Streaming algorithms for multitasking scheduling with shared processing

Bin Fu

Yumei Huo

Hairong Zhao

Streaming Algorithms for Multitasking Scheduling with Shared Processing

Bin Fu[a], Yumei Huo[b], Hairong Zhao[b]

[a]*Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, 78539, USA*
[b]*Department of Computer Science, College of Staten Island, CUNY, Staten Island, NY, 10314, USA*
[c]*Department of Computer Science, Purdue University Northwest, Hammond, IN, 46323, USA*

Abstract

In this paper, we design the first streaming algorithms for the problem of multitasking scheduling on parallel machines with shared processing. In one pass, our streaming approximation schemes can provide an approximate value of the optimal makespan. If the jobs can be read in two passes, the algorithm can find the schedule with the approximate value. This work not only provides an algorithmic big data solution for the studied problem, but also gives an insight into the design of streaming algorithms for other problems in the area of scheduling.

Keywords: streaming algorithm, multitasking scheduling, shared processing, parallel machine, makespan, approximation scheme

1. Introduction

In recent years, with more and more data generated in all applications, the dimension of the computation increases rapidly. As in many other research areas, the need for providing solutions under big data also emerges in the area of scheduling. In this paper, we study the data stream model of multitasking scheduling problem. Under this data model, the input data is massive and cannot be read into memory; the goal is to design streaming algorithms to approximate the optimal solution in a few passes (typically just one) over the data and using limited space.

As Muthukrishnan addressed in his paper [24], traditionally, data are fed from the memory and one can modify the underlying data to reflect the updates; real time queries are also simple by looking up a value in the memory, as we see in banking and credit transactions; as far as complex analyses are concerned, such as trend analysis, forecasting, etc., operations are usually performed offline. However, in the modern world, with more and more data generated in the monitoring applications such as atmospheric, astronomical, networking, financial, sensor-related fields, etc., the automatic data feeds are needed for many tasks. For example, large amount of data need to be fed and processed in a short time to monitor complex correlations, track trends, support exploratory analyses and perform complex tasks such as classification, harmonic analysis etc. These tasks are time critical and thus it is important to process them in near-real time to accurately keep pace with the rate of stream updates and reflect rapidly changing trends in the data. With more data generated and more demands of data streams processing for now and in the future, the researchers are facing the questions: Given a certain amount of resources, a data stream rate and a particular analysis task, what can we (not) do?

While some methods are available for processing large amount of data of these time critical tasks, such as making things parallel, controlling data rate by sampling or shedding updates, rounding data structures to certain block boundaries, using hierarchically detailed analysis, etc., these approaches are ultimately limiting.

A natural approach to dealing with data streams involves approximations and developing algorithmic principles for data stream algorithms. Streaming algorithms were initially studied by Munro and Paterson in 1978 ([23]), and then by Flajolet and Martin in 1980s ([9]). The model was formally established by Alon, Matias, and Szegedy in [2] and has received a lot of attention since then. Formally, streaming algorithms are algorithms for processing the input where some or all of of the data is not available for random access but rather arrives as a sequence of items and can be examined in only a few passes (typically just one). The performance of streaming algorithms is measured by three factors: the number of passes the algorithm must run over the stream, the space needed and the updating time of the algorithm.

In this paper, we study the streaming algorithms for the problem of multitasking scheduling with shared processing. Over the past couple of decades, the problem of multitasking scheduling has attracted a lot of attention in the service industries where workers frequently perform multiple tasks by switching from one task to another. For example, in health care, 21% of hospital employees spend their working time on more than one activity [25], and in consulting where workers usually engage in about 12 working spheres per day [11]. Although in the literature some research has been done on the effect of multitasking ([11], [28], [6], [26]), the study on multitasking in the area of scheduling is still very limited ([15], [16], [27], [29]).

Hall, Leung and Li ([16]) proposed a multitasking scheduling model that allows a team to continuously work on its main, or primary tasks while a fixed percentage of its processing capacity may be allocated to process the routinely scheduled activities as they occur. Some examples of the routinely scheduled activities are administrative meetings, maintenance work, or meal breaks. In these scenarios, some team members need to be assigned to perform these routine activities while the remaining team members still focus on the primary tasks. Since the routine activities are essential to the maintenance of the overall system in many situations, they are usually managed separately and scheduled independently of the primary jobs. When these multitasking problems are modeled in the scheduling theory, a working team is viewed as a machine which may have some periods during which routine jobs and primary jobs share the processing capacity.

In [16], it is assumed that there is only a single machine and the machine capacity allocated to routine jobs is the same for all routine jobs. In this paper, we generalize this model to parallel machine environment and allow the machine capacity allocated to routine jobs to vary from one to another. In practice, it is not uncommon that different number of team members are needed to perform different routine jobs during different time periods.

In many circumstances, it is necessary to have service continuously available for primary jobs, such as in many companies' customer service and technical support departments, the service must be continuous for answering customers' calls and for troubleshooting the customers' product failures. So at least one member from the team is needed to provide these service at any time while the size of a team is typically of ten or fewer members as recommended by Dotdash Meredith Company in their management research. To model this, we allow the capacities allocated for primary jobs on some machines to have a constant lower bound.

1.1. Problem Definition

Formally, our problem can be defined as follows. We are given m identical machines $\{M_1, M_2, \ldots, M_m\}$ and a set $N = \{1, \ldots, n\}$ of primary jobs that are all available for processing at time 0. Each primary job $j \in N$ has a processing time p_j and can be processed by any one of the machines uninterruptedly. Each machine M_i has k_i shared processing intervals during which only a fraction of machine capacity can be allocated to these primary jobs

