

Minimizing Mean Flowtime and Makespan on Master-Slave Systems

Joseph Y-T. Leung^{*,1} and Hairong Zhao²

*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, etc. In this model each job is associated with 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. In this paper we consider scheduling problems on single-master master-slave systems. We first strengthen some previously known complexity results for makespan problems, by showing them to be strongly NP-hard. We then show that the problem of minimizing the mean flowtime is strongly NP-hard even under severe constraints. Finally, we propose some heuristics for the mean flowtime and makespan problems subject to some constraints, and we analyze the worst-case performance of these heuristics.

Key words: Master-slave systems, Scheduling, Makespan, Mean flowtime, NP-hard, Heuristics

* Corresponding author.

Email addresses: leung@njit.edu (Joseph Y-T. Leung), hz2@njit.edu (Hairong Zhao).

¹ The work of this author was supported in part by the FAA Grant 01-C-AW-NJIT and in part by the NSF Grant DMI-0300156. Findings contained herein are not necessarily those of the FAA or NSF.

² The work of this author was supported by the FAA Grant 01-C-AW-NJIT. Findings contained herein are not necessarily those of the FAA.

1 Introduction

We consider scheduling problems in the master-slave model which was recently introduced by Sahni [4]. With this model we are given a set of n jobs and a set of m machines. Each job is associated with 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 and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc.; see [4], [5], [6], [7]. For example, suppose there is a main thread running on one processor 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. Maximum finish time and mean finish time are the two common objectives to minimize.

The master-slave model is closely related to the two-machine flowshop model with transfer lags. In this flowshop model, each job j has two operations: the first operation is scheduled on the upstream machine and the second operation

is scheduled on the downstream machine. The interval or time 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 of the first operations finish before the start of any second operation, then the flow-shop problem is equivalent to the problem of scheduling on a single machine with time lags and two tasks per job, subject to the constraint that all of the first operations are scheduled first. 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 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 *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, i.e., $\sum_{j=1}^n C_j$. 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^* .

Sometimes we may have one or more constraints put 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 preserving* 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 post-processing 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 [5] is one that satisfies the following properties: (1) There are no preemptions; (2) The preprocessing tasks begin on the master machine at time 0 and complete at time $\sum a_i$; (3) The slave tasks begin as soon as their corresponding preprocessing tasks complete; (4) The postprocessing tasks are done in the same order as the slave tasks complete and as soon as possible. If two slave tasks complete at the same time, the postprocessing tasks are scheduled in the same order as the preprocessing tasks.

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 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. For the problem of minimizing the mean flowtime, it is generally not true that the optimal schedule is canonical. However, in some applications, one may be interested only in canonical schedules. Thus the problem is to find a canonical schedule with the minimum mean flowtime among all canonical schedules.

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

any combinations of these constraints.

1.2 Previous works and new results

Previous works. So far the main research efforts to the master-slave model are for makespan minimization. As noted before, it is sufficient to focus on canonical schedules for the makespan objective. The general makespan problem without constraints has been shown to be NP-hard by Kern and Nawijn [3]. Sahni [4] showed that the no-wait-in makespan problem is NP-hard in the ordinary sense, even when there is order preserving constraint. He gave an $O(n \log n)$ -time algorithm that solves the order preserving makespan problem, and an $O(n \log n)$ -time algorithm to determine feasibility as well as a feasible schedule that minimizes the order reversing makespan, with or without the constraint of no-wait-in.

Sahni and Vairaktarakis [5] proposed several heuristics for the makespan problem in the single-master and multiple-master systems. For the general problem under the single-master systems, a heuristic with a worst-case 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 with worst-case bounds of 2 and $2 - \frac{1}{m}$, respectively, where m is the number of master machines.

Further heuristics were given by Vairaktarakis [7] when there are m_1 preprocessors and m_2 postprocessors. Let $m = \max\{m_1, m_2\}$. He gave heuristics with a worst-case 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 the makespan in a two-machine flowshop with delays was shown to be strongly NP-hard by Dell'Amico [1], for both preemptive and nonpreemptive scheduling. Yu et al. [8] strengthened the result to unit

time operations. By our discussion, this means 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 that the canonical makespan problem is equivalent to the makespan problem.

New results. In this paper we first strengthen some previous complexity results for the makespan problem and develop some new results for the mean flowtime problem, based on the result from [8]. 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 and no-wait-in makespan or mean flowtime is strongly NP-hard. Finally, we give some heuristics for the mean flowtime and makespan problems. Our main results can be summarized as follows:

- 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.
- 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.
- 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 $1 \leq i \leq n$.
- The order preserving and no-wait-in makespan problem is NP-hard in the strong sense, even if $a_i = c_i$ for all $1 \leq i \leq n$.
- The canonical and 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 $a \leq c$.
- Heuristics and analyses for the canonical mean flowtime problem in the case $a_i = a$ and $c_i = c$ for all $1 \leq i \leq n$.
- Heuristics and analyses for the no-wait-in makespan problem when $a_i = a$

and $c_i = c$ for all $1 \leq i \leq n$.

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 makespan and mean flowtime problems, with or without constraints, are NP-hard in the strong sense. In Section 3, we prove that if there are canonical and order preserving constraints, then in $O(n \log n)$ time one can find an optimal schedule that minimizes the mean flowtime, when $a_i = a$ and $c_i = c$ for all $1 \leq i \leq n$ and $a \leq c$. After that, in Sections 4 and 5, we give some heuristics and analyses for the canonical mean flowtime problem and the no-wait-in makespan problem, respectively. Finally, we draw some concluding remarks in Section 6.

2 Complexity results

We begin with some recent results about the C_{\max} problem in a two-machine flowshop with delays. Yu et al. [8] proved the following theorem.

Theorem 1 (See Theorem 21 and Corollary 22 [8]) *The flowshop problem $F2|l_j, p_{ij} = 1|C_{\max}$ is strongly NP-hard, even if we have exact delays constraint.*

By our discussion in Section 1, we immediately have the following theorem.

Theorem 2 *The makespan problem with $a_i = c_i = 1$ for all $1 \leq i \leq n$ is strongly NP-hard, even if we have no-wait-in constraint.*

Proof : We give an outline of the proof of this theorem, as it is relevant to our discussion of the mean flowtime problem later. We use almost the same reduction

as in the proof of Theorem 21 in [8], but for each job i , instead of having a lag l_i between its two tasks, it now has a slave task b_i which must start after the preprocessing task and finish before the postprocessing task. The preprocessing and postprocessing tasks are performed on the same master machine, instead of two machines. To ensure the same argument goes through, we let $b_i = l_i + Y$ for each job i , where $Y \geq n$ and n is the number of jobs in the instance of the two-machine flowshop with delays. We also increase the bound for the makespan by Y . The proof of Theorem 21 in [8] used a reduction from the 3-partition problem, which is known to be strongly NP-complete; see [2].

