

# Bi-criteria Scheduling Problems: Number of Tardy Jobs and Maximum Weighted Tardiness

Yumei Huo      Joseph Y-T. Leung      Hairong Zhao

Department of Computer Science  
New Jersey Institute of Technology  
Newark NJ 07102, USA.

yh23@njit.edu    leung@cis.njit.edu    hairong@cis.njit.edu

June 16, 2005

## Abstract

Consider a single machine and a set of  $n$  jobs that are available for processing at time 0. Job  $j$  has a processing time  $p_j$ , a due date  $d_j$  and a weight  $w_j$ . We consider bi-criteria scheduling problems involving the maximum weighted tardiness and the number of tardy jobs. We give NP-hardness proofs for the scheduling problems when either one of the two criteria is the primary criterion and the other one is the secondary criterion. These results answer two open questions posed by Lee and Vairaktarakis in 1993. We consider complexity relationships between the various problems, give polynomial-time algorithms for some special cases, and propose fast heuristics for the general case. The effectiveness of the heuristics is measured by empirical study. Our results show that one heuristic performs extremely well compared to optimal solutions.

## 1 Introduction

Since the beginning most of the research in scheduling has concentrated on a single criterion. Numerous optimal and approximation algorithms have been developed for single-criterion problems; see Pinedo [16] and Brucker [2]. However, companies are usually faced with the problem of satisfying several different groups of people. According to Panwalkar *et al.* [15], managers actually develop schedules based on multiple criteria. Woolsey [22] described a problem faced by the scheduler at a southwestern company that needs to satisfy simultaneously the salespeople and the customers. In this company, when the salespeople take customer orders, they promise the job will be ready on a specific

date. Salespeople are paid commissions based on the tardiness of the order; full commissions are paid for on-time orders, but the commission decreases to a certain minimum value as the tardiness increases. Clearly, the scheduler at this company is faced with unhappy customers and salespeople if not all jobs can be on time. From the perspective of the salespeople, minimizing maximum tardiness will be the fairest measure since the person penalized the most is hurt as little as possible. However, such a schedule could have many tardy jobs, which is not good from the customer's point of view. In this situation there are two criteria in play: number of tardy jobs and maximum tardiness.

From the above example, we can see that there is a need for further research in multi-criteria scheduling problems. Indeed, these problems have received more attention in the last three decades; see [3, 4, 5, 6, 7, 8, 12, 18, 19, 20]. In this paper we are concerned with scheduling problems with two criteria only. We seek a schedule that minimizes one criterion (the secondary criterion), subject to the constraint that the other criterion (the primary criterion) is minimum.

Consider a single-machine scheduling problem, where  $n$  jobs are ready for processing at time 0. Each job  $j$  has a processing time  $p_j$ , a due date  $d_j$  and a weight  $w_j$ . If  $S$  is a schedule of the  $n$  jobs, we let  $C_j$  denote the completion time of job  $j$  in  $S$ . If  $C_j > d_j$ , we say that job  $j$  is tardy and we let  $T_j = C_j - d_j$  denote its tardiness. In addition, we use the variable  $U_j$  as an indicator that job  $j$  is tardy; in this case  $U_j$  is set to 1. On the other hand, if  $C_j \leq d_j$ , we say that job  $j$  is on time, and we let  $U_j = 0$  and  $T_j = 0$ .

Most of the single-criterion scheduling problems are concerned with minimizing the total completion time  $\sum C_j$ , the number of tardy jobs  $\sum U_j$ , the maximum tardiness  $T_{\max} = \max\{T_j\}$ , the maximum weighted tardiness  $\max\{w_j T_j\}$ , and the total tardiness  $\sum T_j$ . Following the notation of Graham *et al.* [10], the above problems are denoted by  $1 \parallel \sum C_j$ ,  $1 \parallel \sum U_j$ ,  $1 \parallel T_{\max}$ ,  $1 \parallel \max\{w_j T_j\}$  and  $1 \parallel \sum T_j$ , respectively.

In this paper we are concerned mainly with the following criteria: the number of tardy jobs  $\sum U_j$ , the maximum tardiness  $T_{\max}$  and the maximum weighted tardiness  $\max\{w_j T_j\}$ .

A schedule with the minimum number of tardy jobs can be obtained by the Hodgson-Moore algorithm [13], which schedules jobs in ascending order of due dates. In the course of scheduling, if a job, say  $k$ , completes after its due date, then the longest job currently in the schedule (including job  $k$ ) will be deleted from the schedule. The deleted jobs will be scheduled after all the on-time jobs, one after another in any order.

Maximum tardiness can be minimized by the EDD (earliest due date first) rule, which schedules jobs in ascending order of due dates. Maximum weighted tardiness can be solved by an algorithm due to Lawler [11], which actually solves a more general problem. Suppose each job  $j$  is subject to a nondecreasing penalty function  $f_j(C_j)$  and our objective is to minimize  $\max\{f_j(C_j)\}$ . This problem can be solved as follows. For a single machine, there must be a job that completes at time  $t = \sum p_j$ . Choose the job  $j^*$  such that  $f_{j^*}(t)$  is the smallest among all unscheduled jobs. Schedule job  $j^*$  to complete at time  $t$ . This reduces the problem to a set of  $n - 1$  jobs to which the same rule applies.

It can be shown that the schedule obtained has the smallest  $\max\{f_j(C_j)\}$ . Returning to the maximum weighted tardiness problem, define for each job  $j$  a penalty function  $f_j(C_j)$ , where  $f_j(C_j)$  is defined as

$$f_j(C_j) = \begin{cases} 0 & \text{if } C_j \leq d_j \\ w_j(C_j - d_j) & \text{if } C_j > d_j \end{cases}$$

Clearly,  $f_j(C_j)$  is a nondecreasing function. Thus, we can apply Lawler's algorithm to find a schedule with the minimum  $\max\{w_j T_j\}$ .

Extending the notation of Graham *et al.* [10], we use  $1 \parallel \gamma_2 \mid \gamma_1$  to denote the single-machine bi-criteria scheduling problem, where  $\gamma_1$  is the primary criterion and  $\gamma_2$  is the secondary criterion. For example,  $1 \parallel \sum U_j \mid T_{\max}$  denotes the problem where the primary criterion is the maximum tardiness and the secondary criterion is the number of tardy jobs. As another example,  $1 \parallel \max\{w_j T_j\} \mid \sum U_j$  denotes the problem where the primary criterion is the number of tardy jobs and the secondary criterion is the maximum weighted tardiness.

The problem  $1 \parallel T_{\max} \mid \sum U_j$  has been studied by Shanthikumar [18], who gave a branch-and-bound algorithm for the general problem. As well, a polynomial-time algorithm was given when the set of tardy jobs is specified. Chen and Bulfin [4] studied the problem  $1 \parallel \sum U_j \mid T_{\max}$  and gave a branch-and-bound algorithm for the general problem. The complexity of both problems remain open [3, 12]. Further results about bi-criteria scheduling problems can be found in [3, 5, 12, 8, 20, 6, 7, 19]. The survey paper by Lee and Vairaktarakis [12] gave the complexity of many bi-criteria scheduling problems. They noted that the complexity of the following problems remained open:

- (1)  $1 \parallel T_{\max} \mid \sum U_j$ .
- (2)  $1 \parallel \sum U_j \mid T_{\max}$ .
- (3)  $1 \parallel \max\{f_j(C_j)\} \mid \sum U_j$ .
- (4)  $1 \parallel \sum U_j \mid \max\{f_j(C_j)\}$ .
- (5)  $1 \parallel \sum C_j \mid \sum U_j$ .
- (6)  $1 \parallel \sum T_j \mid \sum U_j$ .

Problems (5) and (6) had recently been shown to be NP-hard [9]. In this paper we show that (3) and (4) are both NP-hard. Indeed, (3) and (4) are NP-hard even when the penalty function  $f_j$  for each job  $j$  is simply the weighted tardiness of job  $j$ . Despite much effort spent on (1) and (2), the complexity of these two problems remain open.

Lee and Vairaktarakis [12] further differentiate bi-criteria scheduling problems between hierarchical problems and dual criteria problems. In a dual criteria problem, we merely require the primary criterion to satisfy the constraint that  $\gamma_1 \leq \alpha$ , where  $\alpha$  is an input parameter. A hierarchical problem

is a special case of dual criteria problem where  $\alpha$  is stipulated to be the minimum value of  $\gamma_1$  (and hence is not an input parameter). The problems mentioned up to now are all hierarchical problems.