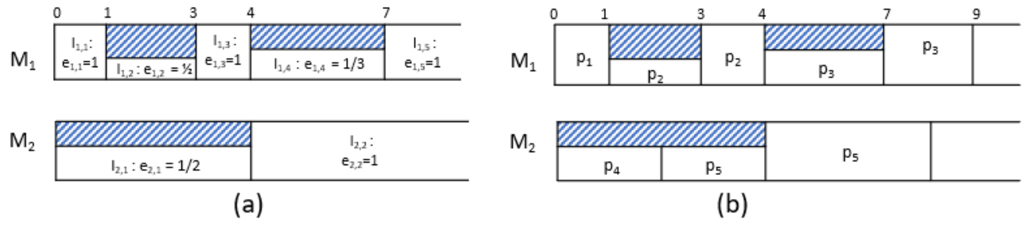

Figure 1: An example of multitasking scheduling on 2 machines with shared processing, 3 routine jobs are shown as shaded intervals. (a) The intervals and the sharing ratios; (b) A schedule of five primary jobs: $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 1$, $p_5 = 5$.

due to the fact some capacity has been pre-allocated to routine jobs. We use "sharing ratio" to refer the fraction of the capacity allocated to the primary jobs. The total number of these intervals is $\tilde{n} = \sum_{1\le i\le m} k_i$. For simplicity, we will treat those intervals with full capacity as intervals with sharing ratio 1. Apparently, each machine M_i has $O(k_i)$ intervals in total. Without loss of generality, we assume that these intervals are given in sorted order, denoted as $I_{i,1} = (0, t_{i,1}]$, $I_{i,2} = (t_{i,1}, t_{i,2}]$, $\ldots$, and their corresponding sharing ratios are $e_{i,1}$, $e_{i,2}$, $\ldots$, all of which are in the range of $(0, 1]$, see Figure 1(a) for an illustration of machine intervals and Figure 1(b) for an illustration of a schedule of primary jobs in these intervals. For any schedule S, let $C_j(S)$ be the completion time of the primary job j in S. If the context is clear, we use C_j for short. The makespan of the schedule S is $\max_{1\le j\le n}\{C_j\}$. The objective is to find a schedule of the primary jobs to minimize the makespan.

In this paper, we consider the above scheduling problem under the data stream model. Specifically, we study the problem that the number of primary jobs is so big that jobs' information cannot be stored in the memory but can only be scanned in one or more passes. Extending the three-field $\alpha \mid \beta \mid \gamma$ notation introduced by Graham et al. [12], our problem is denoted as $P_m \mid$ stream, share $\mid C_{max}$ if the sharing ratios are arbitrary; and if the sharing ratio is at least e_0 for the intervals on the first m_1 $(1 \le m_1 \le m-1)$ machines but arbitrary for other $(m - m_1)$ machines, our problem is denoted as $P_m \mid$ stream, share, $e_{i,k} \geq e_0$ for $i \leq m_1 \mid C_{max}$. The corresponding problems under the traditional data model will be denoted as $P_m \mid$ share $\mid C_{max}$ and $P_m \mid$ share, $e_{i,k} \geq e_0$ for $i \leq m_1 \mid C_{max}$, respectively.

1.2. Literature Review

Various models of shared processing scheduling have been studied in the literature. In the model studied by [7], [17], [8], jobs have their own private processors and can also be processed by other processors which are shared by other jobs to reduce the job's completion time due to processing time overlap. In the model studied by Baker and Nuttle [3], all the resources together are viewed as one machine which has varying availability over time and jobs are scheduled on this single machine with varying capacity. The authors showed that a number of well-known results for classical single-machine problems can be applied with little or no modification to the corresponding variable-resource problems. Then Hirayama and Kijima [18] studied this problem when the machine capacity varies stochastically over time. Adiri and Yehudai [1] studied the problem on single and parallel machines such that the service rate of a machine can only be changed when a job is completed.

The shared processing multitasking model studied in this paper was first proposed by Hall et. al. in [16]. In this model, machine may have reduced capacity for processing primary jobs in some periods where routine jobs are scheduled and share the processing with primary jobs. They studied this model in the single machine environment and assumed that the sharing ratio is a constant e for all the shared intervals. For this model, it is easy to see that the makespan is the same for all schedules that have no unnecessary idle time. The authors in [16] showed that the total completion time can be minimized by scheduling the jobs in non-decreasing order of the processing time, but it is unary NP-Hard for the objective function of total weighted completion time. When the primary jobs have different due dates, the authors gave polynomial time algorithms for maximum lateness and the number of late jobs.

For our studied problems, if $e_{i,k} = 1$ for all time intervals, that is, there are no routine jobs, the problem becomes the classical parallel machine scheduling problem $P_m \mid\mid C_{max}$. For this problem, Graham studied the performance of List Scheduling rule ([13]) and Longest Processing Time ([14]) rule. When the number of machines m is fixed, Horowitz and Sahni [20] developed a fast approximation scheme. Later, Hochbaum and Shmoys [19] designed a approximation scheme for this problem when m is arbitrary.

On the other hand, if $e_{i,k} \in \{0, 1\}$ for all time intervals, i.e. at any time the machine is either processing a primary job or a routine job but not both, then our problem reduces to the problem of parallel machine scheduling with availability constraint. This problem is also NP-hard and approximation algorithms are developed in [21] and [22].

For the general problem, $P_m \mid share \mid C_{max}$, i.e., the sharing ratios $e_{i,k}$ are arbitrary values in $(0, 1]$, Fu, et al. [10] showed that there is no approximation algorithm for the problem unless $P = NP$. Then they studied $P_m \mid share, e_{i,k} \geq e_0 \mid C_{max}$ and $P_m \mid share, e_{i,k} \geq e_0$ for $i \leq m_1 \mid C_{max}$, where e_0 is a constant. They analyzed the performance of some classical heuristics for the two problems. Finally, they developed an approximation scheme for the problem $P_m \mid share, e_{i,k} \geq e_0$ for $i \leq m_1 \mid C_{max}$.

All the above results are for the problems under the traditional data model where all data can be stored in memory. There is no result for the studied problems under the data stream model where the input data is massive and cannot be read into memory.

While streaming algorithms have been studied in the field of statistics, optimization, and graph algorithms (see surveys by Muthukrishnan [38] and McGregor [37]) in the last twenty years, very little research is conducted in the area of scheduling and operations research in general. In [4], Beigel and Fu developed a randomized streaming approximation scheme for the bin packing problem such that it needs only constant updating time and constant space, and outputs an $(1+\epsilon)$-approximation in $(1/\epsilon)^{O(1/\epsilon)}$. In [5], Cormode and Veselý designed a streaming asymptotic $(1+\epsilon)$-approximation algorithms for bin packing and $(d+\epsilon)$-approximation for vector bin packing in d dimensions. For the related vector scheduling problem, they showed how to construct an input summary in space $\widetilde{O}(d^2 \cdot m/\epsilon^2)$ that preserves the optimum value up to a factor of $(2 - 1/m + \epsilon)$, where m is the number of identical machines.

