

# Minimizing Sum of Completion Times and Makespan in Master-Slave Systems

Joseph Y-T. Leung, *Senior Member, IEEE*, and Hairong Zhao, *Member, IEEE*

## Abstract

We consider scheduling problems in the master-slave model. In this model each job has to be processed sequentially in three stages. In the first stage, a preprocessing task runs on a master machine; in the second stage, a slave task runs on a dedicated slave machine; and in the last stage, a postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. It has been shown that the problem of minimizing the makespan or the sum of completion times is NP-hard in the strong sense even if preemption is allowed. In this paper we design efficient approximation algorithms to minimize the sum of completion times in various settings. These are the first general results for the minsum problem in the master-slave model. We also show that these algorithms generate schedules with small makespan as well.

## Index Terms

Sequence and scheduling, approximation algorithms, makespan, linear programming

## 1 INTRODUCTION

We consider scheduling problems in the master-slave model which was recently introduced by Sahni [21]. In this model, each job has to be processed sequentially in three stages. In the first stage, the preprocessing task runs on a master machine; in the second stage, the slave task runs on a dedicated slave machine; and in the last stage, the postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. We use  $a_i$ ,  $b_i$  and  $c_i$  to denote the preprocessing, slave and postprocessing tasks and task times of job  $i$ , respectively. We assume that  $a_i > 0$ ,  $b_i > 0$  and  $c_i > 0$ .

A job may have a release time  $r_i \geq 0$ , i.e.,  $a_i$  cannot start until time  $r_i$ . In this paper, we assume that every job has a release time 0, unless stated otherwise. There are two cases when arbitrary release time is present. The first case deals with offline problems, i.e., the release times and processing times of all jobs are known in advance. The second case deals with online problems, i.e., we have no knowledge of a job  $i$  until it arrives at time  $r_i$ , and when it arrives, we know everything about job  $i$ . We use the quadruple  $(r_i, a_i, b_i, c_i)$  to denote job  $i$ . For simplicity, if  $r_i = 0$ , we use the triplet  $(a_i, b_i, c_i)$  to represent job  $i$ .

Each machine is either a master machine or a slave machine. The master machines are used to run preprocessing and/or postprocessing tasks, and the slave machines are used to run slave tasks, one slave machine for each slave task. In a single-master system, there is a single master to execute all preprocessing tasks ( $a$  tasks) and postprocessing tasks ( $c$  tasks). In a multi-master system, there are more than one master, each of which is capable of processing both  $a$  tasks and  $c$  tasks. Finally, in some systems, there are distinct preprocessing masters (preprocessors) and postprocessing masters (postprocessors), which are used to process  $a$  tasks and  $c$  tasks, respectively.

The master-slave model is closely related to the flow shop model. The system which has a single preprocessing master and a single postprocessing master can be seen as a two-machine flow shop with transfer lags ([25]). When there are more than one preprocessing and postprocessing masters, the model is a two-stage hybrid flow shop with transfer lags. Hybrid flow shop is often found in electronic

This work was partially supported by NSF Grant DMI-0300156.

J. Y-T. Leung is with the Department of Computer Science, New Jersey Institute of Technology (NJIT), Newark, NJ 07102. Email: leung@cis.njit.edu.

H. Zhao is with the Department of Mathematics, Computer Science, & Statistics, Purdue University Calumet, Hammond, Indiana 46323. Email: hairong@calumet.purdue.edu.

manufacturing environment such as IC packaging and make-to-stock wafer manufacturing. In recent years, hybrid flow shop has received significant attention, see [2], [15], [16] and [26].

The master-slave model finds many applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc. In the following we list some of them; for more applications, see [21], [28], [29] and [30].

Several applications of the master-slave model are found in parallel computer scheduling. A common parallel programming paradigm involves the use of a main computational thread whose function is to prepare data then fork and initiate new child threads that do the computations on different processors. After the computation of a child thread, the main thread collects the computation results and performs some processing on the results. Here, each child thread can be seen as a job with three tasks: the thread initiation and data preparation is the preprocessing task, the computation is the slave task and the postprocessing of the results from the computation is the postprocessing task.

Master-slave paradigm also has applications in certain semiconductor testing operations. In the case of burn-in operations, chips are subject to thermal stress for an extended period of time. The whole process of each chip consists of three phases. First, an initial burn-in operation is accomplished by maintaining the oven at a constant temperature while powering up the chip. The burn-in times for each chip are specified by the customer and it is thus fixed a priori. Then, in the second phase, the chip cools off for a specified amount of time that depends on the length and intensity of the initial burn-in period. In the last phase, the chip is subject to a final burn-in operation. In this application the burn-in oven corresponds to the master machine, the two burn-in tasks correspond to preprocessing and postprocessing and the cooling period corresponds to the slave task. Since the burn-in operations are near the end of the production process, scheduling is critical in determining on-time delivery and output performance for the entire company.

Industrial applications of the master-slave paradigm include the case of consolidators that receive orders to manufacture quantities of various items. The actual manufacturing is done by a collection of slave agencies. In this example, the consolidator is the master machine and the slave agencies are the slave machines. The consolidator needs to assemble the raw material needed for each task, load the trucks that will deliver this material to the slave machines, and perform an inspection before the consignment leaves. All of these work belong to preprocessing task. The slave machines need to wait for the arrival of the raw material, inspect the received goods, perform the manufacture, load the goods onto the trucks for delivery, perform an inspection as the trucks are leaving. These activities together with the delay involved in getting the trucks to their destination (i.e., the consolidator) represent the slave work. When the finished goods arrive at the consolidator, they are inspected and inventoried. This represents the postprocessing.

It is easy to see that all of the above examples generalize to multi-master systems or distinct preprocessing and postprocessing master systems.

### 1.1 Preliminaries

Given a set of jobs in the master-slave system and a schedule  $S$  of the jobs, the completion (or finish) time of job  $i$  in  $S$  is the time when the postprocessing task  $c_i$  finishes. We denote the completion time of  $i$  in  $S$  by  $C_i(S)$ . If  $S$  is clear from the context, we use  $C_i$  instead of  $C_i(S)$ . The *makespan* of  $S$  is the earliest time when all the tasks have been completed. We denote the makespan of  $S$  by  $C_{\max}(S)$ , or  $C_{\max}$  if  $S$  is clear from the context. The *sum of the completion times* of  $S$  is denoted by  $C(S)$ , i.e.,  $C(S) = \sum_{j=1}^n C_j$ . Makespan and sum of completion times are two common objectives to minimize. Throughout this paper we use  $C_{\max}^*$  and  $C^*$  to denote the minimum makespan and the minimum sum of completion times, respectively. Without loss of generality, we always assume that the  $b$  tasks are scheduled immediately as soon as the corresponding  $a$  tasks complete.

An  $\alpha$ -*approximation* algorithm for makespan (or sum of completion times) is an algorithm that for any set of jobs generates a schedule  $S$  whose makespan (or sum of completion times) is at most  $\alpha$  times the optimal makespan (or sum of completion times). It is an  $(\alpha, \beta)$ -*approximation* algorithm if it is an  $\alpha$ -approximation algorithm for makespan and at the same time a  $\beta$ -approximation algorithm for sum of

completion times. For a schedule  $S$ , if  $C_{\max}(S) \leq \alpha C_{\max}^*$  and  $C(S) \leq \beta C^*$ , then we say that  $S$  is an  $(\alpha, \beta)$ -schedule.

The Shortest-Processing-Time (SPT) rule, which always runs the job with the least processing time, and the Shortest-Remaining-Processing-Time (SRPT) rule [23], [24], which always runs the job with the least remaining processing time, are two well-known algorithms for minimizing sum of completion times. Usually, the SPT rule is used to generate non-preemptive schedules, while the SRPT rule is used to generate preemptive schedules. Suppose each job consists of a single task. If all jobs are available at time 0, then the SPT rule is optimal for sum of completion times in the single-machine or multi-machine environment. If the release times are arbitrary, then the SRPT rule is optimal for a single machine and it is a 2-approximation (see [18]) in the multi-machine environment.

In this paper, we *adapt* the above two rules for our setting. We apply both rules to generate *preemptive* schedules. A scheduling decision is made when a task completes so that a master machine becomes free, or when a new  $a$  task or  $c$  task becomes available. At any such time instant, the SPT rule schedules, from the set of available tasks (including those that have been preempted but have not yet completed), the one with the smallest processing time. Depending on how we choose from the set of available jobs, we have the  $SPT_a$  rule and the  $SPT_c$  rule. Specifically, in the  $SPT_a$  rule, preemption occurs only among the  $a$  tasks and the preemption is based on the length of  $a_i$ . In the  $SPT_c$  rule, preemption occurs only among the  $c$  tasks and the preemption is based on the length of  $c_i$ . On the other hand, the SRPT rule schedules, from the set of available tasks, the one with the smallest remaining processing time. Similarly, we define the  $SRPT_a$  rule and the  $SRPT_c$  rule. Both the SPT rule and the SRPT rule may generate schedules with *migration* when there are multiple machines, i.e., after interrupted on one machine, a task resumes on a different machine.

In this paper, we frequently sort jobs in certain ascending order of some parameters  $x_i$  of job  $i$  ( $x_i$  can be  $a_i$ ,  $c_i$  or  $a_i + c_i$ ). For convenience, we say that  $x_i < x_j$  as long as  $x_i$  comes before  $x_j$  in the sorted order, even though it may be the case that  $x_i = x_j$ .

Given a set of jobs in the master-slave system, we may have some constraints or requirements about how these jobs should be scheduled. For example, one may require that all preprocessing tasks run before any postprocessing tasks can start. In other applications, preemption may not be allowed; that is, once a task starts, it continues to run until it finishes. In the following, we discuss these constraints in detail.

**Canonical schedule versus non-canonical schedule.** A *canonical* schedule on the single master system is one such that all the preprocessing tasks complete before any postprocessing tasks can start (Note that this definition of canonical schedule is slightly different from the one given in [21]). In the multi-master system, a canonical schedule is one that is canonical on each master.

Both canonical and non-canonical schedules have some applications. In the consolidators example, canonical schedules make sense while non-canonical schedules do not. On the other hand, in the parallel computer scheduling example, non-canonical schedules make sense while canonical schedules do not. For both makespan and sum of completion times problems, it has been shown that the problems are NP-hard for both canonical and non-canonical schedules. It is easy to see that for a single master, one can always arrange a schedule to be canonical without increasing the makespan. Thus, for the problem of minimizing the makespan, we only need to focus on canonical schedules. However, for the problem of minimizing the sum of completion times, the ratio of the sum of completion times of the best canonical schedule versus that of the best non-canonical schedule can be arbitrarily large. Consider the example:  $(n - 1)$  identical jobs  $(1, \epsilon, 1)$  and one job  $(n^2, \epsilon, 1)$ , where  $\epsilon$  is an arbitrary small positive number. The optimal canonical schedule has the sum of completion times  $O(n^3)$ , while the optimal non-canonical schedule has the sum of completion times  $O(n^2)$ .

**Preemptive schedule versus non-preemptive schedule.** Both non-preemptive and preemptive schedules have some applications. In the consolidators example, non-preemptive schedules are more realistic than preemptive schedules. On the other hand, in the parallel computer scheduling example, preemptive schedules are as realistic as non-preemptive schedules. For both makespan and sum of completion times

problems, it has been shown that the problems are NP-hard even if preemption is allowed. For some instances, preemption does not help to reduce the makespan or the sum of completion times. However, in most cases, preemption can reduce the sum of completion times and makespan.

