{X}^T_{ij}$ values by (20) and (20). If each $\bar{X}^E_{ij}$ and $\bar{X}^T_{ij}$ is integral (i.e. 0 or 1), then, despite the fact that the solution $(\bar{Z}_s, s \in \Omega^E \bigcup \Omega^T)$ of the b&b node is fractional, an integer solution for the b&b node can be constructed as follows based on this fractional solution. First, it is easy to see that the $\bar{X}$ is a feasible solution to the integer problem **IP** and the corresponding objective function value of **IP** is the same as the objective function value of **LSP** under $\bar{Z}$. As we mentioned in Section 2.2.1, in this case, there must exist q $(= \sum_{j \in N} \bar{X}^E_{0j} \leq m)$ sequences of indices $(l_{h_l}, ..., l_2, l_1)$, for $l = 1, ..., q$, (where h_l is the total number of the indices in the l-th sequence), such that $\bar{X}^E_{0,l_1} = \bar{X}^E_{l_1,l_2} = ... = \bar{X}^E_{l_{h_l-1},l_{h_l}} = \bar{X}^E_{l_{h_l},n+1} = 1$. Clearly, $l_j \in B^E_{l_i}$ for each i, j with $i < j$.

Then we can construct q early partial PU schedules with the corresponding job sequences being given by the sequences $(l_{h_l}, ..., l_2, l_1)$, for $l = 1, ..., q$. Let $, ^E$ denote the set of these schedules. Similarly, we can have r $(= \sum_{j \in N} \bar{X}_{0j}^T \leq m)$ tardy partial PV schedules. Let $, ^T$ be the set of these schedules. Define $\hat{Z}_s = 1$ for each $s \in , ^E \bigcup , ^T$ and $\hat{Z}_s = 0$ for any other schedule s. Then it can be easily proved that $\hat{Z}$ is a feasible solution to the integer problem **SP** (and to the linear relaxation problem **LSP** as well) and the resulting objective function value of **SP** is the same as the corresponding objective function value of **IP**. This means that $\bar{Z}$ and $\hat{Z}$ yield the same objective function value of **LSP**. Thus the integer solution $\hat{Z}$ is optimal to **LSP**. Hence, in this case, the b&b node actually satisfies Case 1 and no branching on this node is necessary.

If there exists some $\bar{X}_{ij}^E$ or $\bar{X}_{ij}^T$ that is fractional, then a branching decision should be made on this b&b node. A pair (h, l) is selected such that $\bar{X}_{hl}^E$ or $\bar{X}_{hl}^T$ has the most fractional value, i.e.

$$\min\{|\bar{X}_{hl}^E - 0.5|, |\bar{X}_{hl}^T - 0.5|\} = \min_{(i,j)}\{|\bar{X}_{ij}^E - 0.5|, |\bar{X}_{ij}^T - 0.5|\}$$

There are two possible cases:

(a) $|\bar{X}_{hl}^E - 0.5| = \min\{|\bar{X}_{hl}^E - 0.5|, |\bar{X}_{hl}^T - 0.5|\}$;

(b) $|\bar{X}_{hl}^T - 0.5| = \min\{|\bar{X}_{hl}^E - 0.5|, |\bar{X}_{hl}^T - 0.5|\}$.

If (a) happens, then two son nodes are created, one along the branch with X_{hl}^E fixed as 0 and the other along the branch with X_{hl}^E fixed as 1. If X_{hl}^E 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 early partial PU schedules where job h is scheduled immediately after job l if $h \neq 0$ or job l is scheduled last if $h = 0$. At the same time, the structure of the subproblem **SUB1** is updated such that $A_l^E := A_l^E \setminus \{h\}$, which guarantees that no early partial PU schedule will be generated where job h is processed immediately after job l if $h \neq 0$ or where job l is processed last if $h = 0$. If X_{hl}^E 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 early partial PU schedules where job l is scheduled immediately before a job other than h and the early partial PU schedules

where a job other than l is scheduled immediately before job h. The structure of the subproblem **SUB1** is also updated accordingly such that $A_l^E := \{h\}$ and $A_i^E := A_i^E \setminus \{h\}$ for each $i \in N \setminus \{l\}$, which ensures that any early partial PU schedule that contains job l processes job h immediately after l.