1.3. New Contribution

In this work, we develop the first steaming algorithms for the generalized multitasking shared processing scheduling problems. We assume the machine information is known beforehand, but all the job information including the number of jobs and the processing times are unknown until they are read. Our streaming algorithms are approximation schemes. In one pass, for any positive constant ϵ, the algorithms return a value that is at most $(1+\epsilon)$ times the optimal value of the makespan and it takes constant updating time to process each job in the stream. If the jobs can be input in two passes, the algorithms can generate the schedule with constant processing time for each job in the stream. We show that if an estimate of the maximum processing time can be obtained from priori knowledge, the approximation scheme can be implemented more efficiently. It should be noted that the approximation scheme given by Fu, et al. [10] does not work under the data stream model.

So in this paper, we develop a different approximation scheme.

2. Streaming Algorithms

In this section, we will present our streaming algorithms for the multi-tasking scheduling problem $P_m \mid \text{stream, share, } \; e_{i,k} \geq e_0 \text{ for } i \leq m_1 \mid C_{\max}$.

We first design an approximation scheme for the studied scheduling problem under the tradition model where can be stored in the memory. We then adapt this algorithm for the following three cases, respectively: (1) the maximum job processing time, p_{max}, is given; (2) an estimate of p_{max} is given; (3) no information about p_{max} is given. In all these cases, the number of jobs is not known until all jobs are read.

2.1. An Approximation Scheme under Traditional Data Model

In [10], an approximation scheme has been developed for the studied scheduling problem under traditional data model, where the main idea is to enumerate all assignments for large jobs, prune the similar assignments, schedule the small jobs to all the obtained large job assignments, and finally pick the best schedule. This approximation scheme requires storing all the jobs information and cannot be applied to the data stream model.

So in this subsection we develop a different approximation scheme for the traditional data model but can be adapted to work under the data stream model as well. The idea of the approximation algorithm is to find the best assignment of large jobs and then generate a single schedule based on this large job assignment.

We first introduce a notation before describing our algorithm. For any time t, we let $A_i(t)$ denote the total amount of processing time of the jobs that can be processed during $(0, t]$ on machine M_i. Formally, $A_i(t) = t \cdot e_{i,1}$ if $t \in I_{i,1} = (0, t_{i,1}]$, and $A_i(t) = A_i(t_{i,k}) + (t - t_{i,k}) \cdot e_{i,k+1}$ if $t \in I_{i,k+1} = (t_{i,k}, t_{i,k+1}]$.

By the definition of $A_i(t)$, if N_i is a set of jobs assigned to machine M_i and $\sum_{j \in N_i} p_j \leq A_i(t)$, then the jobs in N_i can be completed before time t on machine M_i. Let $A(t)$ be the total amount of processing time of the jobs that can be processed during $(0, t]$ on all machines, i.e., $A(t) = \sum_{i=1}^{m} A_i(t)$. Our algorithm is outlined as follows.

Algorithm 1

Input:

- Parameters m, m_1, e_0, and ϵ
- The intervals $(0, t_{i,1}]$, $(t_{i,1}, t_{i,2}]$, $\ldots$ on machine M_i, $1 \leq i \leq m$, and their sharing ratios $e_{i,1}$, $e_{i,2}$, $\ldots$
- The number of jobs n, and the jobs' processing time p_j, $1 \leq j \leq n$.

Output: A schedule of n jobs

Steps:

1. Identify the set of large jobs, JS, which is constructed below:

   (i) Choose γ_0 and N_0 as follows:

$$2^{\gamma_0} = \left\lceil \frac{m + m_1 - 1}{\frac{\epsilon}{2} \cdot m_1 \cdot e_0} \right\rceil \tag{1}$$

$$N_0 = \left\lceil \frac{m(m + m_1 - 1)}{\frac{\epsilon}{4} \cdot e_0 \cdot m_1} \right\rceil \tag{2}$$

   (ii) Let p_{max} be the largest processing time of all the jobs. Define $\gamma_0 + 2$ continuous intervals, $IH_{-1} = (0, 2^{q_0}]$, $IH_0 = (2^{q_0}, 2^{q_0+1}]$, $\ldots$,$IH_k = (2^{q_0+k}, 2^{q_0+k+1}]$, $\ldots$, $IH_{\gamma_0} = (2^{q_0+\gamma_0}, 2^{q_0+\gamma_0+1}]$, such that $p_{max} \in IH_{\gamma_0}$, i.e., $2^{q_0+\gamma_0} < p_{max} \leq 2^{q_0+\gamma_0+1}$. Correspondingly, partition the jobs in $N = \{1, 2, \cdots, n\}$ into $\gamma_0 + 2$ groups, H_{-1},

H_0, H_1, …, H_{γ_0}, based on their processing time: if $p_j \in IH_k$, then add job j to H_k.

   (iii) Let k_L, $0 \leq k_L \leq \gamma_0$ be the largest index of the group that contains at least N_0 jobs; if no such group exists, i.e. all the groups have less than N_0 jobs, we let $k_L = -1$. Let $JS = \cup_{k>k_L} H_k$ be the set of large jobs, and the remaining jobs are considered as small jobs.

2. For each possible assignment of jobs in JS, find a time t such that

   (i) Both of the following conditions hold for t: (a) $A(t) \geq \sum_{i=1}^{n} p_i$ and (b) for $1 \leq i \leq m$, $A_i(t) \geq P_i^L$ where P_i^L is the total processing time of the large jobs assigned on M_i.