A 3-partition problem instance has *input* as 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 $1 \leq j \leq m$.

An instance of the makespan problem can be constructed as follows. There are $n = m^2B + mB$ jobs. For job i , $1 \leq i \leq 3m$, referred to as a P -job in [8], $b_i = x_i + Y$ where $Y \geq n$; for job i , $3m+1 \leq i \leq mB$, referred to as a Z -job in [8], $b_i = Y$, and for job i , $mB+1 \leq i \leq m^2B + mB$, referred to as a L -job in [8], $b_i = (m+1)B + 1 + Y$. For all job i , $1 \leq i \leq n$, let $a_i = c_i = 1$. Let the bound for the makespan be $mB + n + 2 + Y$.

Using the same argument as in [8], we can show that if there is a canonical schedule S with makespan less than or equal to $mB + n + 2 + Y$, then S must have the following properties: (1) The makespan of S is exactly $(mB + n + 2 + Y)$, (2) S is a no-wait-in-schedule and the finish times of the jobs are $(mB + 3 + Y)$, $(mB + 4 + Y), \dots$, $(mB + n + 2 + Y)$. See Figure 1 for an illustration of the schedule on the master machine. Also, we can show that there is a solution to the 3-partition problem if and only if there is a schedule S with makespan exactly $mB + n + 2 + Y$. \square

We will use the above result to show that the mean flowtime problem is also strongly NP-hard.

Theorem 3 *The mean flowtime problem with $a_i = c_i = 1$ for all $1 \leq i \leq n$ is strongly NP-hard, even if preemption is allowed, or if we are restricted to canonical or no-wait-in or both canonical and no-wait-in schedules.*

Proof : Let S^* be an optimal schedule for any n jobs with respect to the mean flowtime objective. Let C^* denote the mean flowtime of S^* . For any job i , $1 \leq i \leq n$, we use p_i to denote the time when the task a_i finishes in the schedule S^* . Then, $p_i \geq 1$, and if $i \neq j$ then $p_i \neq p_j$. Thus, we have $\sum_{i=1}^n p_i \geq (1+2+\dots+n) = \frac{1}{2}n(n+1)$. Let the completion time of job i in S^* be C_i . Then we have $C_i \geq p_i + b_i + c_i = p_i + b_i + 1$. Therefore,

$$C^* = \sum_{i=1}^n C_i \geq \sum_{i=1}^n (p_i + b_i + 1) = \sum_{i=1}^n p_i + \sum_{i=1}^n b_i + n \geq \frac{1}{2}n(n+1) + \sum_{i=1}^n b_i + n .$$

Let us examine the constructed instance in the proof of Theorem 2. If there is a schedule S with makespan less than or equal to $mB + n + 2 + Y$, then S must be canonical. Since S is canonical, $\sum_{i=1}^n p_i(S) = (1 + 2 + \dots + n) = \frac{1}{2}n(n+1)$, where $p_i(S)$ is the time when a_i finishes in S . At the same time, S must also be a no-wait-in schedule which means that $C_i(S) = p_i(S) + b_i + 1$. Hence, the mean flowtime of S is $C(S) = \sum_{i=1}^n (p_i(S) + b_i + 1) = \frac{1}{2}n(n+1) + \sum_{i=1}^n b_i + n \leq C^*$. Thus, there is a schedule with makespan less than or equal to $mB + n + 2 + Y$ if and only if there is a no-wait-in schedule with mean flowtime less than or equal to C^* . It is clear that preemption does not help in this case. \square

Observe that the optimal schedule S for the constructed instance in the proof of Theorem 2 is not order preserving, so all of the above results are not applicable to order preserving scheduling problems. In the following we will consider the complexity of no-wait-in makespan or mean flowtime problem with the constraint of order preserving. We show that both problems are NP-hard in

the strong sense by a reduction from the 3-partition problem.

Theorem 4 *The problem of minimizing the order preserving and no-wait-in makespan is strongly NP-hard, even if $a_i = c_i$ for $1 \leq i \leq n$.*

Proof : We create two types of jobs for the instance of the scheduling problem. There are (1) $3m$ *partition* jobs: $a_i = c_i = x_i$ and $b_i = 3B + 2 - x_i$, $1 \leq i \leq 3m$, and (2) $2m$ *separation* jobs: $a_i = c_i = B + 1$ and $b_i = B$, $3m + 1 \leq i \leq 5m$. The problem is to determine whether there is an order preserving and no-wait-in-schedule such that $C_{\max} \leq 2m(3B + 2)$. Clearly, the reduction can be done in polynomial time.

If there is a solution to the 3-partition problem, then we can schedule the separation jobs in any order without overlapping, and for each group of 3-partition jobs corresponding to a partition subset, schedule their preprocessing tasks fully overlapping with one separation job and the postprocessing tasks fully overlapping with the separation job immediately following the previous one. See Figure 2 for an illustration of the schedule. It is clear that the schedule is order preserving and no-wait-in and $C_{\max} = 2m(3B + 2)$.

Now suppose the scheduling problem has a solution; i.e., there is an order preserving and no-wait-in schedule S such that $C_{\max} \leq 2m(3B + 2)$. Since we do not allow waiting and since $a_i = c_i > b_j$ for any two separation jobs i and j , a separation job can not overlap with another one. Hence, $C_{\max} \geq \sum_{j=3m+1}^{5m} (a_j + b_j + c_j) = 2m(3B + 2)$. By assumption, $C_{\max} \leq 2m(3B + 2)$. Therefore, we must have $C_{\max} = 2m(3B + 2)$, which means that all the partition jobs must fully overlap with the separation jobs. For each partition job i , $2B < b_i < 3B + 2$. Therefore, each partition job i must overlap with exactly two adjacent separation jobs in the schedule S . Because $B/4 < a_i = c_i < B/2$, we can have at most three preprocessing and/or postprocessing tasks of the partition jobs overlapping with one separation job. Since we only have $2m$ separation jobs, there must be exactly three preprocessing or postprocessing tasks fully overlapping with each separation job. For each separation

job j , we have $b_j = B$. Thus, the integers corresponding to the three preprocessing or postprocessing tasks that overlap with b_j have a total exactly B . Hence, the 3-partition problem has a solution. \square

Theorem 5 *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 : 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 $b_i = \epsilon$, $5m + 1 \leq i \leq 5m + l$, ϵ is a small positive number and l is an integer greater than $36m^2 + 19m + 24m^2/B + 12m/B$.

Let

$$\begin{aligned} M &= (3B + 2) \cdot m(2m + 1) , \\ L &= 2l m(3B + 2) + l(l + 1)(3B + 2) + \frac{1}{2}l(l + 1)\epsilon, \text{ and} \\ S &= 3m \left[3mB + 2m + \frac{7}{4}B + 1 \right] . \end{aligned}$$

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. See Figure 3 for an illustration of the schedule.