Similarly, if (b) happens, two son nodes are created, one along the branch with X_{hl}^T fixed as 0 and the other along the branch with X_{hl}^T fixed as 1. If X_{hl}^T 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 tardy partial PV schedules where job h is scheduled immediately before job l if $h \neq 0$ or job l is scheduled first if $h = 0$. At the same time, the structure of the subproblem **SUB2** is updated such that $B_l^T := B_l^T \setminus \{h\}$, which guarantees that no tardy partial PV schedule will be generated where job h is processed immediately before job l if $h \neq 0$ or where job l is processed first if $h = 0$. If X_{hl}^T 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 tardy partial PV schedules where job l is scheduled immediately after a job other than h and the tardy partial PV schedules where a job other than l is scheduled immediately after job h. The structure of the subproblem **SUB2** is also updated accordingly such that $B_l^T := \{h\}$ and $B_i^T := B_i^T \setminus \{h\}$ for each $i \in N \setminus \{l\}$, which ensures that any tardy partial PV schedule that contains job l processes job h immediately before l.

4 Computational Results

In this section, we do computational experiments to test the performance of our decomposition algorithm. Our algorithm is coded in C and tested on a Silicon Graphics Iris Workstation with a 100 MHZ processor. The linear programming problems **LSP** are solved by the commercial solver CPLEX. Two groups of test problems are randomly generated as follows.

- Number of machines (m). We use five different numbers: 2, 3, 4, 5, 6.

- Number of jobs (n). We use five different numbers: 20, 30, 40, 50, 60.
- Weights. The earliness and tardiness penalty weights u_j and v_j for each job are both integers uniformly drawn from $[1, 100]$.
- Processing times. For the test problems in Group One, each processing time is an integer uniformly drawn from $[1, 10]$. For those ones in Group Two, each processing time is an integer uniformly drawn from $[1, 100]$.

For every possible combination of m and n, twenty test problems are generated for each group. So there are totally 500 test problems in each group. As we will see, for fixed number of jobs (n), the running time of our algorithm becomes less when the number of machines (m) increases. For this reason, we do not consider problems with more than six machines.

We note that it can be expected that with fixed m and n, a problem in Group One can be solved relatively faster than a problem in Group Two because the complexity of Algorithms 1 and 2 that solve subproblems is increasing with the total processing time of the jobs. This is the reason why we use two groups of test problems.

Tables 1 and 2 list the computational results for the Group One and Group Two test problems respectively. In these tables, the first two columns "m" and "n" represent the number of machines and the number of jobs of a test problem respectively. For each pair of m and n, twenty problems are tested. Each of the other entries in these tables represents an averaged performance measure based on these twenty testing 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 **LSP**. The column "problems solved without branching" indicates the number of problems (out of 20) solved without branching (i.e. solved at the root node of the b&b tree). The column "b&b nodes explored" represents the average number of b&b nodes searched for solving the problems. The other two columns "columns generated" and "cpu time" represent, respectively, the average number of

columns generated and the average cpu time (in seconds) consumed for solving the problems.

From these tables, we can make the following observations:

- We can conclude that the lower bound given by the linear relaxation problem **LSP** is extremely close to the solution value of the integer problem **SP**. For every problem we tested, the gap between the lower bound and the integer solution value is less than 0.1%. We believe that the success of our branch and bound algorithm is mainly due to this excellent lower bounding.
- Our algorithm is capable of solving the problems with up to 60 jobs in reasonable cpu time.
- The performance of our algorithm varies with the ratio (n/m). When this ratio is relatively small (say, less than ten), the algorithm works very well. On the other hand, when this ratio increases, the column generation procedure suffers from degeneracy and the algorithm becomes less effective (even for problems with a small number of machines).

5 Conclusion

We have developed a branch and bound exact solution algorithm for the parallel machine problem of minimizing the total earliness-tardiness penalties in the case when all the jobs have a given and unrestrictively large common due date. The algorithm uses a column generation approach based on a set partitioning formulation. The computational experiments have shown that this algorithm is promising for medium, and even relatively large scale problems. This once again demonstrates the powerfulness of the decomposition approach for hard parallel machine scheduling problems. However, research efforts should be made in the future to resolve the degeneracy associated with the column generation approach so that the algorithm proposed in this paper might be equally effective for those problems with a large number of jobs relative to number of

machines. Another research topic is to apply the same approach to the same problem with uniform and unrelated parallel machines.