## 1.2 Previous works

So far the main research efforts to the master-slave model are for minimizing the makespan and for special cases of the sum of completion times problem. The general makespan problem has been shown to be NP-hard by Kern and Nawijn [14]. Leung and Zhao [17] strengthened the result and showed that the problem remains NP-hard in the strong sense, even if all the preprocessing and postprocessing tasks have unit time. In other words, the problem remains NP-hard in the strong sense even if preemption is allowed.

Sahni and Vairaktarakis [28] proposed several heuristics for the makespan problem in the single-master and multi-master systems. For the general problem under the single-master systems, they developed a heuristic with a worst-case bound of  $\frac{3}{2}$ . For the multi-master systems, they gave heuristics with worst-case bounds of 2. When there are  $m_1$  preprocessors and  $m_2$  postprocessors, Vairaktarakis [30] gave heuristics for makespan that have approximation ratio of  $2 - \frac{1}{m}$ , where  $m = \max\{m_1, m_2\}$ .

The sum of completion times minimization problem is considered only in [17] by Leung and Zhao. They showed that the problem of finding the optimal canonical or non-canonical schedule is NP-hard even if all preprocessing and postprocessing tasks have unit time. They designed efficient approximation algorithm to minimize the sum of completion times of canonical schedules on a single-master system when  $a_i = a$  and  $c_i = c$  for all jobs  $i$ . The performance ratio is  $\frac{7}{6}$  when  $a = c$ ,  $\frac{5}{4}$  when  $a > c$ , and  $\frac{4}{3}$  when  $a < c$ .

Flow shop is one of the most classical models that have been studied for a long time. Let  $m$  be the number of stages. For the makespan problem, Johnson [13] developed an  $O(n \log n)$  time optimal algorithm when  $m = 2$ . The problem becomes NP-hard when  $m \geq 3$ . In this case, Hall [12] presented a  $(1 + \epsilon)$ -approximation algorithm for any fixed positive  $\epsilon$ . For the sum of completion times problem, it is NP-hard in the strong sense even if  $m = 2$  and preemption is allowed [5]. Gonzalez and Sahni [9] developed an approximation algorithm for the  $m$ -stage flow shop problem with approximation ratio of  $m$ . Let  $p_i$  be the total processing time of all operations of job  $i$ . The algorithm schedules the jobs in ascending order of  $p_i$  at each stage. By a careful analysis, Hoogeveen et. al [11] showed that this algorithm has approximation ratio  $2\beta/(\alpha + \beta)$ , where  $\alpha$  denotes the minimal processing time of all tasks and  $\beta$  denotes the maximal processing time of all tasks. If the jobs have different weights, Schulz [22] obtained an approximation algorithm with performance guarantee of  $2m$  (or  $2m + 1$  in case of arbitrary release time) for the sum of weighted completion times based on linear programming.

When there are more than one machine in either or both stages, we have the flexible or hybrid flow shop. Both makespan and sum of completion times minimization problems are NP-hard, even if preemption is allowed; see [7] and [5]. Lee and Vairaktarakis [16] developed heuristics for makespan with approximation ratio of  $2 - 1/\max\{m_1, m_2\}$ , where  $m_1$  and  $m_2$  are the number of machines in stages 1 and 2, respectively. Based on linear programming, Schulz [22] obtained an approximation algorithm with performance guarantee of  $3m$  (or  $3m + 1$  in case of arbitrary release time) for the sum of weighted completion times, where  $m$  is the number of stages. Thus, if  $m = 2$ , it is a 6-approximation in the case of identical release times and a 7-approximation in the case of arbitrary release times.

For the two-stage flow shop with transfer lags model, some research has been done, most of which is about makespan minimization. Dell'Amico [1] proved that the makespan problem is NP-hard, even if preemption is allowed and each stage has only one machine. Later, Yu, Hoogeveen and Lenstra [32] showed that the problem is NP-hard even if all tasks have unit length. This is in contrast to the fact that the problem is solvable in polynomial time when there is no transfer lags. As we mentioned before, this model is the same as the master-slave model when the preprocessing and postprocessing masters are distinct. Thus, the heuristics given in [30] for the master-slave model also work here. Little is known about the sum of completion times minimization problem.

TABLE I  
NEW RESULTS FOR SINGLE-MASTER SYSTEM

|              | preemptive                          | non-preemptive                  |
|--------------|-------------------------------------|---------------------------------|
| $r_i = 0$    | $(\frac{3}{2}, 2)$ , canonical      | $(\frac{5}{2}, \frac{2e}{e-1})$ |
| $r_i = 0$    | $(2, 2)$ , non-canonical            | $(3, 4)$                        |
| $r_i \geq 0$ | $(2, 2)$ , online and non-canonical | $(3, 4)$                        |

TABLE II  
NEW RESULTS FOR MULTI-MASTER SYSTEM, ALL SCHEDULES ARE NON-CANONICAL

|              | preemptive                           | non-preemptive |
|--------------|--------------------------------------|----------------|
| $r_i = 0$    | $(3, 2)$ , no migration              | $(4, 4)$       |
| $r_i \geq 0$ | $(3, 2)$ , offline with no migration | $(5, 4)$       |
| $r_i \geq 0$ | $(3, 2)$ , online with migration     | -              |

### 1.3 New results

In this paper, our main objective is to minimize the sum of completion times and makespan under various scenarios. This includes single-master systems, multi-master systems, and distinct preprocessing and postprocessing master systems. Since the problem is strongly NP-hard, we can only hope to have approximation algorithms. For each type of system, we try to approximate the best canonical preemptive or non-preemptive schedule, the best non-canonical preemptive or non-preemptive schedule. We consider both the case when all jobs have the same release times and the case when the jobs have different release times.

Preemptive relaxation and linear programming relaxation are two important techniques for getting constant-factor approximations for sum of completion times of non-preemptive schedules; see [18], [10], [4], [6], [27] and [22]. Most of these algorithms work by first constructing a relaxed solution, either a preemptive schedule or a linear programming relaxation. These relaxations are then used to obtain an ordering of the jobs, and the jobs are list scheduled (i.e., no unforced idle time) in this order. We will say more about these methods in Section 5.

The advantage of preemptive relaxation is that usually there are very efficient algorithms to generate optimal or near optimal schedules. In most cases, these algorithms (both preemptive and non-preemptive) can be implemented to run in an online fashion, see [18] and [10]. The linear programming relaxation, however, is more time consuming and can only be implemented to run in an offline fashion. On the other hand, it provides better approximation bounds in some cases. Furthermore, the method usually works even if we want to optimize the sum of weighted completion times; see [10], [4], [6], [27] and [22].

In this paper, we use both approaches. In all cases, we first develop efficient algorithms to generate preemptive offline or online schedules with good approximation. We show that these schedules have small makespan as well. Then, by applying the ideas in [18] and [3] to our models, we convert these preemptive schedules into non-preemptive schedules with certain degradation in the quality of approximation. In the case when there are distinct preprocessing masters and postprocessing masters, we also use linear programming relaxation. We show that non-preemptive schedules obtained in this way sometimes have better performance than those obtained by the preemptive relaxation approach. Our results are summarized in Tables I, II, III and IV.

The organization of this paper is as follows. In Sections 2, 3 and 4, we consider preemptive schedules. Section 2 is devoted to the single-master case, Section 3 is devoted to the multi-master case, and Section 4 is devoted to the case of distinct preprocessor and postprocessor. In Section 5, we consider the conversions from preemptive schedules into non-preemptive schedules. Section 6 is devoted to the linear programming relaxation approach. In this section we focus on distinct preprocessor and postprocessor only. Finally, we draw some conclusions in the last section.

TABLE III

NEW RESULTS FOR DISTINCT PREPROCESSOR AND POSTPROCESSOR SYSTEM,  $m_1 = m_2 = 1$ 

|              | preemptive      | non-preemptive            |               |
|--------------|-----------------|---------------------------|---------------|
|              |                 | preemptive relaxation     | LP relaxation |
| $r_i = 0$    | (3, 2)          | $(4, \frac{2e}{e-1})$     | (3, 5)        |
| $r_i \geq 0$ | (3, 2), offline | $(4, 4 + \frac{2e}{e-1})$ | (4, 6)        |
| $r_i \geq 0$ | (3, 2), online  | $(4, 4 + \frac{2e}{e-1})$ | -             |

TABLE IV

NEW RESULTS FOR DISTINCT PREPROCESSOR AND POSTPROCESSOR,  $m_1 \geq 1$  AND  $m_2 \geq 1$ 

|              | preemptive                     | non-preemptive        |               |
|--------------|--------------------------------|-----------------------|---------------|
|              |                                | preemptive relaxation | LP relaxation |
| $r_i = 0$    | (4, 2), with migration         | $(4, \frac{14}{3})$   | (4, 6)        |
| $r_i \geq 0$ | (4, 3), offline with migration | (5, 13)               | (5, 7)        |
| $r_i \geq 0$ | (4, 3), online with migration  | (5, 13)               | -             |

## 2 SINGLE-MASTER SYSTEMS

In this section we assume that there is a single master. We first study canonical schedules, and then non-canonical schedules. Finally, we study the case when jobs have different release times. For convenience, throughout this paper, we let  $A = \sum_{j=1}^n a_j$ ,  $B = \sum_{j=1}^n b_j$  and  $C = \sum_{j=1}^n c_j$ .

### 2.1 Canonical preemptive schedules

We begin with two lower bounds of canonical schedules. Suppose we are given  $n$  jobs 1, 2, ...,  $n$ . For canonical schedules, preemption allowed or not, we have the following lower bound

$$C^* \geq nA + \sum_{j=1}^n \sum_{c_i \leq c_j} c_i, \quad (1)$$

which assumes that there is no idle time in the schedule and that the  $c$  tasks are scheduled in ascending order of their lengths. Another lower bound is

$$C^* \geq \sum_{j=1}^n \sum_{a_i \leq a_j} a_i + B + C, \quad (2)$$

which assumes that jobs are scheduled in ascending order of the  $a_i$ , and that the  $b$  tasks and the  $c$  tasks are scheduled immediately after they are available. Finally, we have the following trivial lower bound for any schedules which is simply the summation of all the processing times.

$$C^* \geq A + B + C \quad (3)$$

This follows from the observation that  $C_j \geq a_j + b_j + c_j$ . Summing  $j$  from 1 to  $n$  gives the result in (3).

In canonical schedules, all the  $a$  tasks are scheduled first. Since all the  $a$  tasks are available at time 0 and the  $c$  tasks cannot start until all the  $a$  tasks finish, there is no need to preempt the  $a$  tasks. Hence, only a  $c$  task can be preempted by another  $c$  task in this case.

**Heuristic 1:** Schedule the  $a$  tasks in an arbitrary order. After all the  $a$  tasks finish, schedule the available  $c$  tasks by the  $SPT_c$  rule.