One can easily verify that the schedule is an order preserving and no-wait-in schedule. To bound the mean flowtime, we calculate the total completion time of each type of jobs separately. Without loss of generality, suppose that the medium jobs are scheduled in the order of $3m + 1, 3m + 2, \dots, 5m$. Since the medium jobs are

scheduled one by one from time 0 without overlap, the total completion time of all medium jobs is

$$\sum_{i=3m+1}^{5m} (i - 3m) \cdot (a_i + b_i + c_i) = \sum_{i=1}^{2m} i \cdot (3B + 2) = (3B + 2) \cdot m(2m + 1) = M .$$

Similarly, the total completion time of all large jobs is

$$\begin{aligned} & \sum_{i=5m+1}^{5m+l} (2m(3B + 2) + (i - 5m) \cdot (a_i + b_i + c_i)) \\ &= l \cdot 2m(3B + 2) + \sum_{i=1}^l i(2(3B + 2) + \epsilon) \\ &= 2lm(3B + 2) + l(l + 1)(3B + 2) + \frac{1}{2}l(l + 1)\epsilon \\ &= L . \end{aligned}$$

Now, we consider the total completion time of the small jobs. Suppose that the small jobs corresponding to the partition subset X_j , $1 \leq j \leq m$, are j_1 , j_2 and j_3 ; furthermore, suppose that j_1 is scheduled before j_2 which is scheduled before j_3 . We also suppose that these jobs overlap with two consecutive medium jobs $3m + 2j - 1$ and $3m + 2j$. Let C_{j_i} denote the completion time of the small job j_i . Then, we have

$$\begin{aligned} C_{j_1} &= [2(j - 1)(3B + 2) + (B + 1)] + a_{j_1} + b_{j_1} + c_{j_1} \\ &= [2(j - 1)(3B + 2) + (B + 1)] + x_{j_1} + (3B + 2 - x_{j_1}) + x_{j_1} \\ &= [(2j - 1)(3B + 2) + (B + 1)] + x_{j_1} \\ &< [(2j - 1)(3B + 2) + (B + 1)] + \frac{1}{2}B . \end{aligned}$$

Similarly, we can get

$$\begin{aligned} C_{j_2} &= [2(j - 1)(3B + 2) + (B + 1) + a_{j_1}] + a_{j_2} + b_{j_2} + c_{j_2} \\ &= [2(j - 1)(3B + 2) + (B + 1) + x_{j_1}] + x_{j_2} + (3B + 2 - x_{j_2}) + x_{j_2} \\ &= [(2j - 1)(3B + 2) + (B + 1)] + x_{j_1} + x_{j_2} \\ &< [(2j - 1)(3B + 2) + (B + 1)] + \frac{3}{4}B \end{aligned}$$

and

$$\begin{aligned}
C_{j_3} &= [2(j-1)(3B+2) + (B+1) + a_{j_1} + a_{j_2}] + a_{j_3} + b_{j_3} + c_{j_3} \\
&= [2(j-1)(3B+2) + (B+1) + x_{j_1} + x_{j_2}] + x_{j_3} + (3B+2 - x_{j_3}) + x_{j_3} \\
&= [(2j-1)(3B+2) + (B+1)] + x_{j_1} + x_{j_2} + x_{j_3} \\
&= [(2j-1)(3B+2) + (B+1)] + B .
\end{aligned}$$

Thus,

$$\begin{aligned}
C_{j_1} + C_{j_2} + C_{j_3} &< 3 \cdot [(2j-1)(3B+2) + (B+1)] + \frac{1}{2}B + \frac{3}{4}B + B \\
&= 3 \cdot [(2j-1)(3B+2) + (B+1)] + \frac{9}{4}B .
\end{aligned}$$

Hence, the total completion time of all small jobs is

$$\begin{aligned}
\sum_{j=1}^{3m} C_j &= \sum_{j=1}^m \sum_{i=1}^3 C_{j_i} < \sum_{j=1}^m \left[3 \cdot ((2j-1)(3B+2) + (B+1)) + \frac{9}{4}B \right] \\
&= 3m^2 \cdot (3B+2) + 3m(B+1) + \frac{9}{4}mB \\
&= 3m \left[3mB + 2m + \frac{7}{4}B + 1 \right] \\
&= S .
\end{aligned}$$

Therefore, the total completion time of all jobs is

$$\sum_{j=1}^{3m} C_j + \sum_{j=3m+1}^{5m} C_j + \sum_{j=5m+1}^{5m+l} C_j < S + M + L = B^* .$$

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 C^* . We have the following observations about S^* .

First, $a_j = c_j > b_i$ for any two jobs i and j that are both medium jobs or large jobs. Therefore, i and j can not overlap 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 must overlap with exactly two adjacent medium jobs in the schedule S^* .

Finally, we give two other properties of S^* .

- Large jobs are scheduled after all medium jobs finish in S^* .

As we have shown, the large jobs can not overlap with the medium jobs. If a large job is scheduled before a medium job, then we can interchange them to obtain a schedule with a smaller mean flowtime.

- Exactly three small jobs overlap with a medium job in S^* .

It is clear that a small job can not be scheduled after a large job; otherwise, we can interchange them without increasing the mean flowtime. Similarly, a small job can not be scheduled between two medium jobs. Therefore, a small job can only be scheduled either before all medium and large jobs or fully overlapping with medium jobs.

Suppose there is a preprocessing task a_i of a small job i that is scheduled before any medium or large jobs in S^* . Then, we have

$$\begin{aligned}
C^* &= \sum_{j=1}^{5m+l} C_j > \sum_{j=3m+1}^{5m+l} C_j \\
&\geq \left[\sum_{j=3m+1}^{5m} (a_i + (j - 3m) \cdot (a_j + b_j + c_j)) \right] \\
&\quad + \left[\sum_{j=5m+1}^{5m+l} (a_i + 2m(3B + 2) + (j - 5m) \cdot (a_j + b_j + c_j)) \right] \\
&= \left[\sum_{j=1}^{2m} (a_i + j \cdot (3B + 2)) \right] + \left[l(a_i + 2m(3B + 2)) + \sum_{j=1}^l j \cdot (2(3B + 2) + \epsilon) \right]
\end{aligned}$$

$$\begin{aligned}
&= (2m + l)a_i + \left[\sum_{j=1}^{2m} j \cdot (3B + 2) \right] + \left[l2m(3B + 2) + \sum_{j=1}^l j \cdot (2(3B + 2) + \epsilon) \right] \\
&= (2m + l)a_i + M + L \\
&> \frac{1}{4}(2m + l)B + B^* - S .
\end{aligned}$$

By assumption, $l > 36m^2 + 19m + 24m^2/B + 12m/B$. Thus, $S < \frac{1}{4}(2m + l)B$. Hence, $C^* > B^*$, contradicting our assumption that $C^* \leq B^*$.