In this paper we consider both dual criteria and hierarchical problems. A *feasible schedule* for a problem  $1 \parallel \gamma_2 \mid \gamma_1$  (be it a dual criteria problem or a hierarchical problem) is a schedule in which the primary criterion is satisfied. An *optimal schedule* is a feasible schedule that minimizes the secondary criterion. For a given set of jobs, let  $k^*$  be the minimum number of tardy jobs,  $T^*$  be the minimum  $T_{\max}$  and  $T_w^*$  be the minimum value of  $\max\{w_j T_j\}$ . In this paper we consider the following problems:

- P1:**  $1 \parallel \sum U_j \mid T_{\max} \leq T$ , where  $T \geq T^*$
- P2:**  $1 \parallel T_{\max} \mid \sum U_j \leq k$ , where  $k \geq k^*$
- P3:**  $1 \parallel \sum U_j \mid T_{\max} = T^*$ , or  $1 \parallel \sum U_j \mid T_{\max}$  for simplicity
- P4:**  $1 \parallel T_{\max} \mid \sum U_j = k^*$ , or  $1 \parallel T_{\max} \mid \sum U_j$  for simplicity
- P5:**  $1 \parallel \sum U_j \mid \max\{w_j T_j\} \leq T_w$ , where  $T_w \geq T_w^*$
- P6:**  $1 \parallel \max\{w_j T_j\} \mid \sum U_j \leq k$ , where  $k \geq k^*$
- P7:**  $1 \parallel \sum U_j \mid \max\{w_j T_j\} = T_w^*$ , or  $1 \parallel \sum U_j \mid \max\{w_j T_j\}$  for simplicity
- P8:**  $1 \parallel \max\{w_j T_j\} \mid \sum U_j = k^*$ , or  $1 \parallel \max\{w_j T_j\} \mid \sum U_j$  for simplicity

Note that P1, P2, P5 and P6 are dual criteria problems, while P3, P4, P7 and P8 are hierarchical problems.

Problems P1, P3, P5 and P7 are related to the problem  $1 \mid \bar{d}_j \mid \sum U_j$ . In the problem  $1 \mid \bar{d}_j \mid \sum U_j$ , each job  $j$  is given an additional deadline  $\bar{d}_j$  which must be met, and the goal is to minimize the number of tardy jobs. For P1 and P3, we can view each job  $j$  as having a deadline  $\bar{d}_j = d_j + T$  and  $\bar{d}_j = d_j + T^*$ , respectively. For P5 and P7, each job  $j$  has a deadline  $\bar{d}_j = d_j + T_w/w_j$  and  $\bar{d}_j = d_j + T_w^*/w_j$ , respectively. In later sections of this paper, we will use the problem  $1 \mid \bar{d}_j \mid \sum U_j$  as a substitute for problems P1, P3, P5 and P7, where the deadline of each job is set appropriately.

In this paper we first consider the complexity relationships between the above eight problems. We then show that problems P7 and P8 are NP-hard, which implies that problems (3) and (4) in the list of open problems of Lee and Vairaktarakis [12] are also NP-hard. These results are given in Section 2. We then develop optimal algorithms for three special cases of problems P1 to P8, which will be presented in Section 3. Finally, we propose several heuristics for the general problem and their effectiveness is studied empirically. Our experiment indicates that one heuristic performs extremely well compared to optimal solutions. These results will be described in Section 4. The last section concludes.

## 2 Complexity results

We first study the complexity relationships between the eight problems. Given two problems  $\Pi_1$  and  $\Pi_2$ , we use  $\Pi_1 \Rightarrow \Pi_2$  to denote that a polynomial-time algorithm for  $\Pi_1$  implies a polynomial-time algorithm for  $\Pi_2$ . We use  $\Pi_1 \equiv \Pi_2$  to denote that  $\Pi_1 \Rightarrow \Pi_2$  and  $\Pi_2 \Rightarrow \Pi_1$ .

**Lemma 1**  $P1 \Rightarrow P3, P2 \Rightarrow P4, P5 \Rightarrow P7$  and  $P6 \Rightarrow P8$ .

**Proof :** It follows from the fact that a hierarchical problem is a special case of a dual criteria problem.  $\square$

**Lemma 2**  $P1 \equiv P3$  and  $P5 \equiv P7$ .

**Proof :** By Lemma 1,  $P1 \Rightarrow P3$  and  $P5 \Rightarrow P7$ . Thus, all we need to show is that  $P3 \Rightarrow P1$  and  $P7 \Rightarrow P5$ . We first show that  $P3 \Rightarrow P1$ . Suppose we have an instance I of P1 consisting of a set of  $n$  jobs and a parameter  $T$ , where  $T \geq T^*$ . If  $T = T^*$ , then this instance is also an instance of P3. So we can use the polynomial-time algorithm for P3 to solve I. Otherwise, we will solve I as follows.

Without loss of generality, we may assume that the jobs in I are indexed in ascending order of due dates; i.e.,  $d_1 \leq d_2 \leq \dots \leq d_n$ . We construct an instance II of P3 as follows: II consists of all the jobs in I plus an additional job  $n+1$ . Job  $n+1$  has processing time  $p_{n+1} = T + (1 + d_n - \sum_{i=1}^n p_i)$  and due date  $d_{n+1} = 1 + d_n$ . Using the algorithm for P3, we can find an optimal schedule  $S_{II}$  for II in polynomial time. Let  $S_I$  be the schedule obtained from  $S_{II}$  by deleting job  $n+1$  from  $S_{II}$  and compacting the schedule if possible. Clearly,  $S_I$  can be obtained in polynomial time. We now show that  $S_I$  is an optimal schedule for I.

Let  $T^*(II)$  be the minimum  $T_{\max}$  for the jobs in II. We first show that  $T^*(II) = T$ . As mentioned in Section 1, the schedule obtained by the EDD rule minimizes the maximum tardiness. Since job  $n+1$  has the largest due date in II, it must be scheduled as the last job in the EDD schedule. Thus,  $T_{n+1} = \max(0, \sum_{j=1}^{n+1} p_j - d_{n+1}) = T$ . If we discard job  $n+1$  from this EDD schedule, the resulting schedule will still be a EDD schedule for the jobs in I. By assumption, the maximum tardiness of the jobs in I is at most  $T$ . Therefore, the maximum tardiness obtained in this EDD schedule is at most  $T$ . Thus, we have  $T^*(II) = T_{n+1} = T$ .

Since  $T^*(II) = T$ , in any feasible schedule for II, the completion time of any job  $i$ ,  $1 \leq i \leq n$ , cannot be greater than  $d_i + T$  which is strictly less than  $d_{n+1} + T = \sum_{j=1}^{n+1} p_j$ . Thus, the only job that can be scheduled last is job  $n+1$ . In other words, all the jobs in I must be scheduled before job  $n+1$  and have tardiness at most  $T$ . So  $S_I$  is also a feasible schedule for the instance I of problem P1. Thus, there is a one-to-one correspondence between the feasible schedules for I and the feasible schedules for II. The number of tardy jobs in a feasible schedule for I is exactly one less than that for II (since job  $n+1$  is always the last one to complete and is always tardy). Therefore, the optimal schedule  $S_{II}$  for II must correspond to the optimal schedule  $S_I$  for I.

We now show that  $P7 \Rightarrow P5$ . Suppose I is an instance of P5 consisting of a set of  $n$  jobs and a parameter  $T_w$ , where  $T_w \geq T_w^*$ . If  $T_w = T_w^*$ , then this instance is also an instance of P7, and hence I can be solved by the polynomial-time algorithm for P7. Otherwise, we will solve I as follows.

Let the jobs in I be indexed in ascending order of due dates (i.e.,  $d_1 \leq d_2 \leq \dots \leq d_n$ ) and let  $w^*$  be the smallest weight among the  $n$  jobs in I. We construct an instance II of P7 as follows: II consists of all the jobs

in I plus an additional job  $n + 1$ , which has processing time  $p_{n+1} = \frac{T_w}{w^*} + (1 + d_n - \sum_{i=1}^n p_i)$ , due date  $d_{n+1} = 1 + d_n$  and weight  $w_{n+1} = w^*$ . Using the algorithm for P7, we can find an optimal schedule  $S_{II}$  for II in polynomial time. Let  $S_I$  be the schedule obtained from  $S_{II}$  by deleting job  $n + 1$  from  $S_{II}$  and compacting the schedule if possible. We will show that  $S_I$  is an optimal schedule for I.

