

# Minimizing Mean Flowtime on Master-Slave Machines

Joseph Y-T. Leung\*

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

Hairong Zhao†

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

## Abstract

*The master-slave scheduling model is a new model recently introduced by Sahni. It has many important applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc. In this model, each job has a preprocessing task, a slave task and a postprocessing task that must be executed in this order. While the preprocessing and postprocessing tasks are scheduled on the master machine, the slave tasks are scheduled on the slave machines. We consider the problem of minimizing mean flowtime in the single-master master-slave model. We prove that the problem is strongly NP-hard even if all the preprocessing and postprocessing tasks have unit execution time. We then give a polynomial-time optimal algorithm for a special case. Finally, we give some heuristics for other special cases and we analyze their worst-case performance.*

**Keywords:** master-slave, scheduling, mean flowtime, heuristic, NP-hard

## 1 Introduction

In this paper, we consider scheduling problems in the master-slave model which was recently introduced by Sahni [3]. With this model, we are given a set of  $n$  jobs and a set of  $m$  machines. Each job has a preprocessing task, a

slave task and a postprocessing task that must be executed in this order. 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  $c_i > 0$ . Each machine is either a master machine or a slave machine. In this paper, we assume that there are a single master machine and  $n$  slave machines which are used to schedule the  $n$  slave tasks, one for each job. While the preprocessing and postprocessing tasks are scheduled on the master machine, each slave task is scheduled on a dedicated slave machine.

The master-slave model finds many applications in parallel computer scheduling. For example, suppose that there is a main thread running on one processor whose function is preparing data then forking and initiating 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 3 tasks: the thread initiation and data preparation is the preprocessing task, computation is the slave task and postprocessing of the results from the computation is the postprocessing task. For parallel computing, we are usually interested in schedules that minimize the maximum finish time. But in many other applications in industrial settings, we may be interested in minimizing either the maximum finish time or the mean finish time; see [3, 4, 5].

Note that the master-slave model is closely related to the two-machine flowshop model with transfer lags; see [1, 6]. In this flowshop model, each job  $j$  has two operations: the first opera-

---

\*The work of this author was supported in part by FAA Grant 01-C-AW-NJIT and in part by NSF Grant DMI-0300156. Findings contained herein are not necessarily those of the FAA or NSF.

†The work of this author was supported by FAA Grant 01-C-AW-NJIT. Findings contained herein are not necessarily those of the FAA.

tion is scheduled on the upstream machine and the second operation is scheduled on the downstream machine. The interval or lag between the finish time of the first operation and the start time of the second operation must be exactly or at least  $l_j$ . If the  $l_j$ 's are large enough such that all the first operations finish before the start of any second operation, then the flowshop problem is equivalent to the problem of scheduling single machine with time lags and two tasks per job, subject to the constraint that all the first operations are scheduled first. On the other hand, the latter problem is identical to the single-master master-slave scheduling model.

### 1.1 Definitions and Notations

Given a schedule  $S$  of  $n$  jobs, two jobs  $i$  and  $j$  are said to *overlap* in  $S$  if the master machine is working on the preprocessing/postprocessing task of job  $i$  while a slave machine is working on the slave task of job  $j$ . Note that there may be several jobs overlapping with a given job  $i$ .

The completion (or finish) time of job  $i$  in a schedule  $S$  is the time when the task  $c_i$  finishes. We denote the completion time of  $i$  in  $S$  by  $C_i(S)$ . For simplicity, 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 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 *mean flowtime* of  $S$ , denoted by  $\sum C_j(S)$  or  $\sum C_j$ , is the sum of the completion times of all  $n$  jobs, that is  $\sum_{j=1}^n C_j$ . The problems of finding a schedule that minimizes the makespan and mean flowtime are referred to as the *makespan ( $C_{\max}$ ) problem* and *mean flowtime ( $\sum C_j$ ) problem*, respectively. A schedule that minimizes  $C_{\max}$  or  $\sum C_j$  is usually denoted by  $S^*$ . In this paper, we focus on the mean flowtime problem.

Sometimes we may have one or more constraints on the order of the jobs or tasks in the schedule. Thus we have minimization problems of makespan or mean flowtime subject to certain constraints.