Therefore, every small job must overlap with exactly two adjacent medium jobs in S^* . Since there are $2m$ medium jobs and $3m$ small jobs, there must be exactly three preprocessing tasks or three postprocessing tasks overlapping with a medium job.

For any medium job j , there are exactly three small jobs overlapping with it 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 and order preserving schedules

The previous section shows that minimizing mean flowtime is strongly NP-hard, even under severe constraints. In this section we show that there is a special case that admits a polynomial time algorithm.

Theorem 6 *The problem of minimizing the canonical and 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 Sahni [4] showed that when $a_i = a$, $c_i = c$ and $a = c$, scheduling jobs in nondecreasing order of the processing times of the slave tasks also minimizes the makespan.

If $a > c$, then the canonical schedule that schedules the jobs in nondecreasing order of the processing times of the slave tasks is still order preserving but may not be optimal. However, it is a $\frac{5}{4}$ -approximation algorithm, as we shall see in the next section.

4 Heuristics for canonical mean flowtime

In this section we consider heuristics for the mean flowtime problem when $a_i = a$ and $c_i = c$ for all i . We will consider canonical schedules only. In this section, when we mention a schedule, we mean a canonical schedule.

Suppose we are given n jobs each of which has preprocessing time a and postprocessing time c . Each job i has a slave task b_i . Let S be a canonical schedule of these n jobs. We use $p_i(S)$ to denote the rank of job i in S ; i.e., a_i is the $p_i(S)$ -th preprocessing task that is processed. Let $r_i(S) = \max(na, p_i(S) \cdot a + b_i)$ be the ready time of c_i ; i.e., at time $r_i(S)$ the master machine will schedule c_i if it finishes the postprocessing tasks that are ready earlier than c_i . Let S^* be the canonical schedule of these n jobs with minimum mean flowtime C^* .

Given a subset G of the n jobs, we use $C_G(S)$ or C_G to denote the total completion time of the jobs in G in S . We will also consider the schedule that minimizes C_G among all schedules of the n jobs. We denote such a schedule by S_G^* , and the corresponding total completion time of the jobs in G by C_G^* .

The next lemma gives a lower bound of C^* . This lemma is useful for later analysis.

Lemma 7

$$C^* \geq n^2 a + \frac{1}{2} n(n+1)c \quad (1)$$

$$C^* \geq \frac{1}{2} n(n+1)a + \sum_{i=1}^n b_i + n c \quad (2)$$

Proof : We will prove (1) first. 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,

$$C^* \geq \sum_{i=1}^n (na + ic) = n^2 a + \frac{1}{2} n(n+1)c .$$

For (2), recall that a postprocessing task can not start earlier than its ready time

$r_i(S)$. Therefore,

$$\begin{aligned}
C^* &\geq \sum_{i=1}^n (r_i(S^*) + c) \\
&\geq \sum_{i=1}^n (p_i(S^*) a + b_i + c) \\
&= \left(\sum_{i=1}^n p_i(S^*) a \right) + \sum_{i=1}^n b_i + n c \\
&= \frac{1}{2} n(n+1)a + \sum_{i=1}^n b_i + n c .
\end{aligned}$$

□

The next two lemmas show that there are some special cases that can be solved by a polynomial time algorithm.

Lemma 8 *If $\max_{1 \leq i \leq n} b_i \leq (n-1) \cdot \min(a, c)$, then any canonical schedule is an optimal schedule.*

Proof : Let S be a canonical schedule. Suppose the jobs are scheduled in the order of j_1, j_2, \dots, j_n . By definition, $p_{j_i}(S) = i$. By assumption, $b_{j_i} \leq (n-1)a$. Thus, $r_{j_1}(S) = \max(na, a + b_{j_1}) = na$. So,

$$C_{j_1} = r_{j_1}(S) + c = na + c .$$

For job j_2 , we have

$$r_{j_2}(S) = \max(na, 2a + b_{j_2}) \leq \max(na, (2a + (n-1) \min(a, c))) \leq na + c = C_{j_1} .$$

Therefore, c_{j_2} will start immediately after c_{j_1} completes at $na + c$. Similarly, we can prove that there is no idle time before all c tasks finish. Thus, $C(S) = \sum_{i=1}^n (na + ic) = n^2 a + \frac{1}{2} n(n+1)c$. By (1), S is an optimal schedule. □

Lemma 9 *Suppose $(n-1)a \leq b_1 \leq b_2 \leq \dots \leq b_n$. If $a \geq c$, or $a < c$ but $b_{i+1} - b_i \geq c - a$ for $1 \leq i \leq n-1$, then an optimal schedule can be obtained*

by scheduling jobs in nondecreasing order of the processing times of the slave tasks.

Proof : Let S be a schedule that schedules jobs in nondecreasing order of the processing times of the slave tasks. Then $p_i(S) = i$. Because $b_i \geq (n-1)a$, $r_i(S) = \max(na, ia + b_i) = ia + b_i$.

If $a \geq c$, then $r_{i+1}(S) = (i+1)a + b_{i+1} \geq ia + b_i + c = r_i(S) + c$, which means that the master machine can schedule c_i at r_i and finish at $r_i + c$ when or before c_{i+1} becomes ready. Thus, $C_i(S) = r_i + c = ia + b_i + c$. The total completion time of S is

$$C(S) = \sum_{i=1}^n (ia + b_i + c) = \frac{1}{2} n(n+1)a + \sum_{i=1}^n b_i + nc .$$

By (2), S is an optimal schedule.

If $a < c$ but $b_{i+1} - b_i \geq c - a$, then we still have $r_{i+1}(S) = (i+1)a + b_{i+1} \geq r_i(S) + c$. Similarly, we can show that S is an optimal schedule. \square

The above lemma does not apply if we only have $a < c$. Consider the three jobs $J_1 = (1, 7, 3)$, $J_2 = (1, 8, 3)$ and $J_3 = (1, 20, 3)$. If we schedule the jobs in nondecreasing order of the processing times of the slave tasks, then the total completion time is 51. But if we schedule the jobs in the order of J_1, J_3 and J_2 , we get a smaller mean flowtime of 50.

The next theorem gives a bound of the worst schedule versus the best schedule.

Theorem 10 *Let S be a schedule that schedules jobs in an arbitrary order. If $a \leq c$, then*

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

If $a > c$, then

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

Proof : Let $C_1 < C_2 < \dots < C_n$ be the completion times in S . 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 .

We let j and all the jobs that finish before j be in G_1 . By assumption, for any $k \leq j$, $k \in G_1$ and $C_k(S) = na + kc$. By (1), the total completion time of the jobs in G_1 in any schedule can not be smaller than that in S . Thus, $C_{G_1}(S) = C_{G_1}^*$.