   (ii) At least one of (a) and (b) doesn't hold when t is replaced by $\frac{t}{1+\epsilon/2}$;

3. Among all the assignments of jobs in JS, we pick the one that is associated with the smallest t and do the following:

   (i) Assign the small jobs in any order to the machines so that at most one job finishes at or after t on each machine M_i

   (ii) Remove the small jobs that finishes after t on machine M_i, $i > m_1$ and schedule them to the first m_1 machines so that there are at most $\left\lceil \frac{m-1}{m_1} \right\rceil$ jobs finishing after t on M_i, $i \leq m_1$.

4. Return the obtained schedule.

Lemma 1. *Algorithm 1 outputs a $(1+\epsilon)$-approximation for the scheduling problem $P_m|share,\ \ e_{i,k} \geq e_0$ for $i \leq m_1|C_{\max}$.*

Proof. First let us look at the assignment of the jobs in JS that is the same as the optimal schedule. In step 2 of the algorithm, we find the time t^*

associated with this large job assignment such that both (a) and (b) hold for t^*, but either (a) or (b) doesn't hold when t^* is replaced by $\frac{t^*}{1+\epsilon/2}$, which means that $\frac{t^*}{1+\epsilon/2} < C^*_{max}$, i.e., $t^* \leq (1+\frac{\epsilon}{2})C^*_{max}$. In step 3 of the algorithm, the large job assignment associated with the smallest t is selected, and we have $t \leq t^* \leq (1+\frac{\epsilon}{2})C^*_{max}$.

For the selected large job assignment in step 3 of the algorithm, by condition (b), all large jobs are finished before or at t. By condition (a), $A(t) \geq \sum_{i=1}^{n} p_i$, there must be at most $(m-1)$ small jobs that finish after t in step 3(i), and thus in step 3(ii) we can distribute them onto the first m_1 machines so that at most $\left\lceil \frac{m-1}{m_1} \right\rceil$ small jobs on each of these machines finish after t. Hence, the job with the largest completion time must be on one of the first m_1 machines.

Let j be the job such that $C_j = C_{max}$, and assume j is scheduled on machine M_i, $i \leq m_1$. Let U be the set of jobs on M_i that finish after t. As the sharing ratio is at least e_0 on M_i, $C_j \leq t + \sum_{u \in U} \frac{p_u}{e_0}$. Note that all jobs in U are small jobs, i.e. $p_u \leq 2^{q_0+k_L+1}$ for each job $u \in U$. From the analysis above, $|U| \leq \left\lceil \frac{m-1}{m_1} \right\rceil$. So,

$$C_{max} = C_j \leq t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0+k_L+1}. \tag{3}$$

If $k_L > -1$, there are at least N_0 jobs in H_{k_L}, and each of which has a processing time in the range of $(2^{q_0+k_L}, 2^{q_0+k_L+1}]$. Therefore, we have $C^*_{max} \geq$

$\frac{N_0 \cdot 2^{q_0+k_L}}{m}$, which implies $2^{q_0+k_L+1} \leq \frac{2mC^*_{max}}{N_0}$. Thus, we have

$$\begin{aligned} C_{max} = C_j &\leq t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0+k_L+1} \\ &\leq t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot \frac{2mC^*_{max}}{N_0} \\ &\leq t + \frac{m+m_1-1}{m_1} \cdot \frac{1}{e_0} \cdot \frac{2mC^*_{max}}{N_0} \\ &\leq \left(1+\frac{\epsilon}{2}\right) C^*_{max} + \frac{m+m_1-1}{m_1} \cdot \frac{1}{e_0} \cdot \frac{2mC^*_{max}}{N_0} \qquad \text{by Equation (2)} \\ &\leq (1+\epsilon)C^*_{max} \ . \end{aligned}$$

Otherwise, $k_L = -1$, using $2^{q_0+\gamma_0} < p_{max} \leq 2^{q_0+\gamma_0+1}$, we have $2^{q_0+k_L+1} = 2^{q_0} \leq \frac{p_{max}}{2^{\gamma_0}} \leq \frac{C^*_{max}}{2^{\gamma_0}}$. Therefore, we can get

$$\begin{aligned} C_{max} = C_j &\leq t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0+k_L+1} \\ &= t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0} \\ &\leq t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot \frac{C^*_{max}}{2^{\gamma_0}} \\ &\leq t + \frac{m+m_1-1}{m_1} \cdot \frac{1}{e_0} \cdot \frac{C^*_{max}}{2^{\gamma_0}} \\ &\leq \left(1+\frac{\epsilon}{2}\right) C^*_{max} + \frac{m+m_1-1}{m_1} \cdot \frac{1}{e_0} \cdot \frac{C^*_{max}}{2^{\gamma_0}} \qquad \text{by Equation (1)} \\ &\leq (1+\epsilon)C^*_{max} \ . \end{aligned}$$

So in both cases, we have $C_{max} \leq (1+\epsilon)C^*_{max}$. □

Now we analyze the running time of Algorithm 1. Let

$$t_1(m, m_1, \tilde{n}, \epsilon, e_0)) = \left(m^{O(\frac{m^2}{\epsilon e_0 m_1} \log \frac{m}{\epsilon e_0 m_1})} \left(\log(\tfrac{1}{\epsilon} \log \tfrac{m}{e_0}) \cdot \sum_{1 \leq i \leq m} \log k_i \right) \right) \ , \quad (4)$$

then we have the following lemma for the running time of Algorithm 1.

Lemma 2. *Let ϵ be a real number in $(0,1)$, Algorithm 1 runs in time*

$$O\left(n + \tilde{n} + t_1(m, m_1, \tilde{n}, \epsilon, e_0)\right),$$

which is linear $O(n+\tilde{n})$ when m is constant.

Proof. In Step 1, we find the set of large jobs JS which can be done in $O(n)$ time. The number of jobs in JS is at most $N_0(\gamma_0+1)$.

In step 2, we consider all possible assignments of jobs in JS, and there are $O(m^{O(N_0\gamma_0)})$ of them. By Equation (2), we have $N_0 = O\left(\frac{m(m+m_1-1)}{\epsilon e_0 \cdot m_1}\right) = O\left(\frac{m^2}{\epsilon e_0 \cdot m_1}\right)$. By Equation (1), we have $\gamma_0 = O\left(\log \frac{m+m_1-1}{\epsilon e_0 \cdot m_1}\right) = O\left(\log \frac{m}{\epsilon e_0 \cdot m_1}\right)$.