**Performance analysis of Heuristic 1.** Let  $C_{a_j}$  denote the time  $a_j$  completes. Then at time  $t_j = \max(A, (C_{a_j} + b_j))$ , all the  $a$  tasks finish and the task  $c_j$  is available to be scheduled. Since  $C_{a_j} \leq A$ ,

we have  $t_j \leq A + b_j$ . According to the heuristic, if there is another available task  $c_i$  that hasn't finished at time  $t_j$  and  $c_i < c_j$ , then  $c_j$  has to wait until  $c_i$  finishes. Also, during the execution of  $c_j$ , if there is another task  $c_i < c_j$  that becomes available, then  $c_i$  preempts  $c_j$ . In both cases, we say that  $c_j$  is *delayed* by  $c_i$ . Let  $C_j$  be the completion time of  $c_j$  in the schedule generated by Heuristic 1. Then, we have

$$C_j = (t_j + c_j) + \sum_{c_i \text{ delays } c_j} c_i \leq (A + b_j + c_j) + \sum_{c_i < c_j} c_i .$$

The sum of completion times is

$$\sum_{j=1}^n C_j \leq \sum_{j=1}^n \left( A + b_j + c_j + \sum_{c_i < c_j} c_i \right) = \left( nA + \sum_{j=1}^n \sum_{c_i < c_j} c_i \right) + (B + C) \leq 2C^* ,$$

where the last inequality comes from the lower bounds (1) and (3).

The above approximation ratio is tight. Consider this example: for  $1 \leq i \leq n-1$ ,  $a_i = c_i = \epsilon$ ,  $b_i = a + (n-1)\epsilon$ ,  $a_n = c_n = a$  and  $b_n = \epsilon$ . The optimal canonical schedule schedules  $a_1, a_2, \dots, a_{n-1}$  first and then followed by  $a_n$ . The sum of completion times is about  $(n+1)a$ . However, if we schedule  $a_n$  first followed by  $a_1, \dots, a_{n-1}$ , then the sum of completion times is about  $2na$ . (For this example, the optimal preemptive schedule has the same sum of completion times as the optimal non-preemptive schedule.)

It has been shown in [28] that any canonical schedule without preemption is a 2-approximation for makespan. Since preemption among the  $c$  tasks can not increase the makespan, the schedule generated by Heuristic 1 has makespan at most twice the optimal. Therefore, we have the following theorem.

*Theorem 1:* Heuristic 1 generates a  $(2, 2)$  canonical preemptive schedule in  $O(n \log n)$  time.

Let  $S_1 = \{i : a_i \leq c_i\}$  and  $S_2 = \{i : a_i > c_i\}$ . Suppose the jobs in  $S_1$  are arranged in ascending order of the  $b$ 's and the jobs in  $S_2$  are arranged in descending order of the  $b$ 's. In [28] it was shown that the canonical schedule in which the  $a$  tasks of  $S_1$  are scheduled before the  $a$  tasks of  $S_2$  has makespan at most  $\frac{3}{2}$  times the optimal makespan. Thus, if we schedule the  $a$  tasks in this order in Heuristic 1, we still get a  $\frac{3}{2}$ -approximation for makespan, since preemption on the available  $c$  tasks will not increase the makespan.

*Corollary 2:* There is an  $O(n \log n)$  time algorithm that generates a  $(\frac{3}{2}, 2)$  canonical preemptive schedule.

Note that when preemption occurs we use the SPT rule instead of the SRPT rule. This is for the purpose of analysis only. In practice, one can use the SRPT rule to get a better approximation for sum of completion times.

## 2.2 Non-canonical preemptive schedules

In this model, the  $a$  tasks and the  $c$  tasks can be scheduled alternatively. A lower bound on the  $C^*$  can be obtained by assuming that all the  $b$  tasks have length 0:

$$C^* \geq \sum_{j=1}^n \sum_{a_i + c_i \leq a_j + c_j} (a_i + c_i) \quad (4)$$

**Heuristic 2:** For any two jobs, if  $(a_j + c_j) < (a_i + c_i)$ , then we say that both  $a_j$  and  $c_j$  have higher priority than  $a_i$  and  $c_i$ . If  $(a_j + c_j) = (a_i + c_i)$ , then the relative priority can be resolved arbitrarily. At any time, if the master processor is free for assignment, assign the available task with the highest priority. If a new task becomes available and has higher priority than the currently running task, the new task preempts the currently running task.

**Performance analysis of Heuristic 2.** Since all the  $a$  tasks are ready at time 0, there is no need for an  $a$  task to preempt another task. Thus, preemption occurs only between a higher priority  $c$  task and a lower priority  $c$  task or a lower priority  $a$  task. Let  $C_{a_j}$  denote the completion time of  $a_j$  in the schedule generated by Heuristic 2. If none of the  $a$  tasks is preempted by a  $c$  task, then  $C_{a_j}$  would be  $a_j + \sum_{a_i+c_i < a_j+c_j} a_i$ . Because of preemption,  $a_j$  can be delayed by some higher priority  $c$  tasks. In other words,  $c_i$  can delay  $a_j$  only if  $a_i + c_i < a_j + c_j$ . At time  $t_j = C_{a_j} + b_j$ , the task  $c_j$  becomes available. According to the heuristic,  $c_j$  can only be delayed by a task  $c_i$  such that  $a_i + c_i < a_j + c_j$ . Note that if  $c_i$  delays  $a_j$ , it will not delay  $c_j$  again. Thus, we can bound the completion time  $C_j$ :

$$\begin{aligned} C_j &\leq C_{a_j} + b_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \\ &= \left( \sum_{a_i+c_i < a_j+c_j} a_i + \sum_{c_i \text{ delays } a_j} c_i \right) + a_j + b_j + c_j + \left( \sum_{c_i \text{ delays } c_j} c_i \right) \\ &\leq \left( \sum_{a_i+c_i < a_j+c_j} a_i \right) + \left( \sum_{a_i+c_i < a_j+c_j} c_i \right) + (a_j + b_j + c_j) \\ &= \left( \sum_{a_i+c_i < a_j+c_j} (a_i + c_i) \right) + (a_j + b_j + c_j) . \end{aligned}$$

Therefore, the sum of completion times is

$$\begin{aligned} \sum_{j=1}^n C_j &\leq \sum_{j=1}^n \left( \left( \sum_{a_i+c_i < a_j+c_j} (a_i + c_i) \right) + a_j + b_j + c_j \right) \\ &\leq \left( \sum_{j=1}^n \sum_{a_i+c_i < a_j+c_j} (a_i + c_i) \right) + (A + B + C) \quad \text{by (4) and (3)} \\ &\leq 2C^* . \end{aligned}$$

We can also bound the makespan of the schedule. Pick the last job  $j$  such that  $c_j$  is scheduled immediately when it is ready. Such a job always exists since the job  $i$  with the highest priority must be such a job. Then, the interval  $I_1 = [0, C_{a_j})$  and the interval after  $c_j$  starts till the end, i.e.,  $I_2 = [(C_{a_j} + b_j), C_{\max})$ , contain no idle time. Denote their lengths by  $|I_1|$  and  $|I_2|$ , respectively. Then  $|I_1| + |I_2| \leq C_{\max}^*$ . Thus, the makespan is

$$C_{\max} = |I_1| + b_j + |I_2| \leq C_{\max}^* + b_j \leq 2C_{\max}^*$$

The above bound is tight. Consider the instance such that  $a_i = 1$ ,  $b_i = c_i = \epsilon$  for  $1 \leq i \leq n-1$ , and  $a_n = 2$ ,  $b_n = n$  and  $c_n = \epsilon$ . When  $\epsilon$  is arbitrarily small, the minimum makespan is about  $n+2$  while the schedule generated by Heuristic 2 has makespan about  $2n+1$ .

*Theorem 3:* Heuristic 2 generates a (2, 2) preemptive schedule for a single-master system when all jobs have the same release times. Furthermore, it can be implemented in  $O(n \log n)$  time.

### 2.3 Arbitrary release times

When arbitrary release times are present, it is not meaningful to have canonical schedules any more. Thus, we consider only non-canonical schedules. Apparently, the lower bound (4) still holds for this case. Let  $R = \sum_{i=1}^n r_i$ . Then, a trivial lower bound for the minimum sum of completion times of any schedule is

$$C^* \geq \sum_{i=1}^n (r_i + a_i + b_i + c_i) = R + A + B + C \quad (5)$$

Observe that Heuristic 2 makes no assumptions about the release time of a job. So, we can still apply Heuristic 2 when jobs have arbitrary release times. Unlike the case when all jobs have the same release times, in this case, a higher priority  $a$  task may also delay a lower priority  $a$  task or  $c$  task. In the following we will show that Heuristic 2 still gives a  $(2, 2)$  preemptive schedule when arbitrary release times are present.

We first bound the sum of completion times of the schedule generated by Heuristic 2 when the release times are arbitrary. Let  $C_{a_j}$  be defined as before. Then, we have

$$C_{a_j} = r_j + \left( \sum_{a_i \text{ delays } a_j} a_i + \sum_{c_i \text{ delays } a_j} c_i \right) + a_j$$

and

$$\begin{aligned} C_j &= C_{a_j} + b_j + c_j + \left( \sum_{a_i \text{ delays } c_j} a_i + \sum_{c_i \text{ delays } c_j} c_i \right) \\ &= r_j + \left( \sum_{a_i \text{ delays } a_j} a_i + \sum_{c_i \text{ delays } a_j} c_i \right) + a_j + b_j + c_j + \sum_{a_i \text{ delays } c_j} a_i + \sum_{c_i \text{ delays } c_j} c_i \\ &\leq \left( \sum_{a_i + c_i < a_j + c_j} (a_i + c_i) \right) + (r_j + a_j + b_j + c_j) , \end{aligned}$$

where the last inequality comes from the fact that the two sets of tasks delaying  $a_j$  and  $c_j$  are disjoint, and they all have higher priority than  $a_j$  and  $c_j$ . Similarly, we can bound the sum of completion times

$$\begin{aligned} \sum_{j=1}^n C_j &\leq \sum_{j=1}^n \left( \left( \sum_{a_i + c_i < a_j + c_j} (a_i + c_i) \right) + (r_j + a_j + b_j + c_j) \right) \\ &\leq \left( \sum_{j=1}^n \sum_{a_i + c_i < a_j + c_j} (a_i + c_i) \right) + (R + A + B + C) \quad \text{by (4) and (5)} \\ &\leq 2C^* . \end{aligned}$$

To bound the makespan, we pick the last job  $j$  such that  $c_j$  starts immediately after it is ready. Then the intervals  $I_1 = [r_j, C_{a_j})$  and  $I_2 = [(C_{a_j} + b_j), C_{\max})$  must be both busy, and the total length  $|I_1| + |I_2|$  of these two intervals is at most  $\sum_{j=1}^n a_j + \sum_{j=1}^n c_j = A + C \leq C_{\max}^*$ . Thus,

$$C_{\max} = r_j + |I_1| + b_j + |I_2| \leq (r_j + b_j) + C_{\max}^* \leq 2C_{\max}^* .$$

*Theorem 4:* Heuristic 2 generates a  $(2, 2)$  preemptive schedule for a single-master system even when the jobs have arbitrary release times.

Note that Heuristic 2 schedules jobs in an online fashion. Therefore, we have the following corollary.

*Corollary 5:* Heuristic 2 is an online algorithm that generates a  $(2, 2)$  preemptive schedule.

### 3 MULTI-MASTER SYSTEMS

In this section, we assume that there are  $m \geq 2$  masters, each of which is capable of processing both the  $a$  tasks and the  $c$  tasks.