Let the remaining jobs $k > j$ be in G_2 . Then, for any job k in G_2 , we have $r_k = p_k a + b_k > na$ and $r_k \leq r_{k+1}$. For job $j+1$, we have $C_{j+1} = r_{j+1} + c$. For job $j+2$, we have

$$C_{j+2} = \max(C_{j+1}, r_{j+2}) + c \leq \max(r_{j+2} + c, r_{j+2}) + c = r_{j+2} + 2c .$$

Similarly, we can show that for $j+1 \leq k \leq n$, we have

$$C_k \leq r_k + (k-j)c = p_k a + b_k + (k-j)c .$$

Thus,

$$\begin{aligned} C_{G_2} &= \sum_{k=j+1}^n C_k \\ &\leq \sum_{k=j+1}^n (p_k a + b_k + (k-j)c) \\ &= \left(\sum_{k=j+1}^n (p_k a + b_k) \right) + \sum_{k=1}^{n-j} kc \\ &= \sum_{k=j+1}^n p_k a + \left(\sum_{k=j+1}^n b_k \right) + \left(\sum_{k=1}^{n-j-1} kc \right) + (n-j)c \\ &\leq \sum_{k=j+1}^n ka + \left(\sum_{k=j+1}^n b_k \right) + \sum_{k=1}^{n-j-1} kc + \sum_{k=1}^{n-j} c \\ &= \left(\sum_{k=1}^{n-j} (ka + b_{k+j} + c) \right) + (n-j)ja + \sum_{k=1}^{n-j-1} kc \\ &\leq C_{G_2}^* + (n-j)ja + \frac{1}{2} (n-j)(n-j-1)c \\ &\leq C_{G_2}^* + [(n-j)j + \frac{1}{2} (n-j)(n-j-1)] \max(a, c) \end{aligned}$$

$$\leq C_{G_2}^* + \frac{1}{2}n(n-1) \cdot \max(a, c)$$

So, we have

$$\begin{aligned} C(S) &= C_{G_1} + C_{G_2} \\ &= C_{G_1}^* + C_{G_2} \\ &\leq C_{G_1}^* + C_{G_2}^* + \frac{1}{2}n(n-1) \cdot \max(a, c) \\ &\leq C^* + \frac{1}{2}n(n-1) \cdot \max(a, c) . \end{aligned}$$

By (1), $C^* \geq n^2 a + \frac{1}{2}n(n+1)c$. Therefore, we have

$$\frac{C(S)}{C^*} \leq 1 + \frac{\frac{1}{2}n(n-1) \cdot \max(a, c)}{n^2 a + \frac{1}{2}n(n+1)c} < 1 + \begin{cases} \frac{1}{1+\frac{2a}{c}} & \text{if } a \leq c \\ \frac{1}{2+\frac{c}{a}} & \text{if } a > c \end{cases} .$$

□

Corollary 11 *Let S be a schedule that schedules jobs in an arbitrary order.*

If $a > c$, then $\frac{C(S)}{C^} < \frac{3}{2}$; if $a = c$, then $\frac{C(S)}{C^*} < \frac{4}{3}$; and if $a < c$, then $\frac{C(S)}{C^*} < 2$.*

Proof : The correctness follows directly from (3) and (4). □

We feel that the bounds in the above theorem are not tight. For the case that $a = c = 1$, we find examples approaching the $\frac{4}{3}$ bound. If $n = 5$, let the b_i 's be 2, 3, 4, 5, 6. If we use the b_i 's to represent the jobs, the optimal order of the jobs is (4, 6, 3, 5, 2) with $C^* = 40$, while the worst scheduling order is (6, 5, 4, 3, 2) with total completion time 50. If $n = 9$, let the b_i 's be 4, 5, 6, ..., 12. The optimal order is (8, 9, 10, 12, 5, 11, 7, 4, 6) with $C^* = 126$, while the worst order is (12, 11, 10, 9, 8, 7, 6, 5, 4) with total completion time 162.

We now consider simple heuristics that improve upon the previous bounds.

The next two theorems show that if we schedule jobs in nondecreasing order of the processing times of the slave tasks, then we can obtain better bounds.

Theorem 12 *Suppose $a \geq c$. Let S be a schedule that schedules jobs in non-decreasing order of the processing times of the slave tasks. Then, we have*

$$\frac{C(S)}{C^*} < 1 + \frac{1}{4 + \frac{2c}{a}} , \quad (5)$$

which is a tight bound. If $a = c$, $\frac{C(S)}{C^} < \frac{7}{6}$ and 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 as in the proof of Theorem 10. Let j and all the jobs that finish before j be in G_1 . 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, $C_{G_1}(S) = C_{G_1}^*$.

The remaining jobs are in G_2 . 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,

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

By (1), $C^* \geq n^2 a + \frac{1}{2} n(n+1)c$. Therefore, we have

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

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

To show that the bound is tight 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 the jobs in nonincreasing order of the b_i 's and the total completion time is $n^2 a + \frac{1}{2} n(n+1)c$. However, scheduling the jobs in nondecreasing order of the b_i 's gives the total completion time $\frac{5}{4} n^2 a + \frac{1}{2} n(n+1)c$. \square

It is clear that the schedule obtained by scheduling jobs in nondecreasing order of the processing times of the slave tasks must be an order preserving schedule. Since the optimal canonical and order preserving schedule can not have a mean flowtime smaller than $n^2 a + \frac{1}{2} n(n+1)c$, the above bound also applies to canonical and order preserving schedules. Thus, we have

Corollary 13 *Given n jobs such that $a_i = a$ and $c_i = c$ for all $1 \leq i \leq n$. Suppose that $a > c$. Then, in $O(n \log n)$ time, one can find a canonical and order preserving schedule with mean flowtime at most $\frac{5}{4}$ of the minimum among all canonical and order preserving schedules.*

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

Theorem 14 *Suppose $a < c$. Let S be the schedule that schedules jobs in*

nondecreasing order of the processing times of the slave tasks. Then we have

$$\frac{C(S)}{C^*} < 1 + \frac{1}{2 + \frac{c}{a}} < \frac{4}{3} . \quad (6)$$

If $4a \leq c$, then the bound is $\frac{7}{6}$.