Let $P = \sum_{i=1}^{n} p_i$, then the time t associated with each assignment of jobs in JS is between the lower bound $LB = P/m$ and the upper bound $UB = \frac{P}{e_0} = LB \cdot \frac{m}{e_0}$. It takes $O(\log(UB-LB))$ time to search the exact smallest t such that both (a) and (b) hold. To speed up the algorithm, instead of searching the exact smallest t, we search t with a $(1+\epsilon/2)$-approximation in step 2(ii). Specifically, we only need to consider those time points whose values are $LB \times (1+\frac{\epsilon}{2})^x$, where $0 \le x \le \log_{1+\epsilon/2} \frac{UB}{LB} = O\left(\frac{1}{\epsilon}\log\frac{m}{e_0}\right)$. In this way, we can use binary search to find the corresponding x in $O\left(\log(\frac{1}{\epsilon}\log\frac{m}{e_0})\right)$ iterations.

In each iteration of binary search, for the specific $t = LB \cdot (1+\frac{\epsilon}{2})^x$, we need to calculate $A_i(t)$, $1 \le i \le m$, which is the total amount of jobs that can be processed by t on machine M_i. To do so, we use binary search to find the interval $(t_{i,k-1}, t_{i,k}]$ such that $t \in (t_{i,k-1}, t_{i,k}]$ in $O(\log k_i)$ time, then compute $A_i(t) = A_i(t_{i,k-1}) + (t - t_{i,k-1})e_{i,k}$, which can be done in constant time if $A_i(t_{i,k})$ is known; indeed, we can pre-calculated $A_i(t_{i,k})$ for all i and $t_{i,k}$ in $O(\tilde{n})$ time. Once $A_i(t)$ is calculated for all i, $1 \le i \le m$, we have $A(t) = \sum A_i(t)$. In total, it takes $O(\sum_{i=1}^{m}\log k_i)$ time to calculate $A_i(t)$ and $A(t)$, and check conditions (a) and (b) for t. With the same running time, one can calculate $A_i(\frac{t}{1+\epsilon/2})$ and check conditions (a) and (b) for $\frac{t}{1+\epsilon/2}$. Therefore, the time of finding t associated with a specific large job assignment is $\left(\log(\frac{1}{\epsilon}\log\frac{m}{e_0})\cdot\sum_{1\le i\le m}\log k_i\right)$. The total time for all assignments would be $t_1(m, m_1, \tilde{n}, \epsilon, e_0)$, which is given by Equation (4).

In step 3, we select the assignment associated with the smallest t and schedule the small jobs based on this assignment. This can be done in time $O(n + \tilde{n})$.

Adding the time in all steps, we get the total time as stated in the lemma.

□

By Lemmas 1 and 2, we have the following theorem.

Theorem 3. *Let ϵ be a real number in $(0, 1)$, and m be a constant. Then there is a $(1 + \epsilon)$-approximation scheme for $P_m|$ share $e_{i,k} \ge e_0$ for $i \le m_1|C_{\max}$ in time $O(n + \tilde{n})$.*

Now in the following we will adapt Algorithm 1 so it works under the streaming model where the processing times of the jobs are given as a stream. In Algorithm 1, we classify jobs as large or small and then process them separately. Whether a job is large or small is determined by the parameters γ_0, N_0 and q_0 which in turn are determined by m, m_1, e_0, ϵ and p_{max}. While m, m_1, e_0 and ϵ are parts of the input, p_{max} may or may not be. We will first give the streaming algorithm when p_{max} is given as an input.

2.2. Streaming Algorithm When p_{max} is Given

Given P_{max} as part of the input, we can partition the jobs into groups and classify a job as large or small as in Algorithm 1, but the challenge is that we can not store the processing times of all jobs. For each group H_k, $-1 \le k \le \gamma_0$, we maintain a triple (n_k, P_k, JS_k), where n_k is the number of jobs in H_k, P_k is the total processing time of jobs in H_k, and JS_k contains the set of jobs in H_k if $n_k < N_0$ and $k \geq 0$; otherwise, JS_k is an empty set. That is, we keep the processing times of the potential large jobs only. And thus we only keep the processing times of at most $N_0(\gamma_0 + 1)$ large jobs. We update the triples for the groups as jobs are scanned one by one. Once all the jobs are input, we have the complete information of large jobs and can use step 2 of Algorithm 1 to find the assignment of large jobs associated with the smallest t. Since we are concerned with the approximate value of the optimal makespan, we don't need to schedule the small jobs as in step 3 of Algorithm 1. Instead we can directly return the approximate value of the optimal makespan as $t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0+k_L+1}$. The algorithm can be described as follows.

Algorithm 2

Input:

- Parameters m, m_1, e_0, and ϵ
- The intervals $(0, t_{i,1}]$, $(t_{i,1}, t_{i,2}]$, $\ldots$ on M_i and their sharing ratios $e_{i,1}$, $e_{i,2}$, $\ldots$, respectively $(1 \leq i \leq m)$
- the maximum processing time p_{max}
- The jobs' processing time p_j (stream input), $1 \leq j \leq n$.

Output: An approximate value of the optimal makespan.