Given  $n$  jobs in the single-master master-slave model, a schedule  $S$  of these jobs is *order pre-*

*servicing* if for any two jobs  $i$  and  $j$  such that  $a_i$  is scheduled before  $a_j$ ,  $c_i$  must also be scheduled before  $c_j$ . Conversely, if for any two jobs  $i$  and  $j$  such that  $a_i$  is scheduled before  $a_j$ ,  $c_i$  must be scheduled after  $c_j$ , then  $S$  is said to be *order reversing*.

A *no-wait-in* schedule is one such that each slave task must be scheduled immediately after the corresponding preprocessing task finishes and each postprocessing task must be scheduled immediately after the corresponding slave task finishes. In other words, once a job starts, it will not stop until it finishes.

A *canonical schedule* [4] is one that satisfies the following properties: (1) There are no preemptions; (2) The  $a$  tasks begin on the master machine at time 0 and complete at time  $\sum a_i$ ; (3) The  $b$  tasks begin as soon as their corresponding  $a$  tasks complete; (4) The  $c$  tasks are done in the same order as the  $b$  tasks complete and as soon as possible. If two  $b$  tasks complete at the same time, the  $c$  tasks are scheduled in the same order as the  $a$  tasks complete.

By definition, a canonical schedule is completely specified by the order of the preprocessing tasks. It is easy to see that one can always arrange any schedule to be canonical without increasing the makespan. Thus, for the problem of minimizing makespan, we only need to focus on canonical schedules. For the problem of minimizing mean flowtime, it is generally not true that the optimal schedule is canonical. However, in some applications, one may be only interested in canonical schedules. Thus the problem is to find a canonical schedule with the minimum possible mean flowtime in these cases.

Corresponding to various constraints, we have *order preserving (or reversing) mean flowtime problem*, *no-wait-in mean flowtime problem*, *canonical mean flowtime problem* and problems resulting from combinations of these constraints.

### 1.2 Previous Works and New Results

**Previous works.** So far the main efforts to the master-slave model are for the problem of makespan minimization. As noted before, it is enough to concentrate on canonical schedules for the makespan problem. The general makespan

problem without constraints has been shown to be NP-hard by Kern and Nawijn [2]. Sahni [3] showed that both the no-wait-in makespan problem and the order preserving no-wait-in makespan problem are NP-hard in the ordinary sense. He gave an  $O(n \log n)$  algorithm that solves the order preserving makespan problem, and an  $O(n \log n)$  algorithm to determine feasibility as well as a feasible schedule that minimizes the order reversing makespan with or without the constraint of no-wait.

Sahni and Vairaktarakis [4] studied some heuristics for the makespan problem in the single-master and multiple-master systems. For the general problem under the single-master systems, a heuristic that achieves the bound of  $\frac{3}{2}$  was given. For the general problem and the restricted reverse order scheduling problem under the multi-master systems, they gave heuristics that have a bound of 2 and  $2 - \frac{1}{m}$ , respectively, where  $m$  is the number of master machines.

Further heuristics were given by Vairaktarakis [5] when there are  $m_1$  preprocessors and  $m_2$  postprocessors. Let  $m = \max\{m_1, m_2\}$ . He developed heuristics with the bound of  $2 - \frac{1}{m}$  for the makespan problems with no constraint, or with the constraints of order preserving or order reversing.

The problem of minimizing makespan in a two-machine flowshop with delays and unit time operations is shown to be strongly NP-hard by Yu et al. [6]. By our discussion, this implies that minimizing the canonical makespan in the single-master master-slave model is also NP-hard in the strong sense, even if all the preprocessing and postprocessing tasks have unit time. And we know the canonical makespan problem is equivalent to the makespan problem.

**New results.** In this paper, we first develop some new results on the mean flowtime problem, based on the result from [6]. We showed that many problems are strongly NP-hard even if all the preprocessing and postprocessing tasks have unit time. We then prove that the problem of minimizing order preserving mean flowtime is strongly NP-hard. Finally, we give some heuristics for the mean flowtime problem. Our main results can be summarized as follows:

1. The mean flowtime problem is NP-hard in the strong sense even if preemption is allowed and all the preprocessing and postprocessing tasks have unit time.
2. The canonical, or no-wait-in, or canonical and no-wait-in mean flowtime problem is NP-hard in the strong sense even if the preprocessing and postprocessing tasks have unit time.
3. The order preserving and no-wait-in mean flowtime problem is NP-hard in the strong sense even if  $a_i = c_i$  for all  $1 \leq i \leq n$ .
4. The canonical order preserving mean flowtime problem can be solved in  $O(n \log n)$  time, when  $a_i = a$  and  $c_i = c$  for all  $1 \leq i \leq n$  and  $a \leq c$ .
5. Heuristics and analyses for canonical mean flowtime problem on single-master master-slave systems.