### 3.1 Non-canonical preemptive schedules

Let us assume that the jobs are indexed in *ascending* order of  $a_j + c_j$ . That is,  $a_j + c_j \leq a_{j+1} + c_{j+1}$  for  $1 \leq j \leq n - 1$ .

**Heuristic 3:** (1) Without loss of generality, we may assume that  $n$  is a multiple of  $m$ . Otherwise, we add dummy jobs with  $a_i = b_i = c_i = 0$ . (2) We assign the jobs to the machines such that jobs  $1, 2, \dots, m$  go to machines  $1, 2, \dots, m$ , respectively; jobs  $m + 1, m + 2, \dots, 2m$  go to machines  $m, m - 1, \dots, 1$ , respectively; jobs  $2m + 1, 2m + 2, \dots, 3m$  go to machines  $1, 2, \dots, m$ , respectively; and so on until all jobs are assigned. (3) Apply Heuristic 2 to each master machine to schedule the jobs assigned to it.

**Performance analysis of Heuristic 3.** By the heuristic, we can assume that  $n = mk$  for some integer  $k$ . For convenience, we reindex the jobs assigned to each machine  $p$  in the form of  $(p, q)$  such that  $a_{(p,q)} + c_{(p,q)} \leq a_{(p,q+1)} + c_{(p,q+1)}$ . A lower bound of the sum of completion times comes from the fact that Heuristic 3 is optimal if  $b_{(p,q)} = 0$  for every job  $(p, q)$ .

$$C^* \geq \sum_{p=1}^m \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \quad (6)$$

Fix a machine  $p$ . Let  $C_{(p,q)}$  denote the completion time of job  $(p, q)$  and  $B_p = \sum_{q=1}^k b_{(p,q)}$ . Following the analysis of Heuristic 2 in Section 2.2, we have

$$\sum_{q=1}^k C_{(p,q)} \leq \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B_p .$$

Thus, the sum of completion times is

$$\begin{aligned} \sum_{p=1}^m \sum_{q=1}^k C_{(p,q)} &\leq \sum_{p=1}^m \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) + B_p \right) \\ &\leq \sum_{p=1}^m \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B \quad \text{by (6) and (3)} \\ &\leq 2C^* . \end{aligned}$$

Now we analyze the makespan of the schedule. Let  $\Delta = A + C$ . Then,

$$C_{\max}^* \geq \frac{\sum_{j=1}^n (a_j + c_j)}{m} = \frac{\Delta}{m} .$$

Let  $\Delta_p = \sum_{j \text{ scheduled on } p} (a_j + c_j)$ . By the pigeon hole principle,  $\min_{1 \leq p \leq m} \Delta_p \leq \Delta/m \leq C_{\max}^*$ . For any two machines  $p$  and  $q$ , by the way we assign jobs to the machines

$$\Delta_p - \Delta_q \leq \max_{1 \leq k \leq n} (a_k + c_k) - \min_{1 \leq k \leq n} (a_k + c_k) \leq \max_{1 \leq k \leq n} (a_k + c_k) \leq C_{\max}^* .$$

This means that for any machine  $p$ ,  $\Delta_p \leq \min_{1 \leq q \leq m} \Delta_q + C_{\max}^* \leq 2C_{\max}^*$ . Suppose the job with the maximum completion time among all jobs is assigned to machine  $p$ . Let  $l$  be the last job on machine  $p$  so that  $c_l$  is scheduled immediately after it is ready. Define  $C_{a_l}$  as before. Then machine  $p$  is busy during the intervals  $I_1 = [0, C_{a_l})$  and  $I_2 = [(C_{a_l} + b_l), C_{\max}^*)$ . The total length of the two intervals is  $|I_1| + |I_2| \leq \Delta_p \leq 2C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = |I_1| + b_l + |I_2| < 3C_{\max}^* .$$

**Theorem 6:** Heuristic 3 generates a (3, 2) preemptive schedule in  $O(n \log n)$  time for multi-master systems without migration when all jobs have release time 0.

### 3.2 Arbitrary release times

**3.2.1 Offline schedule without migration:** In this case, we can still apply Heuristic 3. However, note that to assign jobs to machines, Heuristic 3 requires that we have full knowledge of all the jobs at the beginning. Thus, Heuristic 3 is an offline algorithm. We combine the arguments in Sections 2.3 and 3.1.

Fix a machine  $p$ .

$$\sum_{q=1}^k C_{(p,q)} \leq \left( \max_{1 \leq q \leq k} r_{(p,q)} + \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B_p .$$

Thus, the sum of completion times is

$$\begin{aligned} \sum_{p=1}^m \sum_{q=1}^k C_{(p,q)} &\leq \sum_{p=1}^m \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) + B_p + \max_{1 \leq q \leq k} r_{(p,q)} \right) \\ &\leq \sum_{p=1}^m \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B + R \quad \text{by (6) and (5)} \\ &\leq 2C^* . \end{aligned}$$

Now consider the makespan. Suppose the job with the maximum completion time among all jobs is assigned to machine  $p$ . Let  $l$  be the last job on machine  $p$  so that  $c_l$  is scheduled immediately after it is ready. Define  $C_{a_l}$  as before. Then the machine  $p$  is busy during the intervals  $I_1 = [r_l, C_{a_l})$  and  $I_2 = [(C_{a_l} + b_l), C_{\max})$ . The total length of the two intervals is  $|I_1| + |I_2| \leq \Delta_p < 2C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + b_l + |I_2| \leq 3C_{\max}^* .$$

**Theorem 7:** Heuristic 3 generates a  $(3, 2)$  offline preemptive schedule without migration for multi-master systems when jobs have arbitrary release times.

**3.2.2 Online schedule with migration:** In this case, we apply Heuristic 2 to multi-master systems. Note that if a new task becomes available, all machines have been assigned, and one or more currently running tasks have lower priority than the new task, then the task with the lowest priority will be preempted.

**Performance analysis of Heuristic 2 for multi-master system.** We need another lower bound for multi-master systems. Consider the instance  $I$  defined by  $(a_i/m, b_i, c_i/m)$ ,  $1 \leq i \leq n$ , on a single-master system. Then, by (4), its sum of completion times is at least  $\sum_{a_i+c_i \leq a_j+c_j} (a_i + c_i)/m$ . We claim that this is also a lower bound for the instance  $II$  defined by  $(a_i, b_i, c_i)$ ,  $1 \leq i \leq n$ , on  $m$  masters. Let  $S^*$  be an optimal schedule of these  $n$  jobs. Then, starting from time 0, for each time unit, if  $a_i$  or  $c_i$  is scheduled in this unit on some machine, then we schedule  $1/m$  of the task to the single master in the same time unit. It is easy to see that the constraint of release time are preserved and that the interval between the finish time of  $a_i$  and the start time of  $c_i$  is either the same or increased. Thus we get a valid schedule of the instance  $I$  on the single-master system. Therefore, we have

$$C^* \geq \sum_{j=1}^n \sum_{a_i+c_i \leq a_j+c_j} \frac{(a_i + c_i)}{m} , \quad (7)$$

which is a lower bound of the original instance defined by  $(r_i, a_i, b_i, c_i)$ .

Let  $C_{a_j}$  denote the completion time of  $a_j$  in the schedule generated by Heuristic 2. After  $a_j$  is released at  $r_j$ , it may be delayed by other tasks with higher priority before it completes. That is,  $a_i$  or  $c_i$  can delay  $a_j$  only if  $a_i + c_i < a_j + c_j$ . It is easy to see that there is no idle time on any machine during the interval  $I_1 = [r_j, (C_{a_j} - a_j))$ ; otherwise,  $a_j$  would have completed earlier. Furthermore, only tasks with higher priority than  $j$  can be scheduled during this interval. At time  $t_j = C_{a_j} + b_j$ , the task  $c_j$  becomes ready. Similarly, there is no idle time on any machine during the interval  $I_2 = [C_{a_j} + b_j, (C_j - c_j))$ , and

only tasks with higher priority than  $j$  can be scheduled during this interval. Note that the task sets in the intervals  $I_1$  and  $I_2$  are disjoint. Let  $|I_1|$  and  $|I_2|$  denote the lengths of  $I_1$  and  $I_2$ , respectively. Then, we must have  $|I_1| + |I_2| \leq \sum_{a_i+c_i < a_j+c_j} (a_i + c_i)/m$ . Thus, we can bound the completion time  $C_j$  as follows:

$$\begin{aligned} C_j &\leq r_j + |I_1| + a_j + b_j + |I_2| + c_j \\ &\leq \left( \sum_{a_i+c_i < a_j+c_j} \frac{(a_i + c_i)}{m} \right) + (r_j + a_j + b_j + c_j) . \end{aligned}$$

The sum of completion times is

$$\begin{aligned} \sum_{j=1}^n C_j &\leq \sum_{j=1}^n \left( \left( \sum_{a_i+c_i < a_j+c_j} \frac{(a_i + c_i)}{m} \right) + (r_j + a_j + b_j + c_j) \right) \\ &\leq \left( \sum_{j=1}^n \sum_{a_i+c_i < a_j+c_j} \frac{(a_i + c_i)}{m} \right) + (R + A + B + C) \quad \text{by (7) and (5)} \\ &\leq 2C^* . \end{aligned}$$

For the makespan of the schedule, let  $k$  be the job with the maximum completion time among all jobs. Let  $l$  be the last job among all jobs so that  $c_l$  is scheduled immediately after it is ready. Define  $C_{a_l}$  as before. Then, all machines must be busy during the intervals  $I_1 = [r_l, C_{a_l} - a_l]$  and  $I_2 = [(C_{a_l} + b_l), C_{\max} - c_k]$ . The total length of the two intervals is  $|I_1| + |I_2| \leq C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + a_l + b_l + |I_2| + c_k = (r_l + a_l + b_l) + (|I_1| + |I_2|) + c_k \leq 3C_{\max}^* .$$

*Theorem 8:* Heuristic 2 generates a  $(3, 2)$  online preemptive schedule with migration on multi-master systems when jobs have arbitrary release times.

#### 4 DISTINCT PREPROCESSING AND POSTPROCESSING MASTERS

In this section, we study the model with distinct preprocessing masters and postprocessing masters. Different from the previous cases, here an  $a$  task can only be preempted by another  $a$  task and a  $c$  task can only be preempted by another  $c$  task. In all cases, we apply Heuristic 4 to obtain a preemptive schedule.

**Heuristic 4:** Schedule the available  $a$  tasks using the  $SRPT_a$  rule on the preprocessing master. Schedule the available  $c$  tasks using the  $SPT_c$  rule on the postprocessing master.

Let  $m_1$  and  $m_2$  denote the numbers of preprocessing masters and postprocessing masters, respectively. We first study the simple case  $m_1 = m_2 = 1$ .