Steps:

1. Identify the set of large jobs, JS.

    a. Choose γ_0, N_0, q_0 as in Algorithm 1.

    b. Read jobs one by one and do the following:

```
        i. if $p_j \leq 2^{q_0}$
               $k = -1$
           else
               $k = \lceil \log_2 p_j \rceil - q_0 - 1$
       ii. $P_k = P_k + p_j$
      iii. $n_k = n_k + 1$
       iv. if $k = -1$ or $n_k \geq N_0$
               reset $JS_k = \emptyset$
           else
               $JS_k = JS_k \cup \{j\}$
   c. $P = \sum_{-1 \leq k \leq \gamma_0} P_k$
   d. Let $k_L$, $0 \leq k_L \leq \gamma_0$, be the largest index of the group such that
      $n_k \geq N_0$; if $k_L$ doesn't exist, let $K_L = -1$. Let $JS = \cup_{k > k_L} JS_k$.
2. For each possible assignment of jobs in $JS$, find time $t$ associated with
   the assignment as in Algorithm 1
3. Find the smallest $t$ from the above step, and return the value $t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot$
   $\frac{1}{e_0} \cdot 2^{q_0 + k_L + 1}$.
```

Theorem 4. *Let ϵ be a real number in $(0, 1)$, assume that p_{max} is a given input, then Algorithm 2 is a one-pass streaming algorithm for $P_m \mid stream,\ share,\ e_{i,k} \geq e_0 \text{ for } i \leq m_1 \mid C_{\max}$ and it takes*

1. *$O(1)$ time for processing each job in the stream,*
2. *$O\left(\tilde{n} + \frac{m^2}{\epsilon e_0 m_1} \cdot \log \frac{m}{\epsilon e_0}\right)$ space, and*
3. *$O(\tilde{n} + t_1(m, m_1, \tilde{n}, \epsilon, e_0))$ time*

to find an approximate value of the optimal makespan by a $(1+\epsilon)$ *factor.*

Proof. We first show that the returned value is an approximate value of C^*_{max} by a $(1+\epsilon)$ factor. Since we keep all the processing times of large jobs, the selected large job assignment and the associated t obtained in Algorithm 2 are exactly the same as Algorithm 1. Following the same argument in Lemma 1, we can directly return $(t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0+k_L+1})$ as an approximate value of the optimal makespan, which is at most $(1+\epsilon)C^*_{max}$.

The storage used for the streaming algorithm is mainly $O(\gamma_0)$ triples including at most $O(N_0\gamma_0)$ jobs. The total storage for these triples is $O(N_0\gamma_0) = O\left(\frac{m^2}{\epsilon e_0 m_1} \cdot \log \frac{m}{\epsilon e_0}\right)$. In addition, we need to store $O(\tilde{n})$ processing sharing intervals.

As seen in step 1b, each job is processed in $O(1)$ time. After step 1, we get the set of larges jobs, JS, including at most $O(N_0\gamma_0)$ jobs. After $A(t_{i,k})$ and $A_i(t_{i,k})$ are pre-calculated in $O(\tilde{n})$, the time for step 2 is the same as in Theorem 3, $t_1(m, m_1, \epsilon, e_0, n)$. □

If the jobs can be read in a second pass, we can return a schedule of all jobs whose makespan is at most $(1+\epsilon)C^*_{max}$. Specifically, in the first pass, we store the assignment of jobs in JS associated with the minimum t obtained from step 2 as well as the total processing time, P_i, of jobs in JS that are assigned to machine M_i, $1 \le i \le m$. In the second pass, we only need to schedule the small jobs. When a job is read, if it is a job in JS, we don't need to do anything; otherwise we schedule it to the machine so that at most $\left\lceil \frac{m-1}{m_1} \right\rceil$ small jobs finishing after t.

Theorem 5. *There is a two-pass* $(1+\epsilon)$*-approximation streaming algorithm for* $P_m \mid$ *stream, share,* $e_{i,k} \ge e_0$ *for* $i \le m_1 \mid C_{\max}$ *such that it takes*

1. $O(1)$ *time to process each job in both the first pass and the second pass,*

2. $O\left(\tilde{n} + \frac{m^2}{\epsilon e_0 m_1} \cdot \log \frac{m}{\epsilon e_0}\right)$ *space, and*

3. $O(\tilde{n} + t_1(m, m_1, \epsilon))$ *time*

to return a schedule with a $(1+\epsilon)$ *approximation.*

2.3. Streaming Algorithm When an Estimate of P_{max} is Given

Algorithm 2 works when p_{max} is an input. In reality, however, p_{max} may not be obtained accurately without scanning all the jobs. In many practical scenarios, however, the estimate of p_{max} could be obtained based on priori knowledge. If this is the case, we can modify Algorithm 2 to get the approximate value of the optimal makespan as described below.

Let us assume that we are given p_{max}^E, an estimate of p_{max}, such that $p_{max} \leq p_{max}^E \leq \alpha * p_{max}$. And with this input, if we do the same as in Algorithm 2, we would partition the processing time range $(0, p_{max}^E]$ into $\gamma_0 + 2$ continuous intervals such as $IH_{-1} = (0, 2^{q_0'}]$, $IH_0 = (2^{q_0'}, 2^{q_0'+1}] \cdots$, $IH_{\gamma_0} = (2^{q_0'+\gamma_0}, 2^{q_0'+\gamma_0+1}]$, where $q_0' = \lceil \log p_{max}^E \rceil - \gamma_0 - 1$. Comparing with the intervals obtained with the exact p_{max}, we have $q_0 = \lceil \log p_{max} \rceil - \gamma_0 - 1 \geq \left\lceil \log \frac{p_{max}^E}{\alpha} \right\rceil - \gamma_0 - 1 \geq q_0' - \lceil \log \alpha \rceil$, so we need to further split IH_{-1} into $\lceil \log \alpha \rceil + 1$ intervals such as $IH_{-1} = (2^{q_0'-1}, 2^{q_0'}]$, $IH_{-2} = (2^{q_0'-2}, 2^{q_0'-1}]$, $\cdots$, $IH_{-1-\lceil \log \alpha \rceil} = (0, 2^{q_0' - \lceil \log \alpha \rceil}]$. Correspondingly, we have the job groups $H_{-1-\lceil \log \alpha \rceil}$, $\cdots$, H_{γ_0}. When we scan the jobs one by one, we can add the job to the corresponding group based on its processing time as we did in Algorithm 2. After all the jobs are scanned, we can get the exact p_{max} and only keep $\gamma_0 + 2$ groups as in Algorithm 2 and other parts of the algorithm will remain the same as Algorithm 2.