Let  $T_w^*(II)$  be the minimum value of  $\max\{w_j T_j\}$  for the jobs in II. We first show that  $T_w^*(II) = T_w$ . If job  $n + 1$  is scheduled last in  $S_{II}$ , then  $w_{n+1} T_{n+1} = T_w$ . On the other hand, if any job  $i$ ,  $1 \leq i \leq n$ , is scheduled last in  $S_{II}$ , then  $w_i T_i > T_w$ . Thus, job  $n + 1$  must be the last job scheduled in  $S_{II}$ . By assumption, the minimum value of  $\max\{w_j T_j\}$  for the jobs in I is at most  $T_w$ . Thus,  $T_w^*(II) = T_w$ . So  $S_I$  is also a feasible schedule for the instance I of problem P5. Consequently, there is a one-to-one correspondence between the feasible schedules for I and the feasible schedules for II, and hence the optimal schedule  $S_{II}$  for II must correspond to the optimal schedule  $S_I$  for I.  $\square$

**Lemma 3**  $P1 \equiv P2$  and  $P5 \equiv P6$ .

**Proof :** We first prove that  $P1 \Rightarrow P2$ . Suppose there is an algorithm A that takes a parameter  $T$  and a set of  $n$  jobs, and outputs a schedule that minimizes  $\sum U_j$  such that the schedule has maximum tardiness at most  $T$ , where  $T \geq T^*$ . We will show that we can use binary search to find the minimum  $T_{\max}$  among all schedules such that  $\sum U_j \leq k$  for a given parameter  $k \geq k^*$ .

Since  $k \geq k^*$ , there is always a feasible schedule (and hence an optimal schedule) for an instance of P2. The maximum tardiness of any feasible schedule must lie between  $T^*$  and  $\sum_{j=1}^n p_j$ , and hence the optimal  $T_{\max}$  must also lie in this range. Given  $k_1 \geq k_2 \geq k^*$ , let  $T_1$  and  $T_2$  be the optimal  $T_{\max}$  to the problems  $1 \parallel T_{\max} \mid \sum U_j \leq k_1$  and  $1 \parallel T_{\max} \mid \sum U_j \leq k_2$ , respectively. Then we must have  $T_1 \leq T_2$ , since any feasible schedule for the latter problem is also a feasible schedule for the former problem. Thus, as  $T$  increases, the number of tardy jobs is non-increasing, and conversely. Therefore, there must be a smallest integer  $T_O$ ,  $T^* \leq T_O \leq \sum_{j=1}^n p_j$ , such that the number of tardy jobs in an optimal solution to the problem  $1 \parallel \sum U_j \mid T_{\max} \leq T_O$  is at most  $k$ . We know that  $T^*$  can be obtained by the EDD schedule in  $O(n \log n)$  time. For each value of  $T$  obtained in the binary search, we call algorithm A to find the optimal number of tardy jobs. We then search the upper half or the lower half of the range, depending on whether the number of tardy jobs returned by algorithm A is larger than or at most  $k$ . If we use binary search to find  $T_O$ , we need to call algorithm A at most  $\lceil \log(\sum_{j=1}^n p_j) \rceil$  times. So the overall running time is still polynomial if A is a polynomial-time algorithm.

We now show that  $P2 \Rightarrow P1$ . Suppose there is an algorithm B that takes a parameter  $k$  and a set of  $n$  jobs, and outputs a schedule that minimizes  $T_{\max}$  such that the schedule has at most  $k$  tardy jobs. We will show that we can use binary search to find the minimum number of tardy jobs such that the schedule has maximum tardiness at most  $T$  for a given  $T \geq T^*$ .

Since  $T \geq T^*$ , there is always a feasible schedule (and hence an optimal schedule) for an instance of P1. The number of tardy jobs in any feasible schedule must lie between  $k^*$  and  $n$ , and hence the optimal number of tardy jobs must also lie in this range. Given  $T_1 \geq T_2 \geq T^*$ , let  $k_1$  and  $k_2$  be the optimal number of tardy jobs to the problems  $1 \parallel \sum U_j \mid T_{\max} \leq T_1$  and  $1 \parallel \sum U_j \mid T_{\max} \leq T_2$ , respectively. Then we must have  $k^* \leq k_1 \leq k_2 \leq n$ , since a feasible schedule for the latter problem is also a feasible schedule for the former. Thus, as the number

of tardy jobs increases,  $T$  is non-increasing, and conversely. Therefore, there must be a smallest integer  $k_0$ ,  $k^* \leq k_0 \leq n$ , such that the maximum tardiness in an optimal solution to the problem  $1 \parallel T_{\max} \mid \sum U_j \leq k_0$  is at most  $T$ . We know that  $k^*$  can be obtained by the Hodgson-Moore algorithm in  $O(n \log n)$  time. For each value of  $k$  obtained in the binary search, we call algorithm B to find the optimal  $T_{\max}$ . We then search the upper half or the lower half of the range, depending on whether the  $T_{\max}$  returned by algorithm B is larger than or at most  $T$ . If we use binary search to find  $k_0$ , we need to call algorithm B at most  $\lceil \log n \rceil$  times. So the overall running time is still polynomial if B is a polynomial-time algorithm.

The proof of  $P5 \equiv P6$  follows the same arguments as above. □

From Lemmas 1, 2 and 3, we have the following theorem.

**Theorem 1**  $P3 \equiv P1 \equiv P2 \Rightarrow P4$  and  $P7 \equiv P5 \equiv P6 \Rightarrow P8$ .

Next, we consider the complexity of problems P7 and P8.

**Theorem 2** *Problems P7 and P8 are both NP hard.*

**Proof :** Lawler has shown that the problem  $1 \mid \bar{d}_j \mid \sum U_j$  is NP hard by a reduction from the partition problem; see the proof in [21]. Our proof is based on his reduction. For completeness, we will sketch his reduction first. (Note that the reason we follow his reduction, rather than directly from  $1 \mid \bar{d}_j \mid \sum U_j$ , is that the proof immediately shows that P8 is also NP-hard.)

An instance of the partition problem has  $n$  integers  $a_1, a_2, \dots, a_n$  with  $\sum a_i = 2A$ . The problem is to decide whether there is a subset  $S$  of the index set  $\{1, \dots, n\}$  such that  $\sum_{i \in S} a_i = A$ . From an instance of the partition problem, we construct an instance of  $1 \mid \bar{d}_j \mid \sum U_j$  as follows. There will be  $4n$  jobs, with each job  $i$  having a processing time  $p_i$ , a due date  $d_i$  and a deadline  $\bar{d}_i$ , see Table 1<sup>1</sup>, where  $x_{i-1} = 3A \cdot 2^{i-1}$  and  $P_{i-1} = \sum_{k=1}^n (2^n + 1)x_{k-1} + \sum_{k=1}^{i-1} (2^n + 1)x_{k-1}$ . The problem is to decide whether there is a feasible schedule with at most  $2n$  tardy jobs.

Table 1: The jobs in the reduction.

$j$	$p_j$	$d_j$	$\bar{d}_j$
$i (1 \leq i \leq n)$	$x_{i-1} + a_i$	$\left( \sum_{k=1}^i x_{k-1} \right) + A$	$P_{i-1} + x_{i-1} + A$
$n + i (1 \leq i \leq n)$	$x_{i-1}$	$\left( \sum_{k=1}^i x_{k-1} \right) + A$	$P_{i-1} + x_{i-1} + 2^n \cdot x_{i-1} - A$
$2n + i (1 \leq i < n)$	$2^n \cdot x_{i-1}$	$\left( \sum_{k=1}^n x_{k-1} + \sum_{k=1}^i 2^n \cdot x_{k-1} \right) + A$	$P_{i-1} + 2^n \cdot x_{i-1} - A$
$3n$	$2^n \cdot x_{n-1}$	$\left( \sum_{k=1}^n x_{k-1} + \sum_{k=1}^n 2^n \cdot x_{k-1} \right) - A$	$P_{n-1} + 2^n \cdot x_{n-1} - A$
$3n + i (1 \leq i < n)$	$2^n \cdot x_{i-1} - 2a_i$	$\left( \sum_{k=1}^n x_{k-1} + \sum_{k=1}^i 2^n \cdot x_{k-1} \right) + A$	$P_{i-1} + x_{i-1} + 2^n \cdot x_{i-1} - A$
$4n$	$2^n \cdot x_{n-1} - 2a_n$	$\left( \sum_{k=1}^n x_{k-1} + \sum_{k=1}^n 2^n \cdot x_{k-1} \right) - A$	$P_{n-1} + x_{n-1} + 2^n \cdot x_{n-1} - A$

<sup>1</sup>There is a typo in [21]. The deadline of job  $i$ ,  $1 \leq i \leq n$ , should be  $P_{i-1} + x_{i-1} + A$  instead of  $P_{i-1} + x_{i-1} - A$

The idea of the reduction is that, for each  $i = 1, \dots, n$ , the jobs  $\{i, 3n + i\}$  and  $\{n + i, 2n + i\}$  form two pairs, one of which goes into the on-time set and the other one goes into the tardy set. One can show that there is a schedule with exactly  $2n$  tardy jobs such that every job meets its deadline if and only if there is a solution to the partition instance.