### 1.3 Organization of the Paper

The paper is organized as follows. In Section 2 we present the main complexity results. We show that many mean flowtime problems, with or without constraints, are NP-hard in the strong sense. In Section 3, we prove that if there are order preserving and canonical constraints, then in  $O(n \log n)$  time one can find an optimal schedule that minimizes the mean flowtime, assuming that  $a_i = a$  and  $c_i = c$  for all  $1 \leq i \leq n$  and  $a \leq c$ . After that, in Sections 4, we give some heuristics and analyses for the canonical mean flowtime problem. Finally, we draw some concluding remarks in Section 5.

## 2 Complexity Results

Yu et al. [6] showed that the problem  $F2 \mid l_j, p_{ij} = 1 \mid C_{\max}$  is strongly NP-hard. In fact, they [6] showed that the problem remains strongly NP-hard even with exact delays constraint (i.e., no-wait-in).

We can adapt the proof in [6] to show that the problem of minimizing mean flowtime with or

without constraints in the single-master master-slave model is strongly NP-hard. Because of space limitation, we will skip the proof of the following theorem.

**Theorem 1** *The problem of minimizing mean flowtime is strongly NP-hard, even if preemption is allowed and  $a_i = c_i = 1$  for  $1 \leq i \leq n$ . Furthermore, it remains strongly NP-hard even if we are restricted to canonical schedules, or no-wait-in schedules, or both canonical and no-wait-in schedules.*

As it turns out, the proof of the above theorem does not apply to order preserving scheduling problems. In the following, we will consider the complexity of mean flowtime problem with the order preserving constraint. We can show that the problem is NP-hard in the strong sense.

**Theorem 2** *The problem of minimizing the order preserving and no-wait-in mean flowtime is strongly NP-hard even if  $a_i = c_i$  for  $1 \leq i \leq n$ .*

**Proof :** Because of space limit, we only sketch the reduction here. We reduce from the 3-partition problem: given a set of non-negative integers  $X = \{x_1, x_2, \dots, x_{3m}\}$  and a non-negative integer  $B$  such that  $\sum_{i=1}^{3m} x_i = mB$  and  $B/4 < x_i < B/2$  for all  $1 \leq i \leq 3m$ . The problem is to decide whether  $X$  can be partitioned into  $m$  disjoint subsets  $X_1, \dots, X_m$  such that for all  $1 \leq j \leq m$ ,  $\sum_{x_i \in X_j} x_i = B$ ? In the following we call  $X_j$  a partition subset, where  $1 \leq j \leq m$ .

We create three types of jobs for the scheduling problem: (1)  $3m$  small jobs:  $a_i = c_i = x_i$  and  $b_i = 3B + 2 - x_i$ ,  $1 \leq i \leq 3m$ ; (2)  $2m$  medium jobs:  $a_i = c_i = B + 1$  and  $b_i = B$ ,  $3m + 1 \leq i \leq 5m$ ; (3)  $l$  large jobs:  $a_i = c_i = 3B + 2$  and  $b_i = \epsilon$ ,  $5m + 1 \leq i \leq 5m + l$ ,  $\epsilon$  is a small positive number and  $l$  is an integer greater than  $36m^2 + 19m + 24m^2/B + 12m/B$ .

Let

$$M = (3B + 2) \cdot m(2m + 1) ,$$

$$L = 2l m(3B + 2) + l(l + 1)(3B + 2) + \frac{l(l + 1)\epsilon}{2},$$

$$S = 3m \left[ 3mB + 2m + \frac{7}{4}B + 1 \right] .$$

Let  $B^* = S + M + L$ . Our scheduling problem is: Is there an order preserving and no-wait-in schedule of these jobs such that  $\sum_{j=1}^{5m+l} C_j \leq B^*$ ?

If the partition problem has a solution, then we can schedule the jobs as follows: first schedule the medium jobs in any order without overlapping; schedule any three small jobs that correspond to the three integers in the same partition subset fully overlapping with two adjacent medium jobs; finally, schedule the large jobs after the medium jobs one by one in any order without overlapping.

One can easily verify that the schedule is an order preserving and no-wait-in schedule. By simple calculation, we can see that the total completion time of the small jobs, medium jobs, large jobs are bounded by  $S$ ,  $M$ , and  $L$  respectively.