Corollary 6. *Let ϵ be a real number in $(0, 1)$, assume that an estimate of p_{max} is a given input, then for $P_m \mid stream,\ share,\ e_{i,k} \geq e_0 \ for\ i \leq m_1 \mid C_{\max}$ there is a one-pass streaming algorithm to find an approximate value of the* *optimal makespan by a $(1+\epsilon)$ factor and a two-pass streaming algorithm to find a schedule with this approximate value. The algorithms have $O(1)$ time for processing each job in the stream, $O\left(\tilde{n} + \frac{m^2}{\epsilon e_0 m_1} \cdot \log \frac{m}{\epsilon e_0}\right)$ space usage, and $O(\tilde{n} + t_1(m, m_1, \tilde{n}, \epsilon, e_0))$ running time.*

2.4. Streaming Algorithm When No Information about p_{max} is Given

Now we consider the case that not only the exact p_{max} but also the estimate of p_{max} is not given. In this case, γ_0 and N_0 can be calculated as before, q_0 cannot be determined until all jobs are read, thus we can not immediately determine which group a job belongs to as in Algorithm 2. To solve this problem, we need to modify Algorithm 2. When a job is read, we assign it to a group based on the information that we have so far, as more jobs are read later, we dynamically update the partition of the jobs so that we can maintain the following invariant: As in Algorithm 2, there are (γ_0+2) groups of jobs, H_{-1}, H_0, …, H_{γ_0}. If $p_j \leq 2^{q_0}$, then $j \in H_{-1}$; otherwise if $p_j \in (2^{q_0+k}, 2^{q_0+k+1}]$, $j \in H_k$.

To implement this efficiently, we use a B-tree (or other balanced search tree) to store information of those non-empty groups H_k for $k \geq 0$ and additionally we maintain the total processing time of jobs in H_{-1}, P_{-1}. Each group H_k in B-tree is represented as a quadruple $(\kappa_k, n_k, P_k, JS_k)$, where n_k, P_k, JS_k are defined the same as in Algorithm 2; $\kappa_k = 2^{q_0+k+1}$ is the key representing the processing time range $IH_k = (2^{q_0+k}, 2^{q_0+k+1}]$ of jobs in H_k for $0 \leq k \leq \gamma_0$. There are at most γ_0+1 quadruples in the B-tree. Initially, $q_0 = 0$ and the tree is empty. If a job j has the processing time $p_j \leq 2^{q_0+\gamma_0+1}$, we update the quadruple with the key $2^{\lceil \log p_j \rceil}$ or insert a new quadruple with the key $2^{\lceil \log p_j \rceil}$ if there is no such quadruple in the tree. If $p_j > 2^{q_0+\gamma_0+1}$, then let $q_0' = \lceil \log p_j \rceil - \gamma_0 - 1$, delete all the quadruples with the key less than or equal to $2^{q_0'}$ and update P_{-1} correspondingly, insert a new quadruple with the key $2^{q_0'+\gamma_0+1}$, and update $q_0 = q_0'$. A detailed description of the algorithm is given below.

Algorithm 3

Input:

- Parameters m, m_1, e_0, and ϵ
- The intervals $(0, t_{i,1}]$, $(t_{i,1}, t_{i,2}]$, $\ldots$ on M_i and their sharing ratios $e_{i,1}$, $e_{i,2}$, $\ldots$, respectively $(1 \le i \le m)$
- The jobs' processing time p_j (stream input), $1 \le j \le n$.

Output: An approximate value of the optimal makespan.

Steps:

```
1. Identify the set of large jobs, $JS$.
    (a) Choose $\gamma_0$ and $N_0$ as in Algorithm 1 and set $q_0 = 0$
    (b) Create an empty B-tree to store the quadruples
    (c) Read jobs one by one and do the following:
        i. Let $k = \lceil \log p_j \rceil - q_0 - 1$
        ii. If the quadruple with the key $2^{k+q_0+1}$ is already in the tree,
            update as follows:
                $n_k = n_k + 1$
                $P_k = P_k + p_j$
                if $n_k \le N_0$, then $JS_k = JS_k \cup \{j\}$,
                else reset $JS_k = \emptyset$
        iii. Else
```

```
if $p_j \leq 2^{q_0}$, let $P_{-1} = P_{-1} + p_j$
else if $p_j \leq 2^{q_0+\gamma_0+1}$
    insert a new quadruple $(2^{q_0+k+1}, 1, p_j, \{j\})$
else (in this case, $p_j > 2^{q_0+\gamma_0+1}$)
    $q_0' = \lceil \log p_j \rceil - \gamma_0 - 1$
    for each $(\kappa_k, n_k, P_k, JS_k)$ in the tree where $\kappa_k \leq 2^{q_0'}$
        $P_{-1} = P_{-1} + P_k$,
        delete the quadruple $(\kappa_k, n_k, P_k, JS_k)$ from the tree.
    insert quadruple $(2^{q_0'+\gamma_0+1}, 1, p_j, \{j\})$ in the tree;
    update $q_0 = q_0'$.
```

(d) $P = \sum_{-1 \leq k \leq \gamma_0} P_k$.

(e) Let k_L, $0 \leq k_L \leq \gamma_0$, be the largest index of the group such that $n_k \geq N_0$, if k_L doesn't exist, let $K_L = -1$. Let $JS = \cup_{k > k_L} JS_k$.

2. For each possible assignment of jobs in JS, find the time t associated with it as in Algorithm 1

3. Find the smallest t from previous step, and return the value $t + \left\lceil \frac{m-1}{m_1} \right\rceil \cdot \frac{1}{e_0} \cdot 2^{q_0+k_L+1}$.

Theorem 7. *Let ϵ be a real number in $(0,1)$. Then Algorithm 3 is a one-pass $(1+\epsilon)$-approximation streaming algorithm for $P_m \mid$ stream, share, $e_{i,k} \geq e_0$ for $i \leq m_1 \mid C_{\max}$ such that it takes*

1. $O(1)$ time to process each job in the stream,

2. $O\left(\tilde{n} + \frac{m^2}{\epsilon e_0 m_1} \cdot \log \frac{m}{\epsilon e_0}\right)$ space, and

3. $O(\tilde{n} + t_1(m, m_1, \tilde{n}, e_0, \epsilon))$ time *to find an approximation of C^*_{max} by a factor of $(1+\epsilon)$.*

Proof. The proof is similar to that of Theorem 6, so we only discuss the difference - the updating time for each job.