We first show that the decision version of the problem P7 is NP complete. We will use almost the same reduction as in Lawler's proof, but instead of having a deadline, we now have a weight for each job. We also introduce an additional job 0. Specifically, let  $T_w = \max\{\bar{d}_j - d_j\}$ ,  $1 \leq j \leq 4n$ , where  $\bar{d}_j$  and  $d_j$  have the same values as above. We create  $4n + 1$  jobs,  $0, 1', \dots, 4n'$ . We let  $p_0 = T_w = \max\{\bar{d}_j - d_j\}$ ,  $d_0 = 0$  and  $w_0 = 1$ . For  $1 \leq j' \leq 4n$ , let  $p_{j'} = p_j$ ,  $w_{j'} = T_w/(\bar{d}_j - d_j)$  and  $d_{j'} = p_0 + d_j$ , where  $p_j$  and  $d_j$  have the same values as above. The problem is to decide whether there is a feasible schedule with at most  $2n + 1$  tardy jobs such that  $\max\{w_j T_j\}$  is at most  $T_w$ ?

Because job 0 has due date 0, its weighted tardiness has to be at least  $w_0 p_0 = T_w$  in any schedule. Thus,  $\max\{w_j T_j\}$  is at least  $T_w$ . On the other hand, if we schedule job 0 first and then schedule the remaining jobs in an order corresponding to a feasible schedule for the instance of the  $1 \parallel \sum U_j$  problem, then one can easily show that every job has a weighted tardiness at most  $T_w$ . Thus,  $T_w^*$  is exactly  $T_w$ . Hence, in an optimal schedule, job 0 has to be scheduled before any other job. For all other tardy jobs, we must have  $w_i' T_i' = w_i'(C_i' - d_i') \leq T_w$ , where  $C_i'$  is the completion time of job  $i'$  in the optimal schedule. Plugging in the values of  $w_i'$  and  $d_i'$ , we have  $C_i' \leq (\bar{d}_i + p_0)$ . This means that for each job  $i'$ ,  $w_i' T_i' \leq T_w$  if and only if  $i'$  completes before  $\bar{d}_i + p_0$ . Using almost the same argument, we can show that there is a schedule with at most  $2n + 1$  tardy jobs such that  $C_i' \leq \bar{d}_i + p_0$  for every  $1 \leq i' \leq 4n$  if and only if there is a solution to the partition instance.

Thus, problem P7 is NP-hard. On the other hand, using the Hodgson-Moore algorithm, we can show that the minimum number of tardy jobs for these  $4n + 1$  jobs is exactly  $2n + 1$ . So the same argument shows that problem P8 is also NP-hard.  $\square$

Since problems P7 and P8 are special cases of  $1 \parallel \sum U_j \mid f_{\max}$  and  $1 \parallel f_{\max} \mid \sum U_j$ , respectively, we immediately have the following corollary.

**Corollary 4** *The problems  $1 \parallel \sum U_j \mid f_{\max}$  and  $1 \parallel f_{\max} \mid \sum U_j$  are both NP-hard.*

By Theorems 1 and 2, we have the following corollary.

**Corollary 5** *The problems P5 to P8 are all NP-hard.*

### 3 Optimal algorithms for special cases

In this section we will present polynomial-time algorithms to solve three special cases of problems P1 to P8. We will concentrate on P1 and P5 only. By Theorem 1, a polynomial-time algorithm for P1 yields a polynomial-time algorithm for P2, P3 and P4, whereas a polynomial-time algorithm for P5 yields a polynomial-time algorithm for P6, P7 and P8. Thus, it will be sufficient to consider P1 and

P5 only. We shall be studying  $1 \mid \bar{d}_j \mid \sum U_j$  as a substitute for P1 and P5, by setting the deadline of each job appropriately; i.e.,  $\bar{d}_j = d_j + T$  for P1 and  $\bar{d}_j = d_j + \frac{T w_j}{w_j}$  for P5.

In this section we assume that jobs are indexed in ascending order of their due dates; i.e.,  $d_1 \leq d_2 \leq \dots \leq d_n$ . The constraints of the three special cases are given as follows.

**Case 1:**  $d_i \leq d_j$  implies  $\bar{d}_i \leq \bar{d}_j$  and  $p_i \leq p_j$ .

**Case 2:**  $p_i \geq p_j$  implies  $sl_i \leq sl_j$ , where  $sl_k = d_k - p_k$  is the slack of job  $k$ .

**Case 3:** There is an integer  $m$ ,  $1 < m < n$ , such that (1) for  $1 \leq i < j \leq m$ ,  $d_i \leq d_j$  implies  $\bar{d}_i \leq \bar{d}_j$  and  $p_i \leq p_j$ , (2)  $\max_{j=1}^m \{\bar{d}_j\} \leq \min_{j=m+1}^n \{\bar{d}_j\}$ , and (3) for  $m < i, j \leq n$ ,  $p_i \geq p_j$  implies  $sl_i \leq sl_j$ .

Case 1 requires that the due dates, deadlines and processing times be *agreeable*. That is, there is an ordering of jobs such that

$$\begin{aligned} d_1 &\leq d_2 \leq \dots \leq d_n, \\ \bar{d}_1 &\leq \bar{d}_2 \leq \dots \leq \bar{d}_n, \\ p_1 &\leq p_2 \leq \dots \leq p_n. \end{aligned}$$

For Case 2, the condition “ $p_i \geq p_j$  implies  $sl_i \leq sl_j$ ” can be satisfied by the condition “ $d_i \leq d_j$  implies  $p_i \geq p_j$ ”. This special case corresponds to the situation when there is an ordering of jobs such that

$$\begin{aligned} d_1 &\leq d_2 \leq \dots \leq d_n, \\ \bar{d}_1 &\leq \bar{d}_2 \leq \dots \leq \bar{d}_n, \\ p_1 &\geq p_2 \geq \dots \geq p_n. \end{aligned}$$

Case 3 is a combination of Cases 1 and 2; i.e., there is an integer  $m$  such that the first  $m$  jobs satisfy the condition of Case 1, the last  $n - m$  jobs satisfy the condition of Case 2, and the largest deadline in the first  $m$  jobs is less than or equal to the smallest deadline in the last  $n - m$  jobs..

In each case we assume that the set of jobs has a feasible schedule; i.e., there is a schedule such that all jobs can meet their deadlines. We will give a polynomial-time algorithm for each case and prove that our algorithm is optimal. The three cases are given in the next three subsections.

### 3.1 Case 1

In this subsection we assume that  $d_i \leq d_j$  implies  $\bar{d}_i \leq \bar{d}_j$  and  $p_i \leq p_j$ . Let  $TS$  be a subset of  $JS$ , where  $JS$  is a set of  $n$  jobs,  $1, \dots, n$ . A *tight schedule* for  $JS$  with respect to  $TS$  is a schedule such that: (1) the jobs in  $JS \setminus TS$  are scheduled in nondecreasing order of their due dates, (2) all jobs in  $TS$  are scheduled in nondecreasing order of their deadlines, and (3) for each job  $i \in TS$  scheduled

immediately before a job  $j \in JS \setminus TS$ , we must have  $C_i \leq \bar{d}_i$  and  $C_i + p_j > \bar{d}_i$ , where  $C_i$  is the completion time of job  $i$ . In other words, we schedule jobs in  $TS$  as late as possible without missing their deadlines.

A tight schedule with respect to  $TS$  can be obtained as follows. We schedule jobs in  $JS$  backwards, starting at time  $t = \sum p_j$ . If the job in  $TS$  with the largest deadline can be scheduled to complete at time  $t$  (i.e., without missing its deadline), then schedule it to complete at time  $t$ . Otherwise, schedule the job in  $JS \setminus TS$  with the largest due date to complete at time  $t$ . Iterate this process with the remaining jobs starting at time  $t'$ , where  $t'$  is the starting time of the previously scheduled job.

It is easy to see that for any optimal schedule, there is another optimal schedule that is tight. Thus, it is sufficient to concentrate on tight schedules only. The algorithm given below solves Case 1.

### Algorithm 1

**Input:** A set of  $n$  jobs satisfying the condition of Case 1

**Output:** A schedule  $S$  that minimizes  $\sum U_j$  subject to the condition that every job meets its deadline

1. Schedule the jobs in EDD order. If there is a tie, schedule the one with the smallest processing time first. Let the schedule be  $S$
2. Let  $TS = \emptyset$  be the initial tardy set
3. Repeat until every tardy job in  $S$  is contained in  $TS$ :
  - (a) let  $i \notin TS$  be the first tardy job in  $S$
  - (b)  $TS = TS \cup \{i\}$
  - (c) reschedule the  $n$  jobs such that  $S$  is a tight schedule with respect to  $TS$

**Theorem 3** *The schedule produced by Algorithm 1 is optimal for the problem  $1 \mid \bar{d}_j \mid \sum U_j$  under the condition that  $d_i \leq d_j$  implies  $\bar{d}_i \leq \bar{d}_j$  and  $p_i \leq p_j$ .*

**Proof :** Assume that  $d_i \leq d_{i+1}$  for  $1 \leq i \leq n - 1$ . Let  $TS = \{i_1, \dots, i_k\}$ ,  $i_1 < i_2 < \dots < i_k$ , be the tardy set obtained by Algorithm 1. We prove the theorem by showing that any feasible and tight schedule must have at least  $k$  tardy jobs. It is sufficient to prove that (1) there must be at least  $m$  tardy jobs from the job set  $\{1, 2, 3, \dots, i_m\}$ ,  $1 \leq m \leq k$ , in any feasible schedule, and (2) if a feasible schedule chooses  $\{j_1, j_2, \dots, j_m\}$  to be the tardy jobs, then we must have  $d_{j_k} \leq d_{i_k}$ ,  $1 \leq k \leq m$ . We prove this by induction on  $m$ .