**Performance analysis of Heuristic 4 when  $m_1 = m_2 = 1$ .** We first consider the sum of completion times. Let  $C_{a_j}^*$  be the time  $a_j$  finishes in an optimal schedule. Then, we have

$$C^* \geq \sum_{j=1}^n (C_{a_j}^* + b_j + c_j) = \left( \sum_{j=1}^n C_{a_j}^* \right) + B + C .$$

Let  $C_{a_j}$  be the time  $a_j$  finishes in the schedule obtained by Heuristic 4. Since the  $SRPT_a$  rule is optimal if  $b_j = c_j = 0$ , Heuristic 4 must have the minimum  $\sum_{j=1}^n C_{a_j}$  among all possible schedules. That is

$$\sum_{j=1}^n C_{a_j} \leq \sum_{j=1}^n C_{a_j}^* .$$

Thus, the sum of completion times is at most

$$\sum_{j=1}^n \left( C_{a_j} + b_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \right) \leq \sum_{j=1}^n (C_{a_j} + b_j + c_j) + \sum_{j=1}^n \sum_{c_i < c_j} c_i \leq 2C^* .$$

For the makespan, consider the last job  $l$  such that  $c_l$  runs immediately when it is available at time  $C_{a_l} + b_l$ . There is no idle time in the interval  $I_1 = [r_l, C_{a_l})$  and the interval  $I_2 = [(C_{a_l} + b_l), C_{\max})$ . The length of each interval is at most  $C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + b_l + |I_2| \leq 3C_{\max}^* .$$

*Theorem 9:* Heuristic 4 generates a  $(3, 2)$  preemptive schedule in  $O(n \log n)$  time when  $m_1 = m_2 = 1$  and jobs have the same or arbitrary release times. Furthermore, it can be implemented online.

Next we assume that  $m_1, m_2 > 1$ . We consider the case with the same release times separate from the case with different release times.

**Performance of Heuristic 4 when  $m_1, m_2 > 1$  and  $r_j = 0$  for all  $j$ .** Since all  $a$  tasks are available at time 0, the  $SRPT_a$  rule is the same as the  $SPT_a$  rule. As we mentioned before, the  $SPT_a$  rule is optimal when the  $b$  tasks and the  $c$  tasks have zero length; that is, it minimizes the sum of completion times of the  $a$  tasks. Let  $C_{a_i}^*$  be the finish time of  $a_i$  in an optimal schedule, and let  $C_{a_i}$  be the finish time of  $a_i$  in the schedule generated by Heuristic 4. Then, as in the case of  $m_1 = m_2 = 1$ , a lower bound of the sum of completion times is

$$C^* \geq \sum_{i=1}^n (C_{a_i}^* + b_j + c_j) = \left( \sum_{i=1}^n C_{a_i}^* \right) + B + C \geq \left( \sum_{i=1}^n C_{a_i} \right) + B + C \quad (8)$$

When the task  $c_j$  is ready, it can be delayed by a task  $c_i$  only if  $c_i < c_j$ . The length of the interval  $[(C_{a_j} + b_j), C_j - c_j]$  is at most  $\sum_{c_i < c_j} \frac{c_i}{m_2}$ , since all postprocessing masters must be busy and can only run the task  $c_i$  such that  $c_i < c_j$  during this interval. Hence the sum of completion times is at most

$$\sum_{j=1}^n C_j \leq \sum_{j=1}^n (C_{a_j} + b_j + c_j + \sum_{c_i < c_j} \frac{c_i}{m_2}) = \left( \sum_{j=1}^n C_{a_j} + B + C \right) + \sum_{j=1}^n \sum_{c_i < c_j} \frac{c_i}{m_2} \leq 2C^* ,$$

where the last inequality comes from (7) and (8).

Now we show that the makespan is at most four times the optimal makespan. As before, let  $k$  be the job with the maximum completion time, and let  $l$  be the last job such that the task  $c_l$  runs immediately after it is ready at  $C_{a_l} + b_l$ . Then the intervals  $I_1 = [0, C_{a_l} - a_l)$  and  $I_2 = [(C_{a_l} + b_l), (C_{\max} - c_k))$  must be both busy. And  $|I_1| \leq \sum_{a_j < a_l} a_j / m_1 < C_{\max}^*$  and  $|I_2| \leq \sum_{c_j < c_l} c_j / m_2 < C_{\max}^*$ . Therefore,

$$C_{\max} = |I_1| + (a_l + b_l) + |I_2| + c_k < 4C_{\max}^*$$

*Theorem 10:* Heuristic 4 generates a  $(4, 2)$  preemptive schedule with migration when jobs have the same release times, and  $m_1 > 1$  and  $m_2 > 1$ .

**Performance of Heuristic 4 when  $m_1, m_2 > 1$  and  $r_j \geq 0$ .** Let  $C_{a_j}^*$  be the completion time of  $a_j$  in an optimal schedule. As in the last section, we have

$$C^* \geq \left( \sum C_{a_j}^* \right) + B + C$$

Let  $C_{a_j}$  be the completion time of  $a_j$  in the schedule generated by Heuristic 4. As we mentioned before, the  $SRPT_a$  rule is a 2-approximation when  $b_j = c_j = 0$ . Thus, we must have

$$\sum C_{a_j} \leq 2 \sum C_{a_j}^*$$

and

$$\begin{aligned}
\sum_{j=1}^n C_j &= \sum_{j=1}^n \left( C_{a_j} + b_j + c_j + \sum_{c_i < c_j} c_i/m_2 \right) \\
&\leq \left( \sum_{j=1}^n C_{a_j} \right) + B + C + \sum_{j=1}^n \sum_{c_i < c_j} c_i/m_2 \\
&\leq \sum C_{a_j}^* + \left( \sum C_{a_j}^* + B + C \right) + \sum_{j=1}^n \sum_{c_i \leq c_j} c_i/m_2 \\
&\leq 3C^*
\end{aligned}$$

Now we consider the makespan. As before, let  $k$  be the job with the maximum completion time, and let  $l$  be the last job such that  $c_l$  runs immediately after it is ready at time  $C_{a_l} + b_l$ . Then the intervals  $I_1 = [r_l, C_{a_l} - a_l)$  and  $I_2 = [(C_{a_l} + b_l), (C_{\max} - c_k))$  must be both busy. And  $|I_1| \leq \sum_{a_j < a_l} a_j/m_1 \leq C_{\max}^*$  and  $|I_2| \leq \sum_{c_j < c_l} c_j/m_2 \leq C_{\max}^*$ . Therefore, we have

$$C_{\max} = r_l + |I_1| + (a_l + b_l) + |I_2| + c_k \leq 4C_{\max}^*$$

*Theorem 11:* Heuristic 4 generates a  $(4, 3)$  preemptive schedule with migration when jobs have arbitrary release times, and  $m_1 > 1$  and  $m_2 > 1$ .

## 5 CONVERTING PREEMPTIVE SCHEDULES INTO NON-PREEMPTIVE SCHEDULES

As we mentioned before, we obtain non-preemptive schedules by converting from preemptive schedules. Our approach is based on the technique introduced by Phillips et. al in [18], and improved by Chekuri et. al in [3]. For completeness, we describe their approaches in the following.

The model studied in [18] consists of one or more identical machines and  $n$  jobs. For this model, a general approach of converting a preemptive schedule  $S$  into a non-preemptive schedule  $S'$  has been given in [18]. The idea is to form a list of jobs in ascending order of their completion times in  $S$  and then list schedule the jobs in this list one by one, respecting their release times.

Let  $C_j$  and  $C'_j$  denote the completion time of job  $j$  in  $S$  and  $S'$ , respectively. Suppose the jobs are indexed such that  $C_i < C_{i+1}$ . Then they showed that  $C'_j \leq 2C_j$  for a single machine environment and  $C'_j \leq 3C_j$  for a multi-machine environment. Let  $r_{\max}^j = \max_{1 \leq i \leq j} r_i$ . The result is based on the observations that (1)  $C_j > r_{\max}^j$ , (2)  $C_j \geq \sum_{i=1}^j p_i$  in the single-machine case, and  $C_j \geq \sum_{i=1}^j p_i/m$  in the multi-machine case, where  $p_j$  is the processing time of job  $j$ , (3)  $C'_j \leq r_{\max}^j + \sum_{i=1}^j p_i$  if there is one machine and  $C'_j \leq r_{\max}^j + (\sum_{i=1}^{j-1} p_i/m) + p_j$  if there are two or more machines. These results imply that if  $S$  is a  $\beta$ -approximation for the sum of completion times, then  $S'$  is a  $2\beta$ -approximation for the sum of completion times in the single-machine environment, and a  $3\beta$ -approximation for the sum of completion times in the multi-machine environment. In addition, this conversion also yields an online non-preemptive algorithm if the preemptive schedule can be generated online: simply simulate the algorithm for preemptive schedule, start a job  $j$  if  $j$  completes in the preemptive schedule or put it in the waiting queue if the machine is busy.

Now, let us consider the makespan which is equal to the completion time of job  $n$  in  $S'$ . It is easy to see that  $C'_n \leq C_n + \sum_{j=1}^n p_j \leq C_n + C_{\max}^*$ . Therefore, if  $S$  is an  $\alpha$ -approximation for makespan, i.e.,  $C_{\max}(S) = C_n \leq \alpha C_{\max}^*$ , then  $C_{\max}(S') = C'_n \leq (1 + \alpha) C_{\max}^*$ . In other words,  $S'$  is an  $(\alpha + 1)$ -approximation for makespan.

Later, Chekuri et. al [3] improved the above results. They designed a deterministic  $O(n^2)$  time *offline* algorithm such that the schedule obtained has the sum of completion times at most  $\frac{e}{e-1}$  times that of the preemptive schedule, where  $e$  is the natural log. As well, they gave a randomized *online* algorithm with *expected* performance  $\frac{e}{e-1}$ . The difference between the two approaches in [3] and [18] is how to obtain

the list of jobs. Given  $S$  and a parameter  $\lambda \in (0, 1]$ , let  $C_j^\lambda(S)$ , the  $\lambda$ -point of  $j$ , be the time when  $\lambda p_j$  (a  $\lambda$ -fraction) of job  $j$  is completed. Instead of forming a list based on  $C_j(S)$ , now we form a list based on  $C_j^\lambda(S)$ . A  $\lambda$ -schedule is a non-preemptive schedule obtained by list scheduling jobs in ascending order of  $C_j^\lambda(S)$ , possibly introducing idle time to account for the release times. It is easy to see that the algorithm given by [18] is a  $\lambda$ -schedule with  $\lambda = 1$ . It is also clear that a  $\lambda$ -scheduler can be made to be an on-line algorithm if the underlying preemptive algorithm is an on-line algorithm.

The main result in [3] is that for each given instance, there exists a  $\lambda$ , the best  $\lambda$ , such that the  $\lambda$ -schedule has the sum of completion times at most  $\frac{e}{e-1}$  times that of  $S$ , and has makespan at most  $(1 + \lambda)$  times that of  $S$ . One can obtain such a  $\lambda$ -schedule by finding the best  $\lambda$  in  $O(n^2)$  time offline deterministically, or one can obtain through a randomized on-line algorithm a schedule with *expected* sum of completion times at most  $\frac{e}{e-1}$  times and *expected* makespan at most  $(1 + \lambda)$  times that of  $S$ .

In the multi-machine case, Chakrabarti et al. [4] showed that the convert procedure given in [18] has a bound of  $\frac{7}{3}$ , instead of 3 times that of  $S$ .