We use a B-tree (or other balanced search tree) to store the job groups where each group is represented as a quadruple. At any time, there are at most $\gamma_0 + 1 = O(\log \frac{m}{\epsilon e_0})$ quadruples/keys in tree.

For each job j, we perform a search operation, and maybe insert or delete. Since the number of keys/quadruples in the B-tree is $O(\gamma_0)$, all these operations can be done in $O(\log \gamma_0) = O(\log \log \frac{m}{\epsilon e_0})$ time, which is O(1) since m is a constant. □

Similarly, we can find the approximation schedule in two passes.

Theorem 8. *There is a two-pass $(1+\epsilon)$-approximation streaming algorithm for $P_m \mid$ stream, share $(e_{i,k} \geq e_0)$ for $i \leq m_1 \mid C_{\max}$ such that it takes*

1. *$O(1)$ time to process each job in the stream,*
2. *$O\left(\tilde{n} + \frac{m^2}{\epsilon e_0 m_1} \cdot \log \frac{m}{\epsilon e_0}\right)$ space, and*
3. *$O(\tilde{n} + t_1(m, m_1, \epsilon))$ time*

to find a $(1+\epsilon)$ approximation of the optimal makespan after receiving all jobs in the stream in the first pass, and $O(1)$ time for each job in the second pass to return a schedule for all jobs.

3. Conclusions

In this paper, we studied the multitasking scheduling problem with shared processing under the data stream model. There are multiple machines with sharing ratios varying from one interval to another, and we allow the sharing ratios on some machines have a constant lower bound. The goal is to minimize the makespan.

We designed the first streaming approximation schemes for the problem where the processing times of the jobs are input as a stream and no prior information about the number of jobs is required. This work not only provides an algorithmic big data solution for our studied scheduling problem, but also leads to one future research direction for the area of scheduling. The classical scheduling literature contains a large number of problems that remain to be studied under the data stream model presented here.

For our studied problems, it is also interesting to design streaming algorithms for other performance criteria including total completion time, maximum tardiness, and other machine environments such as uniform machines, flowshop, etc.

References

[1] I. Adiri and Z. Yehudai. Scheduling on machines with variable service rates. *Computers & Operations Research*, 14(4):289–297, 1987. doi: 10.1016/0305-0548(87)90066-9.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. doi: https://doi.org/10.1006/jcss.1997.1545.

[3] Kenneth R. Baker and Henry L. W. Nuttle. Sequencing independent jobs with a single resource. *Naval Research Logistics Quarterly*, 27:499–510, 1980.

[4] Richard Beigel and Bin Fu. A dense hierarchy of sublinear time approximation schemes for bin packing. *Electronic Colloquium on Computational Complexity*, 18:28, 2012.

[5] Graham Cormode and Pavel Veselý. Streaming algorithms for bin packing and vector scheduling. *Theory of Computing Systems*, 65:916–942, 2021. doi: 10.1007/s00224-020-10011-y.

[6] Decio Coviello, Andrea Ichino, and Nicola Persico. Time allocation and task juggling. *American Economic Review*, 104(2):609–23, 2014. doi: 10.1257/aer.104.2.609.

[7] Dariusz Dereniowski and Wieslaw Kubiak. Shared multi-processor scheduling. *European Journal of Operational Research*, 261:503–514, 2017.

[8] Dariusz Dereniowski and Wieslaw Kubiak. Shared processor scheduling of multiprocessor jobs. *European Journal of Operational Research*, 282: 464–477, 2020.

[9] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31 (2):182–209, 1985. doi: https://doi.org/10.1016/0022-0000(85)90041-8.

[10] Bin Fu, Yumei Huo, and Hairong Zhao. Multitasking scheduling with shared processing, 2022. Manuscript under review.

[11] Víctor M. González and Gloria Mark. Managing currents of work: Multi-tasking among multiple collaborations. In *European Conference of Computer-supported Cooperative Work*, 2005.

[12] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi: https://doi.org/10.1016/S0167-5060(08)70356-X.

[13] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[14] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[15] Nicholas G. Hall, Joseph Y.-T. Leung, and Chung lun Li. The effects of multitasking on operations scheduling. *Production and Operations Management*, 24:1248–1265, 2015.

[16] Nicholas G. Hall, Joseph Y.-T. Leung, and Chung lun Li. Multitasking via alternate and shared processing: Algorithms and complexity. *Discrete Applied Mathematics*, 208:41–58, 2016.

[17] Behzad Hezarkhani and Wieslaw Kubiak. Decentralized subcontractor scheduling with divisible jobs. *Journal of Scheduling*, 18:497–511, 2015.

[18] Tetsuji Hirayama and Masaaki Kijima. Single machine scheduling problem when the machine capacity varies stochastically. *Operations Research*, 40:376–383, 1992.

[19] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[20] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23:317–327, 1976.

[21] Hans Kellerer. Algorithms for multiprocessor scheduling with machine release times. *IIE Transactions*, 30:991–999, 1998.

[22] Chung-Yee Lee. Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30:53–61, 1991.

[23] J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980. doi: https://doi.org/10.1016/0304-3975(80)90061-4.

[24] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, aug 2005. doi: 10.1561/0400000002.

[25] Kevin J. O'Leary, David M. Liebovitz, and David W. Baker. How hospitalists spend their time: insights on efficiency and safety. *Journal of hospital medicine*, 1:88–93, 2006.

[26] David M. Sanbonmatsu, David L. Strayer, Nathan Medeiros-Ward, and Jason Watson. Who multi-tasks and why? multi-tasking ability, perceived multi-tasking ability, impulsivity, and sensation seeking. *PLoS ONE*, 8(1), 2013.

[27] John Sum and Kevin Ho. Analysis on the effect of multitasking. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 204–209, 2015. doi: 10.1109/SMC.2015.48.

[28] Vanessa Vega, Kristle I. McCracken, Clifford Nass, and Lumos Labs. Multitasking effects on visual working memory, working memory and executive control, 2008. Presentation at annual Meeting of the International Communication Association.

[29] Zhanguo Zhu, Feifeng Zheng, and Chengbin Chu. Multitasking scheduling problems with a rate-modifying activity. *International Journal of Production Research*, 55:296 – 312, 2017.