Now, suppose there is an order preserving and no-wait-in schedule of all these jobs such that  $\sum_{j=1}^{5m+l} C_j \leq B^*$ . We will show that there is a solution to the partition problem. Let  $S^*$  be such a schedule with the smallest mean flowtime. We have the following observations about  $S^*$ .

First, since  $a_j = c_j > b_i$  for any two jobs  $i$  and  $j$  that are both medium jobs or large jobs,  $i$  and  $j$  can not overlap with each other in  $S^*$ . By the same reason, a large job can not overlap with a medium job in  $S^*$ , nor can it overlap with a small job. Hence, overlapping can only occur between the small jobs or between the small and medium jobs. Next, because  $a_i = c_i = x_i > B/4$  for any small job  $i$ ,  $1 \leq i \leq 3m$ , there are at most three small preprocessing/postprocessing tasks that can overlap with a medium job  $j$ . Since  $2B < b_i < 3B + 2$  for any small job  $i$ , job  $i$  can overlap with at most two medium jobs in the schedule  $S^*$ .

Finally, one can show that (1) Large jobs must be scheduled after all medium jobs finish in  $S^*$  and (2) Exactly three small jobs overlap with a medium job  $j$  in  $S^*$ . Because  $b_j = B$ , the sum of the preprocessing or postprocessing tasks of the three small jobs overlapping with  $j$  is exactly  $B$ , which means that the corresponding three integers have a total exactly  $B$ . Thus, the partition problem has a solution.  $\square$

### 3 Optimal Canonical Order Preserving Schedules

While most of the mean flowtime scheduling problems are strongly NP-hard, there is a special case that admits a polynomial time solution. This is the case when  $a_i = a$  and  $c_i = c$  for  $1 \leq i \leq n$  and  $a \leq c$ , and we are restricted to canonical order preserving schedules. In this case, scheduling jobs in ascending order of the slave tasks' processing times yields an optimal schedule.

**Theorem 3** *The problem of minimizing the canonical order preserving mean flowtime can be solved in  $O(n \log n)$  time if  $a_i = a$  and  $c_i = c$  for  $1 \leq i \leq n$  and  $a \leq c$ .*

**Proof :** We will show that a canonical schedule  $S^*$  that schedules the preprocessing tasks in nondecreasing order of the processing times of the slave tasks gives an optimal order preserving schedule.

For any job  $i$ , let  $p_i$  denote the rank of job  $i$  in  $S^*$ ; i.e.,  $a_i$  is the  $p_i$ -th task scheduled in  $S^*$ . Because we consider canonical schedules, the earliest possible time to start  $c_i$  is  $r_i = \max(na, p_i a + b_i)$ . For any two jobs  $i$  and  $j$ , if  $p_i < p_j$  and  $b_i \leq b_j$ , then  $r_i \leq r_j$ . By the definition of canonical schedules,  $c_i$  will be scheduled before  $c_j$ . Thus,  $S^*$  is order preserving.

We will prove that  $S^*$  has the minimum mean flowtime among all canonical and order preserving schedules. Suppose there is another canonical and order preserving schedule  $S$  that is optimal. Suppose the jobs in  $S$  are in the order of  $1, 2, \dots, n$ , and there are two jobs  $i$  and  $i+1$  such that  $b_i > b_{i+1}$ . Let their finish times in  $S$  be  $C_i$  and  $C_{i+1}$ , respectively. By interchanging them, we have new finish times  $C'_i$  and  $C'_{i+1}$  and all the other jobs have the same finish times as before. We can show that  $C'_i \leq C_{i+1}$  and  $C'_{i+1} \leq C_i$ . Thus, the new schedule has mean flowtime no larger than before. By repeatedly interchanging jobs, we will arrive at the schedule  $S^*$ .  $\square$

Note that by Sahni [3], when  $a_i = a$ ,  $c_i = c$  and  $a = c$ , scheduling jobs in ascending order of the slave tasks' processing times also minimizes the makespan.

### 4 Heuristics for Canonical Mean Flowtime

In this section, we consider canonical schedules only, i.e., all the preprocessing tasks finish before any postprocessing task starts. Before we give our heuristic, we give an upper bound for the ratio of the worst schedule versus the best schedule. We assume that  $a_i = a$  and  $c_i = c$  for all  $1 \leq i \leq n$ . For a schedule  $S$ , we let  $C(S)$  denote  $\sum C_j(S)$ . We let  $C^*$  denote the optimal mean flowtime; i.e.,  $C^* = \sum C_j(S)$ .