Proof : Let S be the schedule that schedules jobs in nondecreasing order of the processing times of the slave tasks. 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 $C_{G_i}(S)$ be the total completion time of the jobs in G_i under the schedule S . Let $S_{G_i}^*$ be the schedule that minimizes C_{G_i} and let $C_{G_i}^*$ be the total completion time of the jobs in G_i in $S_{G_i}^*$. Then the jobs in G_i must be scheduled before all other jobs in $S_{G_i}^*$. Thus, $C_{G_0}(S) = C_{G_0}^*$. For G_i , there are $\sum_{j=0}^{i-1} x_j$ jobs scheduled before the jobs in G_i in S . Hence, $C_{G_i}(S) \leq C_{G_i}^* + x_i(\sum_{j=0}^{i-1} x_j)a$ for $1 \leq i \leq m$. Therefore, we have

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

Thus,

$$\frac{C(S)}{C^*} \leq 1 + \frac{\frac{1}{2}n^2a}{C^*} \leq 1 + \frac{\frac{1}{2}n^2a}{n^2a + \frac{1}{2}n(n+1)c} < 1 + \frac{1}{2 + \frac{c}{a}} < \frac{4}{3} ,$$

and if $4a \leq c$, we have $\frac{C(S)}{C^*} < \frac{7}{6}$. □

5 Heuristics for no-wait-in makespan

In this section we propose two heuristics for the no-wait-in makespan problem. Both heuristics assume that $a_i = a$ and $c_i = c$ for all i . The first heuristic is for the case when $a \geq c$, while the second heuristic is for the case when $a < c$.

Heuristic 1 - $a \geq c$

1. Sort the jobs in nondecreasing order of the processing times of the slave tasks. Suppose the sorted jobs are in the order of $1, 2, \dots, n$.
2. Schedule task a_1 at time 0, b_1 at time a and c_1 at time $a + b_1$.
3. Repeat until all jobs are scheduled (see Figure 4):

Suppose the jobs $1, 2, \dots, i - 1$ have been scheduled. Find the first idle interval $[t_1, t_2)$ before C_{i-1} whose length is equal to a , schedule a_i at t_1 on the master machine, b_i at $t_1 + a$ on the slave machine and c_i at $t_1 + a + b_i$ on the master machine. If no such idle interval exists, schedule a_i at time C_{i-1} , b_i at $C_{i-1} + a$ and c_i at $C_{i-1} + a + b_i$.

Theorem 15 *Let S be a schedule produced by Heuristic 1 for a given set of n jobs with preprocessing tasks a and postprocessing tasks c and $a \geq c$. Then S is a feasible no-wait-in schedule, and*

$$\frac{C_{\max}(S)}{C_{\max}^*} \leq 3 .$$

Proof : Suppose $b_1 \leq b_2 \leq \dots \leq b_n$. We first show that S is a feasible no-wait-in schedule. For convenience, let $t_a(i)$, $t_b(i)$ and $t_c(i)$ be the times that a_i , b_i and c_i start in S , respectively. By the heuristic, $t_b(i) = t_a(i) + a$, $t_c(i) = t_a(i) + a + b_i$. Thus, we only need to show that S is a feasible schedule.

It is sufficient to show that after we schedule jobs 1, 2, \dots , $i - 1$, the intervals $[t_a(i), t_a(i) + a)$ and $[t_c(i), t_c(i) + c)$ are both idle. By the heuristic, we know that $[t_a(i), t_a(i) + a)$ is idle. Besides, since we always look for the first idle interval $[t_a(i), t_a(i) + a)$ in order to schedule the task a_i , we must have $t_a(i - 1) + a \leq t_a(i)$. Recall that $t_c(i - 1) = t_a(i - 1) + a + b_{i-1}$ and $t_c(i) = t_a(i) + b_i + a$. Since $b_{i-1} \leq b_i$ and $c \leq a$, we have

$$C_{i-1} = t_c(i - 1) + c = t_a(i - 1) + a + b_{i-1} + c \leq t_a(i) + b_i + a = t_c(i) .$$

So, every task c_i starts after the task c_{i-1} completes, which means that the interval $[t_c(i), t_c(i) + c)$ must be idle.

By the above analysis, $C_i < C_{i+1}$ in S . Therefore, the makespan of S is $C_{\max}(S) = C_n = t_a(n) + a + b_n + c$. We first show that $t_a(n) \leq 2C_{\max}^*$.

If there is no idle time before $t_a(n)$, then this is obviously true. Otherwise, there must be some idle intervals before $t_a(n)$. By the heuristic, the length of each idle interval before $t_a(n)$ must be less than a ; additionally, there are two types of intervals: those that overlap with b_i 's and those that are between C_i and $t_c(i + 1)$ for some i , where $1 \leq i \leq n - 2$. Let I_1 denote the total length of first type of idle intervals and I_2 denote the total length of the second type of idle intervals. Because we consider no-wait-in schedules, the first type of idle intervals are inevitable even in an optimal schedule. There are at most $(n - 1)$ idle intervals of the second type all of which are smaller than a . Hence,

$$\begin{aligned} t_a(n) &= \text{non-idle time before } t_a(n) + I_1 + I_2 \\ &< C_{\max}^* + I_2 \end{aligned}$$

$$\begin{aligned}
&< C_{\max}^* + (n - 1)a \\
&< 2C_{\max}^* .
\end{aligned}$$

On the other hand, we know that $a + b_n + c \leq C_{\max}^*$. Therefore,

$$C_{\max}(S) = t_a(n) + a + b_n + c < 3C_{\max}^* .$$

□

Note that if $a = c$, then $(n - 1)a < \frac{1}{2}C_{\max}^*$. By the above analysis, $t_a(n) < \frac{3}{2}C_{\max}^*$. Thus, $C_{\max}(S) < \frac{5}{2}C_{\max}^*$. If $a = c = 1$ and all task times are integers, then there is no idle time of the second type before $t_a(n)$, so $t_a(n) < C_{\max}^*$. Therefore, $C_{\max}(S) = t_a(n) + 1 + b_n + 1 \leq 2C_{\max}^*$. The bound of 2 can be achieved. Consider $n = 2m + 1$ jobs, where $a_i = c_i = 1$ for all $1 \leq i \leq n$, $b_i = 1$ for $1 \leq i \leq 2m$, and $b_{2m+1} = 4m$. The optimal schedule schedules job $2m + 1$ first, and all other jobs completely overlapping with b_{2m+1} . The ratio between the heuristic and the optimal solution for this instance is $1 + \frac{4m}{4m+2}$, which approaches 2 when m is large enough.

Note that Heuristic 1 actually produces order preserving schedules. Since the minimum order preserving and no-wait-in makespan cannot be smaller than the minimum no-wait-in makespan, we have the following corollary.