In the following sections, we will describe how to convert preemptive schedules generated by Heuristics 1-4 to non-preemptive schedules. The difficulty of our conversion is that we need to respect not only the release time of  $a_i$ ,  $1 \leq i \leq n$ , but also respect the constraint that the interval between the finish time of  $a_i$  and the start time of  $c_i$  has length at least  $b_i$ . By our convention, we use  $S$  to denote a  $(\alpha, \beta)$  preemptive schedule,  $C_{a_j}$  to denote the completion time of  $a_j$  in  $S$ ,  $C_j$  to denote the completion time of  $S$ , and  $C_{\max}$  to be the makespan of  $S$ . Our goal is to convert  $S$  into a non-preemptive schedule  $S'$ . We define  $C'_{a_j}$ ,  $C'_j$ ,  $C'_{\max}$  similarly for the non-preemptive schedule  $S'$ .

### 5.1 Single master and multi-master systems

*Theorem 12:* In  $O(n^2)$  time, one can obtain a  $(\frac{5}{2}, \frac{2e}{e-1})$  non-preemptive canonical schedule when there is a single master and  $r_j = 0$  for all  $j$ .

*Proof:* Let  $S$  be a preemptive canonical schedule of  $n$  jobs obtained by applying Corollary 2. Let  $S_a$  be the partial schedule of  $S$  during the interval  $(0, A]$ , and  $S_c$  be the partial schedule of  $S_c$  during the interval  $(A, C_{\max}]$ .

Clearly  $S_a$  contains all  $a$  tasks only. By Heuristic 1, there is no preemption in  $S_a$ . Let  $C_{a_j}$  be the completion time of  $a_j$ . It is easy to see that the partial schedule  $S_c$  contains all  $c$  tasks only and it can be seen as a preemptive schedule of  $n$  tasks on a single machine where each task  $j$  has a ‘‘release time’’  $\max(A, C_{a_j} + b_j)$  and processing time  $c_j$ .

To convert  $S$  into a non-preemptive schedule  $S'$ , we fix  $S_a$  and convert  $S_c$  to a non-preemptive schedule  $S'_c$  of  $c$  tasks by using the approach of [3]. Let  $C_j$  and  $C'_j$  be the completion time of  $c_j$  in  $S$  and  $S'$ , respectively. As mentioned at the beginning of the section, it has been shown in [3] that  $C'_j \leq \frac{e}{e-1} C_j$  and  $C'_j \leq C_j + C_{\max}$ . Since  $S$  is a  $(\frac{3}{2}, 2)$  canonical schedule, the obtained schedule is a  $(\frac{5}{2}, \frac{2e}{e-1})$  non-preemptive canonical schedule. This concludes the proof. ■

*Theorem 13:* In  $O(n \log n)$  time, one can obtain a  $(3, 4)$  online non-preemptive schedule when there is a single master and  $r_j \geq 0$  for all job  $j$ .

*Proof:* Let  $S$  be a non-canonical preemptive schedule  $S$ . One can get a  $\lambda$ -schedule  $S'$ ,  $\lambda = 1$ , similarly as in [18]: (1) Sort all tasks (both  $a$  tasks and  $c$  tasks) in ascending order of their completion times. (2) List schedule these tasks on the master machine, with the constraint that each task  $a_j$  must start after  $r_j$  and the interval between the time  $a_j$  finishes and the time  $c_j$  starts is at least  $b_j$ .

For the purpose of analysis, we can visualize the above procedure as follows (see Figure 1): For each task,  $a_j$  or  $c_j$ , remove all but the last scheduled piece of the task. Suppose the last piece of  $a_j$  and  $c_j$  have length  $k_{a_j}$  and  $k_{c_j}$ , respectively. Now process the tasks one by one in the scheduling order of their last pieces in  $S$ . If the current task is  $a_j$ , complete its last piece by inserting an extra piece with length  $(a_j - k_{a_j})$  immediately after the last piece of  $a_j$ ; at the same time push backward in time all the last pieces of the tasks which finish after  $a_j$  in  $S$  by an amount of  $(a_j - k_{a_j})$ . If the task is  $c_j$ , complete its last piece by inserting an extra piece with length  $(c_j - k_{c_j})$  immediately after the last piece of  $c_j$ ; at the

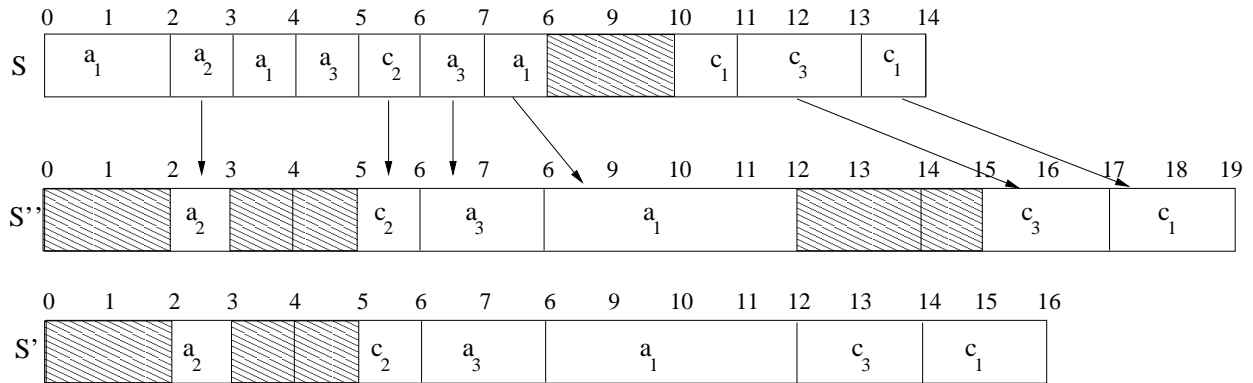


Fig. 1. Convert a preemptive schedule into a non-preemptive schedule:  $S$  is a preemptive schedule of three jobs  $J_1(0, 4, 2, 2)$ ,  $J_2(2, 1, 2, 1)$ ,  $J_3(4, 2, 4, 2)$ ;  $S''$  is the non-preemptive schedule obtained from  $S$  by completing the last piece of each task into a whole piece;  $S'$  is the non-preemptive schedule obtained from  $S''$  by removing unnecessary idle intervals in  $S''$ .

same time push backward in time all the last pieces of the tasks which finish after  $c_j$  in  $S$  by an amount of  $(c_j - k_{c_j})$ . Let the schedule be  $S''$ . It is easy to see that during this process, the two constraints, (a) each task  $a_j$  is scheduled after  $r_j$  and (b) the interval between the time  $a_j$  finishes and the time  $c_j$  starts is at least  $b_j$ , are not violated. Thus  $S''$  is still a feasible schedule.

Now one pushes all jobs forward in time as much as possible without changing the order of the tasks, or violating the constraints (a) and (b) mentioned above. The result is exactly the schedule  $S'$ . Since a task  $c_j$  can only be moved back by processing times associated with tasks finished earlier than  $c_j$  in  $S$ , we have  $C'_j \leq 2C_j$  and  $C'_j \leq C_j + \sum_{i=1}^n (a_i + c_i) \leq C_j + C_{\max}$ , where  $C'_j$  is the completion time of  $c_j$  in  $S'$ . This implies that if we take the (2, 2) non-canonical schedule in Theorem 4, then we get a (3, 4) non-preemptive schedule. Furthermore, it can be implemented online if the preemptive schedule is online. ■

The following theorem shows how to get offline non-preemptive schedules for multi-master systems. We do not know how to obtain an online non-preemptive schedule.

*Theorem 14:* When there are multi-masters, one can obtain a (5, 4) non-preemptive offline schedule.

*Proof:* Let  $S$  be the (2, 2)-schedule generated by Heuristic 3. Then  $S$  has no migration. One can obtain the non-preemptive schedule  $S'$  by converting the schedule on each machine separately in the same way as described in the proof of Theorem 13. Thus, the sum of completion times of  $S'$  is at most twice that of  $S$ . Since  $S$  is a 2-approximation for the sum of completion times,  $S'$  is a 4-approximation for the sum of completion times. For the makespan,  $C'_j \leq C_j + \max_{i=1}^m \Delta_i$  where  $\Delta_i$  is the total length of the  $a$  tasks and the  $c$  tasks assigned on machine  $i$ . In Section 3, it has been shown that  $\Delta_i \leq 2C_{\max}^*$ . Thus,  $C'_j \leq C_j + \max_{i=1}^m \Delta_i \leq C_j + 2C_{\max}^*$ . Since  $S$  is a 3-approximation for the makespan,  $S'$  is a 5-approximation for the makespan. This concludes the proof. ■

## 5.2 Distinct preprocessors and postprocessors

This subsection considers the case when there are  $m_1$  preprocessors and  $m_2$  postprocessors.

*Theorem 15:* In  $O(n^2)$  time, one can obtain a  $(4, \frac{2e}{e-1})$  non-preemptive schedule when  $r_j = 0$  for all  $j$ , and  $m_1 = m_2 = 1$ .

*Proof:* Let  $S$  be the (3, 2)-schedule generated by Heuristic 4. Since  $r_j = 0$  for all  $j$ , there is no preemption on the single preprocessor. Let  $C_{a_j}$  be the completion time of  $a_j$  in  $S$ . To get a non-preemptive schedule, we fix the schedule of the  $a$  tasks and do the conversion simply on the single postprocessor using exactly the approach given in [3], respecting the “release time” of task  $c_j$  (which is  $C_{a_j} + b_j$ ). Following exactly the same argument, we can show that  $S'$  is a  $(4, \frac{2e}{e-1})$  non-preemptive schedule. ■

When the release times are arbitrary, we need to do the conversion carefully so as to make sure that the difference between the finish time of  $a_j$  and the start time of  $c_j$  is at least  $b_j$ .

*Theorem 16:* When  $m_1 = m_2 = 1$ , and  $r_j \geq 0$ , one can obtain a  $(4, 4 + \frac{2e}{e-1})$  non-preemptive offline schedule or an online non-preemptive schedule with expected performance  $(4, 4 + \frac{2e}{e-1})$ .

*Proof:* Let  $S$  be the  $(3, 2)$ -schedule generated by Heuristic 4. Let  $C_j$  be the completion time of job  $j$  in  $S$ . Let  $C_{a_j}$  be the completion time of  $a_j$  in  $S$ .

The conversion consists of two steps. First we remove the preemptions among the  $a$  tasks to get the best (offline)  $\lambda$ -schedule of the  $a$  tasks on the single preprocessor by using the approach of [3]. Let  $C'_{a_j}$  be the new completion time of  $a_j$ . Then, we must have  $C'_{a_j} \leq \frac{e}{e-1} C_{a_j}$ . Now fix the schedule of  $a$  tasks, and remove the preemptions among the  $c$  tasks to get a  $\lambda$ -schedule of the  $c$  tasks, where  $\lambda = 1$ , and make sure the interval between  $C'_{a_j}$  and the start time of  $c_j$  is at least  $b_j$  for each job  $j$ .

Suppose the jobs are indexed so that  $C_i < C_{i+1}$ . Then,  $C_j \geq \max(\max_{i=1}^j r_i, \max_{i=1}^j b_i, \sum_{i=1}^j a_i, \sum_{i=1}^j c_i)$ . Let the new schedule be  $S'$  and let the completion time of  $c_j$  in  $S'$  be  $C'_j$ . Then

$$C'_j \leq C'_{a_j} + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^j c_i \leq \frac{e}{e-1} C_{a_j} + 2C_j \leq (2 + \frac{e}{e-1}) C_j .$$

Thus,  $\sum_{j=1}^n C'_j \leq (2 + \frac{e}{e-1}) \sum_{j=1}^n C_j$ . By assumption,  $S$  is 2-approximation for the sum of completion times. Thus,  $S'$  is  $(4 + \frac{2e}{e-1})$ -approximation for the sum of completion times.