Table 1: Computational results on problems with processing times drawn from the uniform distribution $[1, 10]$

m	n	LP-IP gap	problems solved without branching	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.098%	8	3.5	624	3.5
2	30	0.054%	6	2.8	1381	10.87
2	40	0.018%	5	4.0	2544	48.77
2	50	0.022%	7	4.0	3498	122.43
2	60	0.017%	3	4.1	6727	633.64
3	20	0.004%	16	1.2	369	0.87
3	30	0.007%	13	2.0	876	4.91
3	40	0.005%	7	2.5	1620	19.54
3	50	0.032%	2	5.5	2945	72.40
3	60	0.037%	0	7.6	5031	249.68
4	20	0.047%	16	1.4	309	0.55
4	30	0.015%	11	1.8	676	2.12
4	40	0.048%	3	4.5	1539	14.25
4	50	0.011%	0	5.6	2362	32.59
4	60	0.035%	0	8.0	3924	136.93
5	20	0.0%	20	1	264	0.44
5	30	0.0%	14	1.3	594	1.42
5	40	0.043%	6	2.4	1112	7.10
5	50	0.014%	0	5.1	2137	32.85
5	60	0.024%	0	5.8	2859	61.44
6	20	0.0%	20	1	271	0.36
6	30	0.0%	18	1.3	550	1.33
6	40	0.012%	11	2.2	1023	4.98
6	50	0.008%	4	4.7	1757	15.81
6	60	0.014%	0	8.3	2644	51.71

Table 2: Computational results on problems with processing times drawn from the uniform distribution $[1, 100]$

m	n	LP-IP gap	problems solved without branching	b&b nodes explored	columns generated	cpu time (in seconds)
2	20	0.0%	20	1	568	1.91
2	30	0.014%	7	4.8	1526	77.14
2	40	0.065%	4	5.0	3049	441.37
2	50	0.006%	2	7.6	4091	2430.45
2	60	0.023%	1	10.1	8376	9602.73
3	20	0.0%	20	1	383	0.88
3	30	0.005%	14	1.8	871	10.40
3	40	0.003%	9	2.4	1445	68.66
3	50	0.07%	5	4.5	3113	700.44
3	60	0.009%	3	6.0	3440	2078.57
4	20	0.0%	20	1	285	0.54
4	30	0.033%	13	1.4	717	3.20
4	40	0.032%	10	2.8	1419	34.82
4	50	0.011%	5	5.2	2158	126.31
4	60	0.013%	5	8.3	3717	1397.34
5	20	0.0%	20	1	264	0.58
5	30	0.047%	9	2.1	583	2.97
5	40	0.003%	8	2.2	1100	10.37
5	50	0.009%	9	3.3	1753	69.84
5	60	0.031%	4	8.7	3359	449.08
6	20	0.0%	20	1	240	0.35
6	30	0.021%	16	1.5	524	1.63
6	40	0.004%	13	3.4	1047	14.47
6	50	0.036%	6	4.0	1631	35.93
6	60	0.002%	2	6.9	2342	132.4

Acknowledgement

This research was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research.

References

[1] K.R. Baker and G.D. Scudder. Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38, 1990.

[2] 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.

[3] 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.

[4] 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.

[5] Z.-L. Chen and W.B. Powell. Solving parallel machine scheduling problems by column generation. Technical report, Statistics and Operation Research, Princeton University, 1995.

[6] T.C.E. Cheng. A heuristic for common due-date assignment and job scheduling on parallel machines. *Journal of Operational Research Society*, 40:1129–1135, 1989.

[7] T.C.E. Cheng and Z.-L. Chen. Parallel-machine scheduling with bath setup times. *Operations Research*, 42:1171–1174, 1994.

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

[9] J.S. Davis and J.J. Kanet. Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics*, 40:85–101, 1993.

[10] 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.

[11] 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.

[12] H. Emmons. Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics*, 34:803–810, 1987.

[13] A. Federgruen and G. Mosheiov. Simultaneous optimization of efficiency and performance balance measures in single-machine scheduling problems. *Naval Research Logistics*, 40:951–970, 1993.

[14] N. G. Hall and M.E. Posner. Earliness-tardiness scheduling problems, i: Weighted deviation of completion times about a common due date. *Operations Research*, 39:836–846, 1991.

[15] Y.-D. Kim and C.A. Yano. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics*, 41:913–933, 1994.

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

[17] 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.

[18] C.-L. Li, Z.-L. Chen, and T.C.E. Cheng. A note on one-processor scheduling with asymmetric earliness and tardiness penalties. *Operations Research Letters*, 13:45–48, 1993.

[19] C.-L. Li, T.C.E. Cheng, and Z.-L. Chen. Single-machine scheduling to minimize the weighted number of early and tardy aggreeable jobs. *Computers & Operations Research*, 22:205–219, 1995.

[20] S.D. Liman and C.Y. Lee. Error bound of a heuristic for the common due date scheduling problem. *ORSA Journal on Computing*, 5:420–425, 1993.

[21] Laguna M. and J.L.G. Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.