Multitasking scheduling with shared processing

Bin Fu | Yumei Huo | Hairong Zhao

1Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, Texas, USA
2Department of Computer Science, College of Staten Island, CUNY, Staten Island, New York, USA
3Department of Computer Science, Purdue University Northwest, Hammond, Indiana, USA

Correspondence
Yumei Huo, Department of Computer Science, College of Staten Island, CUNY, Staten Island, NY 10314, USA.
Email: yumei.huo@csi.cuny.edu

Handling Editor: Marc Posner

Abstract
Recently, the problem of multitasking scheduling has raised a lot of interest in the service industries. Hall et al. (Discrete Applied Mathematics, 2016) proposed a shared processing multitasking scheduling model which allows a team to continue to work on the primary tasks while processing the routinely scheduled activities as they occur. With a team being modeled as a single machine, the processing sharing of the machine is achieved by allocating a fraction of the processing capacity to routine jobs and the remaining fraction, which we denote as sharing ratio, to the primary jobs. In this paper, we generalize this model to parallel machines and allow the fraction of the processing capacity assigned to routine jobs to vary from one to another. The objectives are minimizing makespan and minimizing the total completion time of primary jobs. We show that for both objectives, there is no polynomial time approximation algorithm unless \( P=NP \) if the sharing ratios are arbitrary for all machines. Then we consider the problems where the sharing ratios on some machines have a constant lower bound. For each objective, we analyze the performance of the classical scheduling algorithms and their variations and then develop a polynomial time approximation scheme when the number of machines is a constant.

KEYWORDS
makespan, parallel machines, scheduling, shared processing, total completion time

1 | INTRODUCTION

The term “multitasking” was initially introduced from the mid-1960s in the area of computer systems to describe the sharing of computing processor capacity among a number of distinct jobs (see, e.g., Denning 1971). However, multitasking has been existing in the real life, as people naturally perform multiple tasks by switching from one task to another. For example, O’Leary et al. (2006) stated that 21% of hospital employees are involved in multiple activities during their work; and according to González and Mark (2005), workers from consulting companies usually engage in about 12 working spheres daily. Not surprisingly, many researchers studied the effect of multitasking systematically. For example, Legros et al. (2020) showed that by working on several different tasks over a given interval of time, a multitasker can improve overall productivity by reducing the time spent waiting between jobs, and it is more efficient for the worker to switch to a new task rather than idly waiting on a pending task. Some other studies (see examples in Coviello et al. 2014), however, have indicated that multitasking may disrupt work and may result in a significant loss of productivity. From a different perspective, the researchers in the area of scheduling study on multitasking (see, e.g., Hall et al. 2015, 2016; Sun and Ho 2015; Zhu et al. 2017) for the purpose of performance optimization.

Among these works, Hall et al. (2016) proposed a multitasking scheduling model where a team continuously works on its primary tasks but may allocate some of its team members (a.k.a. a percentage of its processing capacity) to process the routinely scheduled activities during some periods. The routinely scheduled activities are exemplified by maintenance work, administrative meetings, or meal breaks. An application of this model can be seen in call centers, where during each day’s two-hour lunch period, half of the working team takes...
a one-hour lunch break each hour to ensure that no customer calls are missed.

These routinely scheduled activities are usually crucial to the functionality of the overall system. In many practical situations, they are managed separately from the primary jobs. Some third-party companies such as Siteware, provide routine activities management for other companies. They help plan the routine activities for all the teamwork including the release times and duration of the routine jobs, the priority of the routine jobs, and the team members to whom the routine jobs can be assigned, etc. As described in the website of Siteware, usually routine jobs are assigned to the employees of the teams based on two criteria: professional skills and procedure priority. Professional skills criteria refers to handling over routine duties according to employees’ skills. Procedure priority refers to delegate routine activities that do not demand special attention and can be done by other people without major issues. The service provided by companies like Siteware motivates our model such that when the primary jobs are all available at time 0 for scheduling, the release times and duration of the routine jobs are all predetermined and are known beforehand and the processing capacity needed for each routine job is predetermined as well.