**Theorem 4** *Let  $S$  be a schedule that schedules jobs in an arbitrary order. If  $a \leq c$ , then*

$$\frac{\sum_{i=1}^n C_i(S)}{\sum_{i=1}^n C_i(S^*)} < 1 + \frac{1}{1 + \frac{2a}{c}} . \quad (1)$$

*If  $a > c$ , then*

$$\frac{\sum_{i=1}^n C_i(S)}{\sum_{i=1}^n C_i(S^*)} < 1 + \frac{1}{2 + \frac{c}{a}} . \quad (2)$$

*If  $a$  is very small compared to  $c$ , the bound approaches 2. If  $a = c$ , the bound becomes  $\frac{4}{3}$ . If  $a$  is very large compared to  $c$ , the bound approaches  $\frac{3}{2}$ .*

Because of space limitation, we will not prove the above theorem. We now consider the heuristic where jobs are scheduled in ascending order of the slave tasks' processing times. The following two theorems give the performance of this heuristic.

**Theorem 5** *Suppose  $a \geq c$ . Let  $S$  be a schedule that schedules jobs in ascending order of the slave tasks' processing times. Then, we have*

$$\frac{C(S)}{C^*} < 1 + \frac{1}{4 + \frac{2c}{a}} . \quad (3)$$

*If  $a = c$ ,  $\frac{C(S)}{C^*} < \frac{7}{6}$ ; if  $a > c$ ,  $\frac{C(S)}{C^*} < \frac{5}{4}$ .*

**Proof :** Suppose  $b_1 \leq b_2 \leq \dots \leq b_n$ . Since  $S$  schedules the jobs in this order, for any  $1 \leq i \leq n$ ,  $r_i = \max(na, ia + b_i)$ . Therefore,  $r_i \leq r_{i+1}$ , which implies that  $C_i < C_{i+1}$ . Let  $j$  be the last job that finishes before the first idle time. We divide the jobs into two subsets  $G_1$  and  $G_2$ . Let  $j$  and all the jobs that finish before  $j$  be in

$G_1$  and let the remaining jobs be in  $G_2$ . Let  $C_{G_i}^*$ ,  $i = 1, 2$  be the minimum total completion time of jobs in  $G_i$  among all schedules of these  $n$  jobs. By the above analysis,  $k \in G_1$  if and only if  $k \leq j$ . Because there is no idle time before  $j$  finishes,  $C_k(S) = na + kc$  for all  $k \leq j$ . Thus,  $\sum_{k \in G_1} C_k(S) = C_{G_1}^*$ .

Since there is idle time before  $c_{j+1}$  starts, we have  $C_{j+1} = r_{j+1} + c = (j+1)a + b_{j+1} + c$ . By assumption,  $a \geq c$  and  $b_{j+1} \leq b_{j+2}$ . Therefore,  $C_{j+1} \leq (j+2)a + b_{j+2} = r_{j+2}$ . Thus,  $C_{j+2} = \max(C_{j+1}, r_{j+2}) + c = r_{j+2} + c$ . Similarly, for any  $j+1 \leq k \leq n$ , we have  $C_k = r_k + c = ka + b_k + c$ . Thus,

$$\begin{aligned} & \sum_{k=1}^n C_k(S) \\ &= \sum_{k \in G_1} C_k(S) + \sum_{k \in G_2} C_k(S) \\ &= C_{G_1}^* + \sum_{k=j+1}^n (ka + b_k + c) \\ &= C_{G_1}^* + \left[ \sum_{k=1}^{n-j} (ka + b_{k+j} + c) \right] + j(n-j)a \\ &\leq C_{G_1}^* + C_{G_2}^* + j(n-j)a \\ &\leq \sum_{k=1}^n C_k(S^*) + \frac{1}{4}n^2a. \end{aligned}$$

Since we consider canonical schedules only, the earliest possible time to schedule a postprocessing task is  $na$ . Hence, the best possible schedule is one that first executes a postprocessing task at  $na$ , and thereafter the remaining postprocessing tasks, one after another without any idle time. Thus,  $\sum_{k=1}^n C_k(S^*) \geq n^2a + \frac{1}{2}n(n+1)c$ .