Corollary 16 *Let S be a schedule produced by Heuristic 1 for a given set of n jobs with preprocessing tasks a and postprocessing tasks c and $a \geq c$. Then S is an order preserving and no-wait-in schedule and*

$$\frac{C_{\max}(S)}{C_{\max}^*} \leq 3 .$$

Heuristic 2 - $a < c$

1. Sort the jobs in nondecreasing order of the processing times of the slave tasks. Suppose the sorted jobs are in the order of $1, 2, \dots, n$.
2. Schedule task a_1 at time 0, b_1 at time a and c_1 at time $a + b_1$.
3. Repeat until all jobs are scheduled (see Figure 5):

Suppose jobs $1, 2, \dots, i - 1$ have been scheduled. Find the first idle interval $[t_1, t_2)$ after $t_a(i - 1) + a$ and before C_{i-1} such that $t_2 - t_1 = a$ and $t_2 + b_i \geq C_{i-1}$. Schedule a_i at t_1 on the master machine, b_i at $t_1 + a$ on the slave machine and c_i at $t_1 + a + b_i$ on the master machine. If no such idle interval exists, schedule a_i at time C_{i-1} , b_i at $C_{i-1} + a$ and c_i at $C_{i-1} + a + b_i$.

Theorem 17 *Let S be a schedule produced by Heuristic 2 for a given set of n jobs with preprocessing tasks a and postprocessing tasks c and $a < c$. Then S is a feasible no-wait-in schedule, and*

$$\frac{C_{\max}(S)}{C_{\max}^*} \leq 3 .$$

Proof : We can use a similar argument as in the proof of Theorem 15 to show that S is a feasible no-wait-in schedule.

Assume that $b_j \leq b_{j+1}$ for $1 \leq j \leq n - 1$. Let C_j be the completion time of job j . By the heuristic, we have $C_1 < C_2 < \dots < C_n$. Suppose that $C_i \leq t_a(n) < C_{i+1}$ for some i . In order to bound $\frac{C_{\max}(S)}{C_{\max}^*}$, we first consider the length of the idle times between C_j and $t_c(j + 1)$ for $1 \leq j \leq i - 1$. There are three cases depending on the number of a tasks scheduled between C_j and $t_c(j + 1)$. Assume that a_k is the last task scheduled before $t_c(j)$. See Figure 6 for an illustration.

There is no a task scheduled between C_j and $t_c(j + 1)$.

In this case the whole period between C_j and $t_c(j + 1)$ is idle. We will prove by contradiction that $t_c(j + 1) - C_j < a$.

Consider the time when we schedule job $k + 1$. By the heuristic, the jobs $1, \dots, k$ have been scheduled at this time. Because a_k is scheduled before c_j , $t_a(k) + a \leq t_c(j)$. By assumption, $b_k \leq b_{k+1}$. Thus, we have

$$C_k = t_a(k) + a + b_k + c \leq t_c(j) + b_k + c = C_j + b_k < C_j + a + b_{k+1} .$$

Let $t_1 = C_j$ and $t_2 = C_j + a$. If $t_c(j+1) - t_1 \geq a$, then $t_2 \leq t_c(j+1)$. Therefore, $[t_1, t_2)$ is an idle interval whose length is equal to a . According to the above inequality, $t_2 + b_{k+1} > C_k$. By the heuristic, a_{k+1} will be scheduled at or before t_1 , which contradicts the assumption that a_{k+1} is scheduled after $t_c(j+1) \geq t_2$. Therefore, we must have $C_j + a > t_c(j+1)$. In other words, the idle interval has length less than a which is less than c .

- (1) There is a single task a_{k+1} scheduled between C_j and $t_c(j+1)$.

By the analysis in Case 5, we know that a_{k+1} must be scheduled immediately after C_j . Thus, the only idle interval between C_j and $t_c(j+1)$ is $[C_j + a, t_c(j+1))$. We will show that $t_c(j+1) - (C_j + a) < c$.

Let $C_j + a = t_1$ and $t_c(j+1) = t_2$. Consider the time when we schedule the job $k+2$. By our assumption, a_{k+2} is scheduled after t_2 . Since $[t_1, t_2)$ is an idle interval, by the heuristic, there are only two possible reasons that a_{k+2} is not scheduled during the interval: (1) the length of the idle interval is less than a , (2) $t_2 - t_1 \geq a$ but $t_2 + b_{k+2} < C_{k+1}$. Since $C_{k+1} = t_1 + b_{k+1} + c$ and $b_{k+2} \geq b_{k+1}$, we must have $t_2 - t_1 < c$. By our assumption, $a < c$. Therefore, in both cases, we have

$$t_c(j+1) - (C_j + a) = t_2 - t_1 < c . \tag{7}$$

- (2) There are several a tasks a_{k+1}, \dots, a_{k+m} scheduled between C_j and $t_c(j+1)$.

As in Case 1, the task a_{k+1} must be scheduled immediately after C_j . So there are at most m disjoint idle intervals during the period from C_j and $t_c(j+1)$. We will show that the length of each of these idle intervals is less than c .

We first consider the length of the idle interval between $t_a(k+p) + a$ and $t_a(k+p+1)$, where $1 \leq p \leq m-1$. Without loss of generality, suppose that this interval has length greater than 0. If $(t_a(k+p) + a) + c > t_c(j+1)$, then the length of the idle interval is obviously less than c . So we may assume that $(t_a(k+p) + a) + c \leq t_c(j+1)$.

Let $t_2 = (t_a(k+p) + a) + c$ and $t_1 = t_2 - a = t_a(k+p) + c$. Consider the time when we schedule job $k+p+1$. It is clear that the interval $[t_1, t_2)$ must be idle at this moment. Since $b_{k+p+1} \geq b_{k+p}$, we have

$$t_2 + b_{k+p+1} = [(t_a(k+p) + a) + c] + b_{k+p+1} \geq t_a(k+p) + a + c + b_{k+p} = C_{k+p} .$$

Thus, by the heuristic, the task a_{k+p+1} must be scheduled at or before t_1 ; that is,

$$t_a(k+p+1) \leq t_1 = t_a(k+p) + c , \quad (8)$$

which means that

$$t_a(k+p+1) - (t_a(k+p) + a) \leq c - a < c . \quad (9)$$

For the interval $[(t_a(k+m) + a), t_a(j+1))$, we can use the same argument as in Case 1 to show that the length of the idle time during this period is less than c .

Now, we can bound the length of the interval $[t_c(1), t_a(n))$. The interval consists of three types of smaller intervals: (i) the intervals occupied by a tasks, (ii) the intervals occupied by c tasks and (iii) idle intervals.

Suppose that a_1, a_2, \dots, a_q are the a tasks scheduled before $t_c(1)$. Then, there are $(n-1-q)$ a tasks that are scheduled during the interval $[t_c(1), t_a(n))$. Therefore, the total length of the type (i) intervals is $(n-1-q)a$. Recall that we assume that $C_i \leq t_a(n) < C_{i+1}$. So the number of c tasks during the interval $[t_c(1), t_a(n))$ is i and the total length of the type (ii) intervals is ic .