By our algorithm,  $i_1$  is the first tardy job in the EDD schedule. Hence, there must be at least one tardy job from the jobs  $1, \dots, i_1$  in any feasible schedule. By assumption,  $i_1$  has the largest due date. Thus, (1) and (2) are true for  $m = 1$ .

Suppose it holds for  $m - 1$ . Let  $S'$  be any feasible and tight schedule. By the induction hypothesis, there are at least  $m - 1$  tardy jobs from the jobs  $1, \dots, i_{m-1}$ . Let these tardy jobs be  $TS' = \{j_1, j_2, \dots, j_{m-1}\}$ . We will show, by contradiction, that there is at least one more tardy job from  $1, \dots, i_m$  in  $S'$ . Suppose all jobs  $x$ ,  $1 \leq x \leq i_m$  and  $x \notin TS'$ , are on time in  $S'$ . Since  $i_m$  has the largest due date and since  $S'$  is a tight schedule, we

may assume that all other on-time jobs  $x < i_m$  are scheduled before  $i_m$ . Let  $TS'_1$  be the subset of  $TS'$  consisting of all jobs scheduled after  $i_m$  in  $S'$ . Since  $i_m$  is on time, we have

$$C'_{i_m} = \sum_{j=1}^{i_m} p_j - \sum_{j \in TS'_1} p_j \leq d_{i_m}$$

where  $C'_{i_m}$  is the completion time of job  $i_m$  in  $S'$ . Consider the schedule  $S$  at the beginning of the  $m$ -th iteration of Algorithm 1. For each job  $j_x \in TS'$ , by the induction hypothesis, we have a tardy job  $i_x \in TS$  such that  $d_{i_x} \geq d_{j_x}$  which implies  $\bar{d}_{i_x} \geq \bar{d}_{j_x}$  and  $p_{i_x} \geq p_{j_x}$ . Thus, if job  $j_x$  can be scheduled after  $i_m$  in  $S'$ , job  $i_x$  can also be scheduled after  $i_m$  in  $S$  without missing its deadline. Let  $TS_1$  be the set of tardy jobs scheduled after  $i_m$  in  $S$ , then we must have  $\sum_{j \in TS_1} p_j \geq \sum_{j \in TS'_1} p_j$ . Thus,

$$C_{i_m} = \sum_{j=1}^{i_m} p_j - \sum_{j \in TS_1} p_j \leq \sum_{j=1}^{i_m} p_j - \sum_{j \in TS'_1} p_j \leq d_{i_m}$$

where  $C_{i_m}$  is the completion time of job  $i_m$  in  $S$ . This is a contradiction, since  $i_m$  is a tardy job at the beginning of the  $m$ -th iteration. Thus,  $S'$  also has  $m$  tardy jobs. Since  $i_m$  has the largest due date among the jobs  $1, 2, 3, \dots, i_m$ , (2) follows immediately.  $\square$

**Corollary 6** *Under the condition stated in Theorem 3, problems P1 to P8 can be solved in polynomial time.*

### 3.2 Case 2

In this subsection we assume that  $p_i \geq p_j$  implies  $sl_i \leq sl_j$ , where  $sl_k = d_k - p_k$  is the slack of job  $k$ . As noted above, this case includes the case where  $d_i \leq d_j$  implies  $p_i \geq p_j$ . The algorithm to solve Case 2 is given below.

#### Algorithm 2

**Input:** A set of  $n$  jobs satisfying the condition of Case 2

**Output:** A schedule with the minimum  $\sum U_j$  such that every job meets its deadline

1.  $t \leftarrow \sum_{j=1}^n p_j$
2. Repeat until all jobs have been scheduled:
  - let  $i$  be the unscheduled job with the largest due date
  - if  $d_i \geq t$ 
    - schedule job  $i$  in the time interval  $(t - p_i, t]$
    - $t \leftarrow t - p_i$
  - else

let  $j$  be the unscheduled job with the largest processing time such that  $\bar{d}_j \geq t$ . In case of a tie, choose the one with the smallest due date  
 schedule job  $j$  in the time interval  $(t - p_j, t]$   
 $t \leftarrow t - p_j$

**Theorem 4** *The schedule produced by Algorithm 2 is optimal for the problem  $1 \mid \bar{d}_j \mid \sum U_j$  under the condition that  $p_i \geq p_j$  implies  $s_i \leq s_j$ .*

**Proof :** Given two jobs  $i$  and  $j$  such that  $p_i \geq p_j$ , we say that job  $i$  *dominates* job  $j$  if  $s_i \leq s_j$ . Suppose we have two jobs  $i$  and  $j$  such that job  $i$  dominates job  $j$ . If both jobs become tardy when scheduled to complete at time  $t$  and one of them has to be scheduled to complete at time  $t$ , then in order to minimize the number of tardy jobs, it is sufficient to schedule job  $i$  to complete at time  $t$  (see [4], [6]). In other words, we should choose the non-dominated job. Since at each time  $t$ , the algorithm either schedules an on-time job, or a tardy job that dominates all other jobs whose deadlines are at least  $t$ , the optimality of Algorithm 2 follows immediately.  $\square$

**Corollary 7** *Under the condition stated in Theorem 4, problems P1 to P8 can be solved in polynomial time.*

### 3.3 Case 3

Case 3 is a combination of Case 1 and Case 2; i.e., there is an integer  $m$ ,  $1 < m < n$ , such that the first  $m$  jobs satisfy the condition of Case 1, the last  $n - m$  jobs satisfy the condition of Case 2, and the largest deadline in the first  $m$  jobs is less than or equal to the smallest deadline in the last  $n - m$  jobs.

#### Algorithm 3

**Input:** A set of  $n$  jobs satisfying the condition of Case 3.

**Output:** A schedule  $S$  with minimum  $\sum U_j$  such that every job meets its deadline

1.  $JS \leftarrow \{1, \dots, n\}$ ,  $JS_1 \leftarrow \{1, \dots, m\}$  and  $JS_2 \leftarrow \{m + 1, \dots, n\}$
2. Apply Algorithm 1 to  $JS_1$  to obtain a schedule  $S_1$  with tardy set  $TS_1$
3. Let  $S$  be the tight schedule of  $JS$  with respect to  $TS_1$
4.  $TS \leftarrow TS_1$  and  $t \leftarrow \sum_{j=1}^n p_j$
5. Repeat until each job in  $S$  is either on time or a tardy job in  $TS$ 
  - (a) scan  $S$  backwards from  $t$ . let  $i$  be the first tardy job in  $S$  such that  $i \notin TS$
  - (b) let  $t_0$  be the completion time of job  $i$  in  $S$
  - (c) pick a job  $x$  from those jobs scheduled in the interval  $(0, t_0]$  as follows:  
 let  $x_1 \notin TS$  be the job in  $JS_1$  with the largest processing time. if there is a tie, choose the one with the largest due date

let  $x_2$  be the job in  $JS_2$  with the largest processing time such that  $\bar{d}_{x_2} \geq t_0$ . if there is a tie, choose the one with the smallest due date

if  $\bar{d}_{x_1} < t_0$

$x \leftarrow x_2$

else if  $p_{x_1} \geq p_{x_2}$

$x \leftarrow x_1$

else

let  $u \in JS_2$  be the first job scheduled after  $x_1$  in  $S$  such that  $p_u > p_{x_1}$

if there is at least one tardy job  $z \in JS_2$  scheduled between  $x_1$  and  $u$  in  $S$

$x \leftarrow x_1$

else

$x \leftarrow x_2$

(d)  $TS \leftarrow TS \cup \{x\}$

(e)  $t \leftarrow t_0 - p_x$

(f) obtain a tight schedule  $S$  in the time interval  $(0, t]$  with respect to the tardy jobs in  $TS$  that are scheduled before  $t$

6. Output  $S$

**Theorem 5** *The schedule  $S$  produced by Algorithm 3 is optimal for the problem  $1 \mid \bar{d}_i \mid \sum U_j$  if there is an integer  $m$ ,  $1 < m < n$ , such that (1) for  $1 \leq i < j \leq m$ ,  $d_i \leq d_j$  implies  $\bar{d}_i \leq \bar{d}_j$  and  $p_i \leq p_j$ , (2)  $\max_{j=1}^m \{\bar{d}_j\} \leq \min_{j=m+1}^n \{\bar{d}_j\}$ , and (3) for  $m < i, j \leq n$ ,  $p_i \geq p_j$  implies  $sl_i \leq sl_j$ .*

The proof of the above theorem is quite involved and hence will be omitted. It can be found in the website “web.njit.edu/~leung/dual-criteria”. From Theorem 5, we have the following corollary.