Therefore, we have

$$\begin{aligned} \frac{\sum_{k=1}^n C_k(S)}{\sum_{k=1}^n C_k(S^*)} &\leq 1 + \frac{\frac{1}{4}n^2a}{n^2a + \frac{1}{2}n(n+1)c} \\ &< 1 + \frac{1}{4 + \frac{2c}{a}}. \end{aligned}$$

It is easy to see that if  $a = c$ ,  $\frac{\sum_{k=1}^n C_k(S)}{\sum_{k=1}^n C_k(S^*)} < \frac{7}{6}$ , and if  $a > c$ ,  $\frac{\sum_{k=1}^n C_k(S)}{\sum_{k=1}^n C_k(S^*)} < \frac{5}{4}$ .  $\square$

To get examples having tight bound for (3) when  $a = c$ , set half of the jobs with  $b_i = (\frac{1}{2}n -$

1) $a$  and the other half with  $b_i = (\frac{3}{2}n - 1)a$ . For example, if  $n = 6$  and  $a = c = 1$ , let the  $b_i$ 's be 2, 2, 2, 8, 8, 8. The optimal schedule will schedule the jobs with  $b_i = 8$  first, then schedule the jobs with  $b_i = 2$ .

If  $a > c$ , let  $b_1 = c$ ,  $b_2 = a$  and  $b_i = (i-1)(a+c)$  for  $3 \leq i \leq n$ . Then the optimal schedule schedules jobs in descending order of  $b_i$  and the total completion time is  $n^2a + \frac{1}{2}n(n+1)c$ . However, scheduling jobs in ascending order of  $b_i$ 's gives the completion time  $\frac{5}{4}n^2a + \frac{1}{2}n(n+1)c$ .

Next we want to bound the performance ratio when we have  $a < c$  and jobs are scheduled in ascending order of the slave tasks' processing times.

**Theorem 6** *Given  $n$  jobs with preprocessing tasks  $a$  and postprocessing tasks  $c$  and  $a < c$ , scheduling jobs in ascending order of the slave tasks' processing times gives mean flowtime at most  $\frac{4}{3}$  times the optimal; if  $4a \leq c$ , then the bound is  $\frac{7}{6}$ .*

**Proof :** Suppose that  $b_1 \leq b_2 \leq \dots \leq b_n$ . Let  $S$  be the schedule that schedules jobs in this order. We divide the jobs into groups according to their completion times. Those jobs that complete before the first idle interval are in  $G_0$ . Those jobs that complete after the first interval and before the second idle interval are in  $G_1$ . Those jobs that complete after the second interval and before the third idle interval are in  $G_2$ , and so on. Suppose there are  $m+1$  groups. Let the numbers of jobs in these groups be  $x_0, \dots, x_m$ , respectively.

Let  $S_{G_i}^*$  be the schedule that minimizes  $\sum_{k \in G_i} C_k$  among all schedules of these  $n$  jobs. Let  $C_{G_i}^* = \sum_{k \in G_i} C_k(S_{G_i}^*)$ . Since the jobs in  $G_0$  complete one by one immediately after all  $a$  tasks finish under  $S$ , we have  $\sum_{k \in G_0} C_k(S) = C_{G_0}^*$ . For  $G_i$ ,  $1 \leq i \leq m$ , let  $y_i = \sum_{j=0}^{i-1} x_j$ , i.e.,  $y_i$  is the number of jobs scheduled before jobs of  $G_i$ . We shall show that  $\sum_{j \in G_i} C_j(S) \leq C_{G_i}^* + x_i \cdot y_i \cdot a$ .

Suppose that the jobs finish in the order  $j_1, j_2, \dots, j_{x_i}$  under  $S_{G_i}^*$ . Then  $C_{j_1}(S_{G_i}^*) \geq a + b_{j_1} + c$  and  $C_{j_k}(S_{G_i}^*) \geq C_{j_1}(S_{G_i}^*) + (k-1)c$ ,  $1 \leq k \leq x_i$ . So

$$C_{G_i}^* = \sum_{k=1}^{x_i} C_{j_k}(S_{G_i}^*) \geq \sum_{k=1}^{x_i} [C_{j_1}(S_{G_i}^*) + (k-1)c].$$

and if  $4a \leq c$ , we have  $\frac{\sum_{k=1}^n C_k(S)}{\sum_{k=1}^n C_k(S^*)} < \frac{7}{6}$ .  $\square$