For the makespan,

$$C'_j \leq \max_{1 \leq i \leq j} r_i + \sum_{C_{a_i} \leq C_{a_j}} a_i + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^j c_i \leq 4C_{\max}^* .$$

Thus,  $S'$  is a  $(4, 4 + \frac{2e}{e-1})$  non-preemptive offline schedule. Similarly, as in [3], we can also get an online schedule whose expected performance is  $(4, 4 + \frac{2e}{e-1})$ . This concludes the proof. ■

*Theorem 17:* In  $O(n \log n)$  time, one can obtain a  $(4, 14/3)$  non-preemptive schedule when  $r_j = 0$  for all  $j$ ,  $m_1 \geq 1$  and  $m_2 \geq 1$ .

*Proof:* Let  $S$  be the  $(4, 2)$ -schedule generated by Heuristic 4. Since all jobs have the same release time, no preemption occurs on the preprocessors. Now fix the schedule of the  $a$  tasks. Let  $C_{a_j}$  be the completion time of  $a_j$  in  $S$ . The task  $c_j$  can be seen as a task with release time  $(C_{a_j} + b_j)$ . The conversion is performed on the postprocessors using the approach given in [18], subject to the constraint that  $c_j$  can not start earlier than its “release time” (which is  $C_{a_j} + b_j$ ). Then, by [4], the sum of completion times of  $S'$  is at most  $\frac{7}{3}$  times that of  $S$ , i.e.,  $S'$  is a  $\frac{14}{3}$ -approximation for the sum of completion times.

Now consider the makespan. Suppose the jobs are indexed such that  $C_i \leq C_{i+1}$  in  $S$ . By the algorithm, for any  $1 \leq i \leq n$ ,  $C_{a_i} + b_i \leq \sum_{k \neq i} a_k/m_1 + (a_i + b_i) \leq 2C_{\max}^*$  and  $\sum_{i=1}^j c_i/m_2 \leq C_{\max}^*$ . In  $S'$ , the latest time the postprocessors become busy is  $\max_{i \leq j} (C_{a_i} + b_i)$ . Therefore,  $C'_j \leq \max_{i \leq j} (C_{a_i} + b_i) + \sum_{i=1}^{j-1} c_i/m_2 + c_j \leq 4C_{\max}^*$ . ■

*Theorem 18:* In  $O(n \log n)$  time, one can obtain a  $(5, 13)$  non-preemptive online schedule when  $m_1 \geq 1$  and  $m_2 \geq 1$ .

*Proof:* Let  $S$  be a  $(4, 3)$ -schedule generated by Heuristic 4. The conversion consists of two steps. First, we use the approach of [18] to remove preemptions among the  $a$  tasks to get a  $\lambda = 1$  schedule of the  $a$  tasks, respecting the release times of the  $a$  tasks. Let  $C'_{a_j}$  be the new completion time of  $a_j$ . By the result of [4], we must have  $C'_{a_j} \leq \frac{7}{3} C_{a_j}$ . Now, fix the schedule of the  $a$  tasks. Each task  $c_j$  can be seen as a task with a release time  $(C'_{a_j} + b_j)$ . Next, we remove preemptions among the  $c$  tasks to get a  $\lambda$ -schedule of the  $c$  tasks, where  $\lambda = 1$ , and make sure that the interval between  $C'_{a_j}$  and the start time of  $c_j$  is at least  $b_j$  for each job  $j$ .

Let  $C_j$  be the completion time in the preemptive schedule. Suppose the jobs are indexed such that  $C_i < C_{i+1}$  in  $S$ . Then,  $C_j \geq \max_{1 \leq i \leq j} (C_{a_i} + b_i + c_i)$  and  $C_j > \sum_{i=1}^j c_i/m_2$ . Let  $C'_j$  be the new completion time of job  $j$ . Then

$$C'_j \leq \max_{1 \leq i \leq j} (C'_{a_i} + b_i) + \sum_{1 \leq i < j} \frac{c_i}{m_2} + c_j \leq \max_{1 \leq i \leq j} (\frac{7}{3} C_{a_i} + b_i) + \sum_{1 \leq i < j} \frac{c_i}{m_2} + c_j \leq \frac{13}{3} C_j .$$

Thus,  $\sum_{j=1}^n C'_j \leq \frac{13}{3} \sum_{j=1}^n C_j$ . By assumption,  $S$  is a 3-approximation for the sum of completion times. Thus,  $C'_j$  is a 13-approximation for the sum of completion times.

For the makespan,  $C_{\max}^* \geq \max(A/m_1, C/m_2, \max(a_j + b_j + c_j))$ . Therefore,

$$\begin{aligned} C'_j &\leq \max_{1 \leq k \leq n} r_k + \left( \sum_{C_{a_k} < C_{a_j}} a_k/m_1 \right) + (a_j + c_j) + \max_{C_i < C_j} b_j + \left( \sum_{1 \leq i < j} \frac{c_i}{m_2} \right) \\ &\leq 5 C_{\max}^* \end{aligned}$$

Thus,  $S'$  is a (5, 13)-schedule. ■

## 6 LINEAR PROGRAMMING: DISTINCT PREPROCESSORS AND POSTPROCESSORS

As we mentioned before, another important approach for NP-hard scheduling problems is to formulate the problem as a linear programming problem. This method has the advantage that it also works for the sum of weighted completion times. However, its disadvantage is that it takes relatively long time to obtain a solution. In this section we present approximation algorithms for the case when there are distinct preprocessing and postprocessing processors. We also allow each job  $j$  to have a weight  $w_j$ .

The basis of our approximation algorithms is a linear programming relaxation that uses as variables the completion times of the  $a$  tasks and the  $c$  tasks. For each task  $j$ , we define  $C_{a_j}$  and  $C_j$  to be the completion time of  $a_j$  and  $c_j$ , respectively. We can formulate the sum of weighted completion times minimization problem as follows:

$$\text{minimize } \sum_{j=1}^n w_j C_j \tag{9}$$

subject to

$$C_{a_j} \geq r_j + a_j \tag{10}$$

$$C_j \geq C_{a_j} + b_j + c_j \tag{11}$$

$$C_j \geq C_k + c_j \text{ or } C_k \geq C_j + c_k \quad \text{for each pair } j \neq k \tag{12}$$

$$C_{a_j} \geq C_{a_k} + a_j \text{ or } C_{a_k} \geq C_{a_j} + a_k \quad \text{for each pair } j \neq k \tag{13}$$

### 6.1 $m_1 = m_2 = 1$

The difficulty with the above characterization is the so-called ‘‘disjunctive’’ constraints (12)-(13), which are not linear inequalities and cannot be modeled using linear inequalities. Instead, we use a class of valid inequalities for any feasible schedules (maybe preemptive), introduced by Queyranne [19] and Wolsey [31]. Suppose a set  $N$  of  $n$  tasks, denoted by  $1, 2, \dots, n$ , are scheduled on a single machine. Let  $p_j$  be the processing time of  $j$  and  $C_j$  be the completion time of  $j$  in any feasible schedule. Then, the following inequality is valid for any subset  $X \subseteq N$ .

$$\sum_{j \in X} p_j C_j \geq \frac{1}{2} \left( \sum_{j \in X} p_j^2 + \left( \sum_{j \in X} p_j \right)^2 \right) \quad \text{for each } X \subseteq N \tag{14}$$

The key to the quality of the approximation deriving from the above relaxations is the following lemma.

*Lemma 19:* ([10],[22]) Let  $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_n$  satisfy (14), and assume without loss of generality that  $\bar{C}_1 < \bar{C}_2 < \dots < \bar{C}_n$ . Then, for each job  $j = 1, \dots, n$ ,

$$\bar{C}_j \geq \frac{1}{2} \sum_{k=1}^j p_k$$

A feasible solution  $\bar{C}_1 < \bar{C}_2 < \dots < \bar{C}_n$  satisfying (14) need not correspond to a feasible schedule. Lemma 19 states that merely satisfying the constraints (14) is sufficient to obtain a relaxation of this:  $\bar{C}_j \geq \frac{1}{2} \sum_{k=1}^j p_k$ .

Queyranne [19] has shown that the linear program with constraints (14) is solvable in polynomial time via the ellipsoid algorithm; the key observation is that there is a polynomial time separation algorithm for the exponentially large class of constraints given by (14).

In our model, we can apply the above constraints to the preprocessing master and the postprocessing master, respectively.

$$\sum_{k \in X} a_k C_{a_k} \geq \frac{1}{2} \left( \sum_{j \in X} a_j^2 + \left( \sum_{j \in X} a_j \right)^2 \right) \quad \text{for each } X \subseteq N \quad (15)$$

$$\sum_{k \in X} c_k C_k \geq \frac{1}{2} \left( \sum_{j \in X} c_j^2 + \left( \sum_{j \in X} c_j \right)^2 \right) \quad \text{for each } X \subseteq N \quad (16)$$

**Heuristic 5:** First obtain an optimal solution to the linear program formed by (9), (10), (11), (15) and (16). Denote the completion time of the jobs by  $\bar{C}_1, \dots, \bar{C}_n$ . Then we form a schedule by scheduling both the  $a$  tasks and the  $c$  tasks in increasing order of  $\bar{C}_j$  under the condition that  $a_j$  can not start until  $r_j$ , and the interval between the start time of  $c_j$  and the finish time of  $a_j$  is at least  $b_j$ , i.e.,  $c_j$  can not start until  $b_j$  finishes.

Let  $C_{a_j}$  be the completion time of  $a_j$  in the schedule obtained by Heuristic 5. By the way we schedule the  $a$  tasks,  $C_{a_j} \leq \max_{k \leq j} \{r_k\} + \sum_{k \leq j} a_k$  since the latest time the preprocessor becomes busy is  $\max_{k \leq j} \{r_k\}$ . Let  $C_j$  be the completion time of job  $j$  in the schedule obtained by Heuristic 5. Because we schedule the  $c$  tasks in the same order as the  $a$  tasks, it is possible that we can not start  $c_j$  even after  $b_j$  finishes at  $C_{a_j} + b_j$ . This is because  $c_{j-1}$  may not have completed yet. However, the time that we need to wait after  $c_j$  is ready is at most  $\max_{k \leq j} \{b_k\} + \sum_{k \leq j-1} c_k$ . Thus, we have

$$\begin{aligned} C_j &\leq C_{a_j} + \max_{k \leq j} \{b_k\} + \sum_{k=1}^{j-1} c_k + c_j \\ &\leq \left( \max_{k \leq j} \{r_k\} + \sum_{k=1}^j a_k \right) + \max_{k \leq j} \{b_k\} + \sum_{k=1}^j c_k \\ &\leq \max_{k \leq j} \{r_k\} + 2\bar{C}_{a_j} + \max_{k \leq j} \{b_k\} + 2\bar{C}_j \quad \text{by Lemma 19} \\ &\leq \max_{k \leq j} \{r_k\} + 5\bar{C}_j \end{aligned}$$

Thus, if all jobs have the same release times, then  $C_j \leq 5\bar{C}_j$ ; otherwise, we have  $C_j \leq 6\bar{C}_j$ .