**Corollary 8** *Under the conditions stated in Theorem 5, problems P1 to P8 can be solved in polynomial time.*

## 4 Heuristics and experimental results

Since problems P5 and P7 are NP-hard, we will propose fast heuristics for them. We again use the problem  $1 \mid \bar{d}_j \mid \sum U_j$  as a substitute. We note that our heuristics are also applicable to problems P1 and P3. Although problems P6 and P8 are also NP-hard, we have not been able to devise good heuristics for them.

Our heuristics fall into three categories. Heuristics that belong to the first category schedule jobs backwards, starting at time  $t = \sum p_j$ . The heuristics determine by some rules a job to complete at time  $t$ . We then decrement  $t$  by the processing time of the chosen job and iterate the process to schedule the remaining jobs.

Heuristics that belong to the second category first construct an EDD schedule  $S$ , and initialize the tardy set  $TS$  to be the empty set. It then repeats the following until every tardy job in  $S$  is already in  $TS$ : (1) Locate the first tardy job  $i$  in  $S$  that is not in  $TS$ ; (2) From those jobs scheduled before and including  $i$ , pick a job according to some rules and put it into  $TS$ ; (3) Obtain a tight schedule  $S$  with respect to  $TS$ ; (4) If a tardy job becomes on time in  $S$ , delete the job from  $TS$ .

The third type of heuristic, called the *hybrid-scheduling* heuristic, schedules jobs in the same manner as the second type, except that at each iteration we update the tardy set with respect to the jobs scheduled up to and including job  $i$ . This update process is done by a backward scheduling algorithm.

In the next subsection we describe the heuristics in detail. Finally, in subsection 4.2, we report our empirical study and compares the effectiveness between the various heuristics as well as relative to optimal solutions. According to our result, the hybrid-scheduling heuristic is the best among all heuristics and its average performance is within 1% more than the optimal value. If we run all heuristics and output the best solution, then the composite heuristic has average performance within 0.7% more than the optimal value. Our result shows that we can get extremely good solutions within a reasonable amount of time.

## 4.1 Heuristics

The first type of heuristic, *backward-scheduling heuristic*, schedules jobs backwards. Depending on the implementation of step 3(a) in the procedure below, we have two different heuristics. The first heuristic, denoted by LPT-B (Largest Processing Time Backward), picks the job  $i$  with the largest processing time. In case of a tie, choose the one with the smallest due date. The rationale is that there may be many jobs scheduled after  $i$  (in the EDD schedule) that have tardiness smaller than  $p_i$ . By scheduling  $i$  to complete at time  $t$ , these jobs will become on time.

The second heuristic, denoted by LS-B (Largest Score Backward), computes a score for each job and picks the one with the highest score. The *score* of each job  $i$  reflects the number of tardy jobs that can be made on time if  $i$  were scheduled to complete at time  $t$ , and it is computed as follows. Let  $S'$  be the EDD schedule of all unscheduled jobs, starting at time 0. If job  $i$  is tardy in  $S'$ , then the score of  $i$  is defined to be the number of tardy jobs  $j$  scheduled after  $i$  in  $S'$  such that  $T_j \leq p_i$ ; otherwise, its score is this number less 1. Note that the score of job  $i$  is the net decrease of tardy jobs if job  $i$  were scheduled to complete at time  $t$ . The rationale is that by scheduling the job with the highest score, we can make more jobs on time.

There are two ways to break ties in the LS-B heuristic; i.e., when several jobs have the same (highest) score. One way is to choose the one with the largest processing time, denoted by LS-P. Another way is to choose the one with the smallest due date, denoted by LS-D. We implemented both heuristics in our experiment.

### Backward-Scheduling Heuristic

**Input:** A set of  $n$  jobs

**Output:** A feasible schedule if one exists

1.  $t \leftarrow \sum_{j=1}^n p_j$
2.  $JS \leftarrow \{1, 2, \dots, n\}$
3. Repeat until all jobs are scheduled or no job can be scheduled at  $t$ :
  - if there is a job  $i$  in  $JS$  such that  $d_i \geq t$ 
    - schedule  $i$  in  $(t - p_i, t]$
    - delete  $i$  from  $JS$
    - $t \leftarrow t - p_i$
  - else
    - if every job in  $JS$  has deadline at least  $t$ 
      - schedule all jobs in  $JS$  using the Hodgson-Moore algorithm
    - else
      - (a) from among those jobs whose deadline is at least  $t$ , choose a non-dominated job  $i$  according to some rule. schedule  $i$  in the time interval  $(t - p_i, t]$
      - (b) delete  $i$  from  $JS$
      - (c)  $t \leftarrow t - p_i$

The second type of heuristic, *forward-scheduling heuristic*, schedules jobs forward. Depending on the implementation of step 3c in the procedure below, we again have two different heuristics. The first heuristic, denoted by LPT-F (Largest Processing Time Forward), picks the job with the largest processing time that can be scheduled after the current job  $i$ . In case of a tie, pick the one with the largest deadline. The second heuristic, denoted by LDL-F (Largest Deadline Forward), picks the job with the largest deadline that can be scheduled after the current job  $i$ . In case of a tie, pick the one with the largest processing time.

### **Forward-Scheduling Heuristic**

**Input:** A set of  $n$  jobs

**Output:** A feasible schedule if one exists

1. Let  $S$  initially be the EDD schedule
2. Let tardy set  $TS$  initially be empty
3. Repeat until every tardy job in  $S$  is in  $TS$  or no feasible schedule can be found:
  - (a) let  $i$  be the first tardy job in  $S$  such that  $i \notin TS$
  - (b) let job  $i$  completes at time  $t$

- (c) from among those jobs scheduled before and including  $i$ , pick a non-dominated job  $j$  with deadline at least  $t$  according to some rules
- (d)  $TS = TS \cup \{j\}$
- (e) obtain a tight schedule  $S$  with respect to  $TS$
- (f) if a job  $j \in TS$  becomes on time in  $S$ , delete  $j$  from  $TS$

The third type of heuristic, called *hybrid-scheduling heuristic*, schedules jobs forward, but at each iteration the tardy set is updated by a backward scheduling algorithm.

Let  $JS$  be a set of  $n$  jobs,  $1, \dots, n$ . Let  $TS$  be a tardy set of  $JS$  such that if we form a tight schedule of  $JS$  with respect to  $TS$ , then the remaining jobs in  $JS$  will be on time. We define an operation, *update tardy set*  $TS$  for job set  $JS$ , as shown below. The update operation has the effect of reducing the number of tardy jobs in  $TS$  without reducing the possible number of on-time jobs in  $JS$ .

**Update**( $TS, JS$ )

**Input:** A job set  $JS$  and a tardy set  $TS \subseteq JS$

**Output:** A tardy set  $\hat{TS} \subseteq TS$

1.  $t \leftarrow \sum_{i \in JS} p_i$
2. Repeat until every job in  $JS$  is scheduled:
  - (a) let  $i$  be the job in  $JS$  with the largest due date
  - (b) if  $d_i \geq t$ 
    - schedule  $i$  to complete at time  $t$
  - Else
    - let  $CS \subseteq TS$  be the set of unscheduled jobs whose deadline is at least  $t$
    - if  $|CS| = 1$ 
      - schedule the job in  $CS$  to complete at time  $t$
    - Else
      - let  $TS'$  be the unscheduled jobs of  $TS$  and  $TS'' = TS' \setminus CS$
      - let  $JS'$  be the unscheduled jobs (including those in  $TS'$ )
      - obtain a tight schedule  $S'$  of  $JS'$  with respect to  $TS''$
      - let  $x$  be the first tardy job in  $S'$
      - pick the job  $y \in CS$  scheduled before and including  $x$  with the largest processing time
      - schedule  $y$  to complete at time  $t$
  - (c) if there is a job in  $TS$  that becomes on time, delete the job from  $TS$
  - (d) decrement  $t$  by the processing time of the job that was chosen to complete at time  $t$
3.  $\hat{TS} \leftarrow TS$
4. return  $\hat{TS}$

Note that  $y$  must exist, since we know  $x$  is either a job in TS or an on-time job when we tardy all jobs in TS. The hybrid-scheduling heuristic is given as follows.