It can be easily shown that  $S$  is order preserving. So job  $y_i + 1$  must finish first among all jobs of  $G_i$ . Because there is idle time before  $c_{y_i+1}$ , we must have  $C_{y_i+1}(S) = (y_i + 1)a + b_{y_i+1} + c$ . By assumption, we know  $b_{y_i+1} \leq b_{j_1}$ . Therefore,  $C_{y_i+1}(S) \leq y_i a + (a + b_{j_1} + c) \leq y_i a + C_{j_1}(S_{G_i}^*)$ . Since there is no idle time between jobs of  $G_i$  under  $S$ , for any other job  $j \in G_i$ , we have  $C_j(S) = C_{y_i+1}(S) + (j - y_i - 1)c$ . Thus,

$$\begin{aligned} \sum_{j \in G_i} C_j(S) &= \sum_{j=y_i+1}^{y_i+x_i} [C_{y_i+1}(S) + (j - y_i - 1)c] \\ &\leq \sum_{j=y_i+1}^{y_i+x_i} [y_i a + C_{j_1}(S_{G_i}^*) + (j - y_i - 1)c] \\ &= x_i \cdot y_i a + \sum_{k=1}^{x_i} [C_{j_1}(S_{G_i}^*) + (k - 1)c] \\ &= x_i \cdot y_i a + C_{G_i}^* . \end{aligned}$$

Therefore, we have

$$\begin{aligned} \sum_{k=1}^n C_k(S) &= \sum_{k \in G_0} C_k(S) + \sum_{i=1}^m \sum_{k \in G_i} C_k(S) \\ &< C_{G_0}^* + \sum_{i=1}^m (C_{G_i}^* + x_i \cdot y_i a) \\ &= \left( \sum_{i=0}^m C_{G_i}^* \right) + \left( \sum_{i=1}^m \sum_{j=0}^{i-1} x_i x_j a \right) \\ &\leq \sum_{k=1}^n C_k(S^*) + \left( \sum_{i=1}^m \sum_{j=0}^{i-1} x_i x_j a \right) \\ &\leq \sum_{k=1}^n C_k(S^*) + \frac{1}{2} \cdot \left( \sum_{i=0}^m x_i \right)^2 a \\ &= \sum_{k=1}^n C_k(S^*) + \frac{1}{2} n^2 a . \end{aligned}$$

Thus,

$$\begin{aligned} \frac{\sum_{k=1}^n C_k(S)}{\sum_{k=1}^n C_k(S^*)} &\leq 1 + \frac{\frac{1}{2} n^2 a}{\sum_{k=1}^n C_k(S^*)} \\ &\leq 1 + \frac{\frac{1}{2} n^2 a}{n^2 a + \frac{1}{2} n(n+1)c} \\ &< 1 + \frac{1}{2 + \frac{c}{a}} < \frac{4}{3} , \end{aligned}$$

## 5 Conclusion

In this paper, we have studied the problem of minimizing mean flowtime in the single-master master-slave model. We proved that many mean flowtime problems with certain constraints, such as canonical, order preserving and no-wait-in, are NP-hard in the strong sense. For the special case where  $a_i = a$  and  $c_i = c$  for all  $1 \leq i \leq n$  and  $a \leq c$ , we give an  $O(n \log n)$ -time optimal algorithm for canonical order preserving schedules. For the special case where  $a_i = a$  and  $c_i = c$ , we developed heuristics that have good performance bounds for canonical mean flowtime. The complexity of minimizing the order preserving mean flowtime is still open. Other future directions include designing heuristics for other cases that were not studied in this paper, as well as heuristics for arbitrary processing times.

## References

- [1] M. Dell'Amica. Shop problems with two machine and time lags. *Operations Research*, 44, 5, 777-787, 1996.
- [2] W. Kern and W. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. University of Twente, 1993.
- [3] S. Sahni. Scheduling master-slave multiprocessor systems. *IEEE Trans. on Computers*, 45, 10, 1195-1199, 1996.
- [4] S. Sahni and G. Vairaktarakis. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9, 357-377, 1996.
- [5] G. Vairaktarakis. Analysis of algorithms for master-slave system. *IIE Transactions*, 29, 11, 939-949, 1997.
- [6] W. Yu, H. Hoogeveen and J.K. Lenstra. Minimizing makespan in a two-machine flowshop with delays and unit-time operations is NP-hard. To appear in *Journal of Scheduling*.