By Cases 5, 1 and 2, the length of the idle interval after each a task is at most c . Since there are $(n - 1 - q)$ tasks during the interval $[t_c(1), t_a(n))$, the total length of the idle intervals after these a tasks is at most $(n - 1 - q)c$. If there is no a task between C_j and C_{j+1} for $1 \leq j \leq i - 1$, then there is an idle interval between C_j and C_{j+1} with length at most a . Since there are i jobs that finish before $t_a(n)$, the total length of the idle intervals between the jobs C_j and C_{j+1} with no a task between them and $1 \leq j \leq i - 1$ is at most ia .

Thus, the length of the interval $[t_c(1), t_a(n))$ is

$$\begin{aligned}
t_a(n) - t_c(1) &= \text{total length of type (i) intervals} + \text{total length of type (ii) intervals} \\
&\quad + \text{total length of type (iii) intervals} \tag{10} \\
&\leq (n - 1 - q)a + ic + [(n - 1 - q)c + ia] .
\end{aligned}$$

Next, we bound $t_c(1)$. By the above assumption, a_1, a_2, \dots, a_q are scheduled before C_1 . Using a similar analysis as in Cases 1 and 2, we can show that $t_a(j+1) - t_a(j) \leq c$ (see (8)) for $1 \leq j \leq q - 1$, and $t_c(1) - (t_a(q) + a) < c$ (see (7)). Therefore, we have

$$t_c(1) = a + b_1 \leq t_a(q) + (t_c(1) - t_a(q)) \leq (q - 1)c + (c + a) = qc + a . \tag{11}$$

By (11) and (10), we have

$$\begin{aligned}
t_a(n) &= t_c(1) + (t_a(n) - t_c(1)) \\
&= t_c(1) + [(n - 1 - q)a + ic + ((n - 1 - q)c + ia)] \\
&= (a + b_1) + [(n - 1 - q)a + ic + ((n - 1 - q)c + ia)] \\
&\leq (qc + a) + [(n - 1 - q)a + ic + (n - 1 - q)c + ia] \\
&= (n - q + i)a + (n - 1 + i)c \\
&\leq 2na + 2nc \\
&= 2n(a + c) \\
&\leq 2C_{\max}^* .
\end{aligned}$$

Hence, we have

$$C_{\max} = t_a(n) + a + b_n + c \leq 2C_{\max}^* + (a + b_n + c) \leq 3C_{\max}^* .$$

□

We have not been able to find a set of jobs achieving a ratio of 3. However, we can show that the ratio can approach 2 asymptotically. Let $n = 2m + 1$, $a = 1$ and $c = 1 + \epsilon$, where ϵ is arbitrarily small. Suppose $b_i = 1 + \epsilon$ for $1 \leq i \leq 2m$ and $b_{2m+1} = m(4 + 3\epsilon)$. The makespan of the schedule produced by Heuristic 2 is $2m(4 + 3\epsilon) + (2 + \epsilon)$. However, the optimal schedule schedules job $2m + 1$ first and all other jobs completely overlapping with b_{2m+1} . Thus, the optimal makespan is $m(4 + 3\epsilon) + (2 + \epsilon)$. If m is very large, the ratio between the heuristic and the optimal solution approaches 2 arbitrarily closely.

Note that Heuristic 2 produces schedules that are order preserving. Since the minimum order preserving and no-wait-in makespan cannot be smaller than the minimum no-wait-in makespan, Theorem 17 implies the following corollary.

Corollary 18 *Let S be a schedule of a given set of n jobs with preprocessing tasks a and postprocessing tasks c and $a < c$. Then S is an order preserving and no-wait-in schedule, and*

$$\frac{C_{\max}(S)}{C_{\max}^*} \leq 3 .$$

6 Conclusion

In this paper we studied scheduling problems in the single-master master-slave model. We showed that many makespan and mean flowtime problems

with constraints such as canonical, order preserving and no-wait-in, are NP-hard in the strong sense. Motivated by the computational complexity of the problem, we proposed several heuristics for the special case when $a_i = a$ and $c_i = c$ for all i , for both the canonical mean flow time problem and the no-wait-in makespan problem. We analyzed the heuristics by proving their worst-case performance bounds.

There are several questions that have not been answered in this paper. The complexity of minimizing the order preserving mean flowtime is still open. We have not been able to analyze any heuristics with arbitrary preprocessing and postprocessing task times, for both the canonical mean flowtime and the no-wait-in makespan problems. Other objectives such as maximum lateness and total tardiness, have not been studied. In view of the NP-hardness of makespan and mean flowtime, minimizing these objectives must also be NP-hard.

References

- [1] M. Dell'Amico. Shop problems with two machine and time lags. *Operations Research*, 44, 5, 777-787, 1996.
- [2] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.
- [3] W. Kern and W. Nawijn. *Scheduling multi-operation jobs with time lags on a single machine*. University of Twente, 1993.
- [4] S. Sahni. Scheduling master-slave multiprocessor systems. *IEEE Trans. on Computers*, 45, 10, 1195-1199, 1996.
- [5] S. Sahni and G. Vairaktarakis. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9, 357-377, 1996.

- [6] 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.
- [7] G. Vairaktarakis. Analysis of algorithms for master-slave system. IIE Transactions, 29, 11, 939-949, 1997.
- [8] 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.

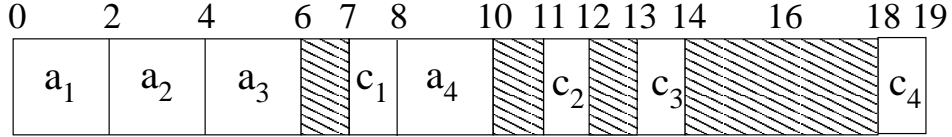


Fig. 4. Illustration of Heuristic 1: jobs are $(2, 5, 1)$, $(2, 7, 1)$, $(2, 7, 1)$ and $(2, 8, 1)$.

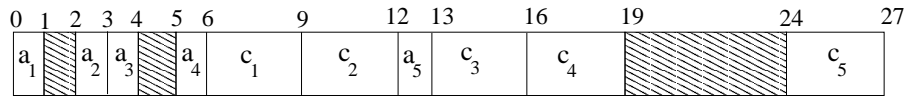


Fig. 5. Illustration of Heuristic 2: jobs are $(1, 5, 3)$, $(1, 6, 3)$, $(1, 9, 3)$, $(1, 10, 3)$ and $(1, 11, 3)$.

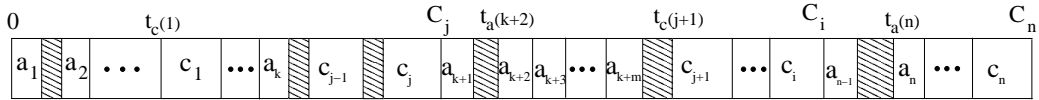


Fig. 6. Illustration of the proof of Theorem 17. The idle interval between c_{j-1} and c_j has length less than a ; the idle interval between a_{k+1} and a_{k+2} has length less than $c - a$ and the idle interval between a_{k+m} and c_{j+1} has length less than c .