### Hybrid-Scheduling Heuristic

**Input:** A set of  $n$  jobs

**Output:** A feasible schedule  $S$  if one exists

1. Let JS be the set of  $n$  jobs
2. Let  $S$  initially be the EDD schedule
3. Let tardy set TS initially be empty
4. Repeat until every tardy job in  $S$  is in TS or no feasible schedule can be found:
  - (a) let  $i$  be the first tardy job in  $S$  such that  $i \notin TS$
  - (b) let job  $i$  completes at time  $t$
  - (c) from among those jobs scheduled before and including  $i$ , pick a job  $j$  such that  $\bar{d}_j \geq t$  and such that it has the largest processing time
  - (d)  $TS \leftarrow TS \cup \{j\}$
  - (e)  $TS \leftarrow \text{Update}(TS, \{1, 2, \dots, i\})$
  - (f) obtain a tight schedule  $S$  of JS with respect to TS

One can easily show that at step 4c, if we pick a job  $j \neq i$ , then  $i$  must become on time in the schedule obtained in step 4f.

## 4.2 Experimental results

We perform an empirical study of the heuristics discussed in Section 4.1 for the  $1 \mid \bar{d}_j \mid \sum U_j$  problem. In the next subsection, we describe how we generate the data. Finally, we report the results and analysis in subsection 4.2.2.

### 4.2.1 Data generation

Instances are characterized by three parameters: number of jobs  $n$ , due date range factor  $R$  and tardiness factor  $\tau$ . The factor  $R$  controls the range of the due date distribution, while  $\tau$  provides an indication of the average tightness of the due dates, see also [1], [17], [14]. In our experiment, we let  $n \in \{10, 20, 50, 100, 150, 200\}$ ,  $R \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$  and  $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ .

We investigate two cases, depending on the difference between the due date and the deadline: (1) the difference between  $\bar{d}_j$  and  $d_j$  is a constant, which corresponds to the  $T_{\max}$  criterion, and (2) the difference between  $\bar{d}_j$  and  $d_j$  is a function of  $w_j$ , which corresponds to the  $\max\{w_j T_j\}$  criterion. Instead of generating deadlines randomly, we generate a weight for each job. In the first case every

job has a weight 1. In the second case we generate a weight  $w_j$  for job  $j$  from the uniform distribution  $[1, 10]$ . We then use Lawler's algorithm [11] to compute  $T_w^* = \max\{w_j T_j\}$ . Finally, the deadline is computed as  $\bar{d}_j = d_j + T_w^*/w_j$ .

We also investigate the effect of processing times on the heuristics. We generate two types of instances, one in which the processing time is drawn uniformly from  $[1, 10]$  (which corresponds to the case where the largest and the smallest processing times do not differ by much) and the other in which the processing time is drawn uniformly from  $[1, 50]$  (which corresponds to the case where the largest and the smallest processing times differ a lot).

We generate five instances for each given  $n$ ,  $R$ ,  $\tau$ , processing time range and weight range. For each instance, we first generate the processing time of each job from the uniform distribution of the given processing time range; i.e., either  $[1, 10]$  or  $[1, 50]$ . Then an integer due date  $d_i$  for each job  $i$  is randomly generated from the uniform distribution  $\left[ \sum_{j=1}^n p_j(1 - \tau - R/2), \sum_{j=1}^n p_j(1 - \tau + R/2) \right]$ . Since there are six values of  $n$ , five values of  $R$ , five values of  $\tau$ , two processing time ranges and two weight ranges, a total of 3,000 instances were generated.

## 4.2.2 Empirical results and analysis

We implement all of the algorithms in C++. The running environment is based on the RedHat Linux 7.0 operating system. The PC used is a Pentium II 400Mhz with 128MB RAM. To test the performance of the heuristics relative to the optimal solution, we develop an enumerative algorithm to find the optimal value. We do not try to optimize the running time of the enumerative algorithm, since our objective is only to compare the performance of the heuristics with the optimal solution. We set the time limit of running the enumerative algorithm to seven days. If the algorithm does not terminate in seven days, we deem the enumerative algorithm as not being able to find an optimal solution. In this case the instance will be discarded and it will not be included in the statistics. Out of the 3,000 instances generated, the enumerative algorithm fails to find an optimal solution in only 38 instances.

The heuristics run very fast, in matters of seconds and minutes. The enumerative algorithm takes hours and days to run in some instances. The heuristics are fast enough to be able to meet real-time environment, while the enumerative algorithm most likely can not.

For each instance, we apply the enumerative algorithm and all six heuristics. The results<sup>2</sup> are summarized in Tables 2-5. Tables 2 and 3 give the statistics for the unweighted case with processing time ranges  $[1, 10]$  and  $[1, 50]$ , respectively. Tables 4 and 5 give the statistics for the weighted case with processing time ranges  $[1, 10]$  and  $[1, 50]$ , respectively. In each of these tables, we group instances with  $n = 10, 20$  and  $50$  as small instances and instances with  $n = 100, 150$  and  $200$  as large instances, and there are 375 instances in each group. Statistics are generated for small instances separate from large instances. We also generate statistics for all instances.

---

<sup>2</sup>The raw data and results are available at "web.njit.edu/~leung/dual-criteria"

In each of these tables, the first column “Opt” refers to the enumerative algorithm, the next six columns refer to the six heuristics, and the last column “Comp” refers to the composite algorithm of running all six heuristics and outputs the best solution. The row “# of opt” gives the number of instances in which each algorithm generates an optimal solution. The row “#/xxx” gives the fraction of instances in which each algorithm generates an optimal solution. Note that “Opt” always gives a fraction of 1, since we discard instances that take longer than seven days to run. The row “avg ratio” gives the average performance ratios of the heuristics versus the optimal value, while the row “worst ratio” gives the worst-case ratios.

From the tables we can draw the following conclusions:

- The composite algorithm has the best performance; its worst-case ratio is never more than 1.25 and its average ratio is never more than 1.007. Since all six heuristics run very fast, it is indeed viable to use the composite algorithm in practice.
- For a single heuristic, the hybrid-scheduling heuristic outperforms all other heuristics. Its worst-case ratio is never more than 1.639 and its average ratio is never more than 1.01. The hybrid-scheduling heuristic outperforms just about every heuristic in every category: number of optimal solution found, average ratio and worst-case ratio.
- The performance (both average ratio and worst-case ratio) of the heuristics are quite good.
- All heuristics perform better with small instances than large instances.
- All heuristics perform better with the unweighted case than the weighted case.
- Processing time range does not play an important role in the performance of the heuristics.
- Generally speaking, the backward-scheduling heuristics (LS-P, LS-D and LPT-B) are more effective than the forward-scheduling heuristics (LPT-F and LDL-F).
- Between the two forward-scheduling heuristics, the LPT-F heuristic performs better than the LDL-F heuristic for the unweighted case, while the opposite is true for the weighted case.
- Among the three backward-scheduling heuristics (LS-P, LS-D and LPT-B), both LS-P and LPT-B outperforms LS-D most of the times, while LS-P and LPT-B are comparable with each other.

## 5 Conclusions and recommendations for future studies

In this paper we have studied single-machine scheduling problems with two criteria. We focus on the number of tardy jobs  $\sum U_j$ , the maximum tardiness  $T_{\max}$  and the maximum weighted tardiness  $\max\{w_j T_j\}$ . We study both dual criteria and hierarchical problems. Altogether we have considered eight problems, P1 to P8. If the primary criterion is  $T_{\max}$  or  $\max\{w_j T_j\}$  and the secondary criterion is  $\sum U_j$ , the problems can be viewed as special cases of  $1 \mid \bar{d}_j \mid \sum U_j$ .

We first establish the complexity relationships between these eight problems. We then show that  $1 \parallel \sum U_j \mid \max\{w_j T_j\}$  and  $1 \parallel \max\{w_j T_j\} \mid \sum U_j$  are both NP-hard. These two results partially answer

two of the open questions posed by Lee and Vairaktarakis [12]. We have not been able to devise a pseudo-polynomial time algorithm for the two problems, nor can we show that they are strongly NP-hard. For future research, we think it is worthwhile to settle this issue. Despite much efforts spent on  $1 \parallel \sum U_j \mid T_{\max}$  and  $1 \parallel T_{\max} \mid \sum U_j$ , their complexity remain open. Although we cannot prove it, we conjecture that they are both NP-hard. It will be worthwhile to settle this issue in the future.

We give polynomial-time algorithms for three special cases of  $1 \mid \bar{d}_j \mid \sum U_j$ , which yield polynomial-time algorithms for problems P1 to P8. For the general problem, we propose several fast heuristics. We perform empirical study to get a feel for the effectiveness of our heuristics. According to our result, the hybrid-scheduling heuristic gives extremely good solutions within a reasonable amount of time. The average ratio of the hybrid-scheduling heuristic is never more than 1.01, while the worst-case ratio is no more than 1.639. If time permits, we should run all six heuristics and outputs the best solution. The composite algorithm has even better statistics: average ratio no more than 1.007 and worst-case ratio no more than 1.25.