For the makespan, we have

$$\begin{aligned} C_j &\leq C_{a_j} + \max_{k \leq j} b_k + \sum_{k=1}^j c_k \\ &\leq \left( \max_{k \leq j} \{r_k\} + \sum_{k=1}^j a_k \right) + \max_{k \leq j} \{b_k\} + \sum_{k=1}^j c_k \\ &\leq \max_{k \leq j} \{r_k\} + 3C_{\max}^* \end{aligned}$$

Thus, if all jobs have the same release times, then  $C_j \leq 3C_{\max}^*$ ; otherwise, we have  $C_j \leq 4C_{\max}^*$ .

*Theorem 20:* Suppose there are a single preprocessing master and a single postprocessing master. Furthermore, each job has an arbitrary weight. Then, Heuristic 5 produces a (4, 6)-schedule when each job has an arbitrary release time and a (3, 5)-schedule when all jobs have the same release time. Furthermore, the  $a$  tasks and the  $c$  tasks complete in the same order.

## 6.2 $m_1 \geq 1$ and $m_2 \geq 1$

If the jobs are scheduled on  $m$  machines, then we have similar valid inequalities which were also made by Queyranne [20].

$$\sum_{k \in X} p_k C_k \geq \frac{1}{2m} \left( \sum_{j \in X} p_j^2 + \left( \sum_{j \in X} p_j \right)^2 \right) \quad \text{for each } X \subseteq N \quad (17)$$

We have a similar lemma as in the single machine case.

*Lemma 21:* Let  $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_n$  satisfy (17), and assume without loss of generality that  $\bar{C}_1 < \bar{C}_2 < \dots < \bar{C}_n$ . Then, for each job  $j = 1, \dots, n$ ,

$$\bar{C}_j \geq \frac{1}{2m} \sum_{k=1}^j p_k$$

In our model, we can apply the above constraints to the  $m_1$  preprocessing masters and the  $m_2$  postprocessing masters, respectively.

$$\sum_{k \in X} a_k C_{a_k} \geq \frac{1}{2m_1} \left( \sum_{j \in X} a_j^2 + \left( \sum_{j \in X} a_j \right)^2 \right) \quad \text{for each } X \subseteq N \quad (18)$$

$$\sum_{k \in X} c_k C_k \geq \frac{1}{2m_2} \left( \sum_{j \in X} c_j^2 + \left( \sum_{j \in X} c_j \right)^2 \right) \quad \text{for each } X \subseteq N \quad (19)$$

**Heuristic 6:** First obtain an optimal solution to the linear program given by (9), (10), (11), (18) and (19). Denote the completion time of the jobs by  $\bar{C}_1, \dots, \bar{C}_n$ . Schedule both the  $a$  tasks and the  $c$  tasks one by one in ascending order of  $\bar{C}_j$  under the condition that:  $a_j$  can not start until  $r_j$ , and the interval between the finish time of  $a_j$  and the start time of  $c_j$  is at least  $b_j$ . In other words,  $c_j$  can not start until  $b_j$  finishes.

Let  $C_{a_j}$  be the completion time of task  $a_j$  in the schedule obtained by Heuristic 6. By the way we schedule the  $a$  tasks, the latest time the preprocessor becomes busy is  $\max_{k \leq j} \{r_k\}$ . Then we schedule  $a_k$ ,  $k \leq j-1$ , one by one. By time  $t \leq \max_{k \leq j} \{r_k\} + \sum_{k \leq j-1} a_k/m_1$ , the first  $j-1$  tasks must be finished and there will be an idle machine to process the  $j^{\text{th}}$  task. Thus,  $C_{a_j} \leq \max_{k \leq j} \{r_k\} + \sum_{k \leq j-1} a_k/m_1 + a_j$ . Let  $C_j$  be the completion time of job  $j$  in the schedule obtained by Heuristic 6. Because we schedule the  $c$  tasks in the same order as the  $a$  tasks, it is possible that we can not start  $c_j$  even after  $b_j$  finishes at  $C_{a_j} + b_j$ , because  $c_{j-1}$  may not have completed yet. However, the time that we need to wait after  $c_j$  is ready is at most  $\max_{k \leq j} \{b_k\} + \sum_{k \leq j-1} c_k/m_2$ . Thus, we have

$$\begin{aligned} C_j &\leq C_{a_j} + \max_{k \leq j} \{b_k\} + \frac{1}{m_2} \sum_{k=1}^{j-1} c_k + c_j \\ &\leq \left( \max_{k \leq j} \{r_k\} + \frac{1}{m_1} \sum_{k=1}^{j-1} a_k + a_j \right) + \max_{k \leq j} \{b_k\} + \frac{1}{m_2} \sum_{k=1}^{j-1} c_k + c_j \\ &\leq \max_{k \leq j} \{r_k\} + 2 \max_{k \leq j-1} \bar{C}_{a_k} + \max_{k \leq j} \{b_k\} + 2\bar{C}_j + (a_j + c_j) \\ &\leq \max_{k \leq j} \{r_k\} + 6\bar{C}_j \end{aligned}$$

Thus, if all jobs have the same release times, then  $C_j \leq 6\overline{C}_j$ ; otherwise, we have  $C_j \leq 7\overline{C}_j$ . We now turn our attention to the makespan. We have

$$\begin{aligned}
C_j &\leq C_{a_j} + \max_{k \leq j} \{b_k\} + \frac{1}{m_2} \sum_{k=1}^{j-1} c_k + c_j \\
&\leq \left( \max_{k \leq j} \{r_k\} + \frac{1}{m_1} \sum_{k=1}^{j-1} a_k + a_j \right) + \max_{k \leq j} \{b_k\} + \frac{1}{m_2} \sum_{k=1}^{j-1} c_k + c_j \\
&\leq \max_{k \leq j} \{r_k\} + \frac{1}{m_1} \sum_{k=1}^{j-1} a_k + \max_{k \leq j} \{b_k\} + \frac{1}{m_2} \sum_{k=1}^{j-1} c_k + (a_j + c_j) \\
&\leq \max_{k \leq j} \{r_k\} + 4C_{\max}^*
\end{aligned}$$

Thus, if all jobs have the same release times, then  $C_j \leq 4C_{\max}^*$ ; otherwise, we have  $C_j \leq 5C_{\max}^*$ .

*Theorem 22:* Suppose there are  $m_1$  preprocessing masters and  $m_2$  postprocessing masters. Furthermore, each job has an arbitrary weight. Then, Heuristic 6 produces a  $(5, 7)$ -schedule when each job has an arbitrary release time and a  $(4, 6)$ -schedule when all jobs have the same release times.

## 7 CONCLUSION

In this paper we have considered the problem of minimizing the sum of completion times in the master-slave model in various settings. We designed efficient algorithms to generate preemptive schedules with good performance ratio. In most cases, these schedules have small makespan as well. Then we convert the preemptive schedules into non-preemptive schedules using the techniques developed in [18], [4] and [3]. We also use linear programming relaxation to guide us to obtain good non-preemptive schedules directly. In some cases, this technique gives better performance ratio than the method of converting preemptive schedules into non-preemptive schedules.

Several problems remain open. We are not able to find good canonical schedules in multi-master systems. The difficulty is in obtaining a good lower bound of the optimal schedule in this case. In some cases, the non-preemptive schedules obtained by the conversion procedure give us a rather large performance ratio. It is worthwhile to consider using different techniques such as linear programming relaxation. Most of the performance bounds are quite loose. For future research, it is worthwhile to tighten the performance bounds.

## REFERENCES

- [1] M. Dell'Amico. Shop problems with two machines and time lags. *Operations Research*, 44(5), pp. 777-787, 1996.
- [2] R.E. Buten and V.Y. Shen. A scheduling model for computer systems with two classes of processors. *Sagamore Computer Conference on Parallel Processing*, pp. 130-138, 1973.
- [3] C. Chekuri, R. Motwani, B. Natarajan and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1), pp. 146-166, 2001.
- [4] S. Chakrabarti, C. Phillips, A. Schulz, D.B. Shmoys, C. Stein and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pp. 646-657, 1996
- [5] J. Du and J.Y-T. Leung. Minimizing mean flow time in two-machine open shops and flow shops. *Journal of Algorithms*, 14:24-44, 1993.
- [6] M.X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms*, pp. 591-598, 1997.
- [7] J.N.D. Gupta. Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 38, pp. 359-364, 1988.
- [8] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
- [9] T.F. Gonzalez and S. Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, 26, pp. 26-52, 1978.
- [10] L.A. Hall, A.S. Schulz, D.B. Shmoys and J. Wein. Scheduling to minimize average completion time: Offline and online algorithms. *Mathematics of Operations Research*, 22, pp. 513-544, 1997.
- [11] J.A. Hoogeveen and T. Kawaguchi. Minimizing sum of the completion times in a two machine flowshop: Analysis of special cases. *Mathematics of Operations Research*, 24(4), 887-910, 1999.

- [12] L.A. Hall. Approximability of flow shop scheduling. *Mathematical Programming*, 82, pp. 175–190, 1998.
- [13] S.M. Johnson. Optimal two and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, pp. 61-68, 1954.
- [14] W. Kern and W. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. University of Twente, 1993.
- [15] M.A. Langston. Interstage transportation planning in the deterministic flow-shop environment. *Operations Research*, 35(4), pp. 556-564, 1987.
- [16] C.-Y. Lee and G.L. Vairaktarakis. Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16, pp. 149-158, 1994.
- [17] J.Y-T. Leung and H. Zhao. Minimizing mean flowtime and makespan on master-slave systems. *Journal of Parallel and Distributed Computing*, 65, pp. 843-856, 2005.
- [18] C. Phillips, C. Stein and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199-223, 1998.
- [19] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263-285, 1993.
- [20] M. Queyranne. Personal communication, 1995.
- [21] S. Sahni. Scheduling master-slave multiprocessor systems. *IEEE Trans. on Computers*, 45(10), pp. 1195-1199, 1996.
- [22] A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. *Proceedings of the 5th Integer Programming and Combinatorial Optimization (IPCO)*, pp. 301-315, 1996.
- [23] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687-690, 1968.
- [24] D. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197-199, 1976.
- [25] W. Szwarc. Flow-shop problems with time lags. *Management Science*, 29, pp. 447-491, 1983.
- [26] C. Sriskandarajah and S.P. Sethi. Scheduling algorithms for flexible flowshops: worst and average case performance. *European Journal of Operational Research*, 43, pp. 143-160, 1989.
- [27] A.S. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria In *Proceedings of the Fifth Annual European Symposium on Algorithms*, pp. 416 429, 1997.
- [28] S. Sahni and G. Vairaktarakis. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9, pp. 357-377, 1996.
- [29] S. Sahni and G. Vairaktarakis. The master-slave scheduling model. In J. Y-T. Leung (Ed): *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL, 2004.
- [30] G. Vairaktarakis. Analysis of algorithms for master-slave system. *IIE Transactions*, 29(11), pp. 939-949, 1997.
- [31] L.A. Wolsey. Mixed integer programming formulations for production planning and scheduling problems. Invited talk at the 12th International Symposium on Mathematical Programming, MIT, Cambridge, 1985.
- [32] W. Yu, H. Hoogeveen and J.K. Lenstra. Minimizing makespan in a two-machine flowshop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5), 333 - 348, 2004.