**Acknowledgments:** The authors gratefully acknowledge the support by the National Science Foundation through grant DMI-0300156. Computing facilities are provided by the PC Cluster Lab of the CS Department at NJIT, which is partially supported by NSF grant NSF-9977508 and partially supported by NJIT SBR grant SBR-421350.

## References

- [1] K.R. Baker and J.B. Martin (1974). An experimental comparison of solution algorithms for the single machine tardiness problem. *Naval Research Logistics Quarterly*, **21**, 187-200.
- [2] P. Brucker (2001). *Scheduling Algorithms*, Springer-Verlag, New York.
- [3] C.L. Chen and R.L. Bulfinch (1993). Complexity of single machine multi-criteria scheduling problems. *European Journal of Operational Research*, **70**, 115-125.
- [4] C.L. Chen and R.L. Bulfinch (1994). Scheduling a single machine to minimize two criteria: maximum tardiness and number of tardy jobs. *IIE Transactions*, **26**, 76-84.
- [5] P. Dileepan and T. Sen (1988). Bicriterion static scheduling research for a single machine. *OMEGA*, **16**, 53-59.
- [6] H. Emmons (1975a). One machine sequencing to minimize mean flow time with minimum number tardy. *Naval Research Logistics Quarterly*, **22**, 585-592.
- [7] H. Emmons (1975b). A note on a scheduling problem with dual criteria. *Naval Research Logistics Quarterly*, **22**, 615-616.
- [8] H. Heck and S. Roberts (1972). A note on the extension of a result on scheduling with a secondary criteria. *Naval Research Logistics Quarterly*, **19**, 403-405.

- [9] Y. Huo, J.Y-T. Leung, and H. Zhao (2004). Complexity of two-dual criteria scheduling problems. Submitted to *Operations Research Letters*.
- [10] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, **5**, 287-326.
- [11] E.L. Lawler (1973). Optimal sequencing of a single machine subject to precedence constraints, *Management Science*, **19**, 544-546.
- [12] C.-Y. Lee and G.L. Vairaktarakis (1993). Complexity of single machine hierarchical scheduling: A survey. In Panos M. Pardalos (ed.), *Complexity in Numerical Optimization*, World Scientific Publishing Co., New Jersey, U.S.A., 269-298.
- [13] J.M. Moore (1968). An  $n$  job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, **15**, 102-109.
- [14] P.S. Ow and T.E. Morton (1992). The single machine early/tardy problem. *Management Science*, **35(2)**, 177-191.
- [15] S.S. Panwalkar, R.K. Dudek, and M.I. Smith (1973). Sequencing research and the industrial scheduling problems. In S.E. Elmaghraby (ed.), *Symposium on the Theory of Scheduling and Its Application*, Springer-Verlag, New York, 29-38.
- [16] M.L. Pinedo (2002). *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, New Jersey, U.S.A.
- [17] A.H.G. Rinnooy Kan, B.J. Lageweg and J.K. Lenstra (1975). Minimizing total costs in one machine scheduling. *Operations Research*, **23**, 908-927.
- [18] J.G. Shanthikumar (1983). Scheduling  $n$  jobs on one machine to minimize the maximum tardiness with minimum number tardy. *Computers Operations Research*, **10**, 255-266.
- [19] W.E. Smith (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly*, **3**, 59-66.
- [20] G.L. Vairaktarakis and C.-Y. Lee (1995). The single-machine scheduling problem to minimize total tardiness subject to minimum number of tardy jobs. *IIE Transactions*, **27**, 250-256.
- [21] M. van den Akker and H. Hoogeveen (2004). Minimizing the number of tardy jobs. In J.Y-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman and Hall/CRC, Boca Raton, FL, USA.
- [22] R.E.D. Woolsey (1992). Survival scheduling with Hodgson's rule or see how those traveling salesmen love one another. *INTERFACES*, **22**, 81-84.

Table 2: Empirical results for instances with processing time range [1, 10] and weight 1.

		Opt	Hybrid	LS-P	LS-D	LPT-B	LPT-F	LDL-F	Comp.
small instances	# of opt	375	364	349	296	349	292	269	368
	#/375	1.000	0.971	0.931	0.789	0.931	0.779	0.717	0.981
	avg ratio	-	1.003	1.004	1.024	1.005	1.042	1.059	1.001
	worst ratio	-	1.200	1.200	1.375	1.200	4.000	4.000	1.167
large instances	# of opt	374	327	314	142	314	203	165	346
	#/374	1.000	0.874	0.840	0.380	0.840	0.543	0.441	0.925
	avg ratio	-	1.006	1.007	1.061	1.007	1.033	1.072	1.004
	worst ratio	-	1.154	1.100	1.306	1.167	1.667	1.667	1.077
all instances	# of opt	749	691	663	438	663	495	434	714
	#/749	1.000	0.923	0.885	0.585	0.885	0.661	0.579	0.953
	avg ratio	-	1.004	1.006	1.043	1.006	1.038	1.066	1.003
	worst ratio	-	1.200	1.200	1.375	1.200	4.000	4.000	1.167

Table 3: Empirical results for instances with processing time range [1, 50] and weight 1.

		Opt	Hybrid	LS-P	LS-D	LPT-B	LPT-F	LDL-F	Comp.
small instances	# of opt	375	360	351	300	343	322	315	367
	#/375	1.000	0.960	0.936	0.800	0.915	0.859	0.840	0.979
	avg ratio	-	1.004	1.006	1.022	1.008	1.025	1.029	1.002
	worst ratio	-	1.333	1.333	1.333	1.333	1.750	1.750	1.250
large instances	# of opt	359	316	288	132	289	222	210	328
	#/359	1.000	0.880	0.802	0.368	0.805	0.618	0.585	0.914
	avg ratio	-	1.021	1.023	1.072	1.024	1.051	1.058	1.019
	worst ratio	-	1.111	1.125	1.327	1.143	1.407	1.519	1.083
all instances	# of opt	734	676	639	432	632	544	525	695
	#/734	1.000	0.921	0.871	0.589	0.861	0.741	0.715	0.947
	avg ratio	-	1.012	1.015	1.047	1.016	1.038	1.043	1.011
	worst ratio	-	1.333	1.333	1.333	1.333	1.750	1.750	1.250

Table 4: Empirical results for instances with processing time range [1, 10] and weight range [1, 10].

		Opt	Hybrid	LS-P	LS-D	LPT-B	LPT-F	LDL-F	Comp.
small instances	# of opt	375	352	292	289	281	188	200	362
	#/375	1.000	0.939	0.779	0.771	0.749	0.501	0.533	0.965
	avg ratio	-	1.004	1.014	1.016	1.023	1.098	1.095	1.002
	worst ratio	-	1.25	1.185	1.250	1.500	2.000	2.000	1.080
large instances	# of opt	367	253	163	121	164	83	84	271
	#/367	1.000	0.689	0.444	0.330	0.447	0.226	0.229	0.738
	avg ratio	-	1.010	1.027	1.033	1.034	1.160	1.143	1.007
	worst ratio	-	1.286	1.257	1.182	1.333	2.000	2.000	1.083
all instances	# of opt	742	605	455	410	445	271	284	633
	#/742	1.000	0.815	0.613	0.553	0.600	0.365	0.383	0.853
	avg ratio	-	1.007	1.021	1.025	1.028	1.129	1.119	1.004
	worst ratio	-	1.286	1.257	1.250	1.500	2.000	2.000	1.083

Table 5: Empirical results for instances with processing time range [1, 50] and weight range [1, 10].

		Opt	Hybrid	LS-P	LS-D	LPT-B	LPT-F	LDL-F	Comp.
small instances	# of opt	375	353	284	289	266	182	205	362
	#/375	1.000	0.941	0.757	0.771	0.709	0.485	0.547	0.965
	avg ratio	-	1.006	1.017	1.017	1.029	1.108	1.074	1.002
	worst ratio	-	1.273	1.286	1.500	1.333	1.750	1.667	1.143
large instances	# of opt	362	265	151	114	159	83	82	285
	#/362	1.000	0.732	0.417	0.315	0.439	0.229	0.227	0.787
	avg ratio	-	1.009	1.031	1.040	1.034	1.187	1.126	1.006
	worst ratio	-	1.220	1.260	1.333	1.320	1.966	1.724	1.120
all instances	# of opt	737	618	435	403	425	265	287	647
	#/737	1.000	0.839	0.590	0.547	0.577	0.360	0.389	0.878
	avg ratio	-	1.007	1.024	1.028	1.031	1.148	1.100	1.004
	worst ratio	-	1.273	1.286	1.500	1.333	1.966	1.724	1.143