In these practical situations, due to the predetermined routine jobs, a working team is viewed as a machine which may have a sequence of time intervals with different processing capacities that are available for primary jobs.

In the scheduling model proposed by Hall et al. (2016), it is assumed that the machine capacity is the same for all routine jobs and there is only a single machine. In this paper, we generalize this model to parallel machine environment. Moreover, the machine capacity allocated to routine jobs can vary from one to another, instead of being same for all routine jobs. This is more practical considering there may exist different routine jobs with different priorities. Hence, the goal is to schedule the primary jobs on the machines subject to the varying capacity constraints so as to minimize the objectives.

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 requirement $p_j$ and can be processed by any one of the machines uninterruptedly. A primary job can only be processed on one machine, and cannot be preempted. We are also given a set of routine jobs that have to be processed during certain time intervals on certain machines. When a routine job is processed, a fraction of the machine capacity is given to the routine job and the remaining machine capacity is used for the continuation of the primary job. Thus depending on the routine jobs’ processing time intervals, the actual processing duration for each primary job may be longer than its processing requirement. If a primary job shares the processing with a routine job but completes before the routine job, then the next primary job will be immediately started and share the processing with this routine job. On the other hand, if the routine job shares the processing with a primary job but completes before the primary job and no other routine job is waiting for processing, this primary job will immediately have full capacity of the machine for processing. We use $k_i$ to denote the number of routine jobs that need to be processed on machine $M_i$, and $\bar{n}$ to denote the total number of the routine jobs, i.e. $\bar{n} = \sum_{i \in S} k_i$.

As we addressed earlier, routine job management, independent from the primary job scheduling, has predetermined the time interval during which a routine job is processed, the machine on which the routine job is processed, and the machine capacity that is assigned to a routine job. So we are only concerned with the schedule of the primary jobs when some fraction of the machine capacity has been assigned to the routine jobs during some time intervals on some machines. In this sense, we can view each machine consisting of intervals with full capacity alternating with intervals with a fraction of the machine’s capacity. We use the term “sharing ratio” to refer to the fraction of the capacity available to the primary jobs, which falls within the range of (0, 1]. Apparently, each machine $M_i (1 \leq i \leq m)$ 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.

In Figure 1(b), there are 5 primary jobs with the following processing requirements: 1, 2, 4, 1, and 5, respectively. Let’s go through the scheduling details for each job. Job 1 has a processing requirement of 1 and is scheduled at time 0 on machine $M_1$. With full capacity available during the interval (0, 1] on $M_1$, job 1 can be completed at time 1. Job 2 has a processing requirement of 2 and is scheduled at time 1 on machine $M_1$. Machine $M_1$ has a sharing ratio of $\frac{1}{2}$ during the interval (1, 3], which means only half of the machine’s capacity is available for job 2. Therefore, the length of job 2 that can be processed in this interval is $2 \cdot \frac{1}{2} = 1$. The remaining part of job 2 is continued to be processed during the interval (3, 4] with a sharing ratio of 1. Job 3 is scheduled during the interval (4, 7] with a sharing ratio of $\frac{3}{2}$ and the interval (7, 9] with a sharing ratio of 1. Job 4 has a processing requirement of 1 and is scheduled on $M_2$. It is processed in the interval (0, 2] with a sharing ratio of $\frac{1}{2}$. After job 4 completes at time 2, job 5 is first processed during the interval (2, 4] with a sharing ratio of $\frac{1}{2}$. It is then continued to be processed during the interval (4, 8] with a sharing ratio of 1.

In this paper, we focus on two objectives: minimizing the makespan and minimizing the total completion time of the primary jobs. 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
1.2 | Literature review

The multitasking model studied in this paper was initially proposed by Hall et al. (2016). The authors only considered the single machine environment where during any interval the machine is either fully available for primary jobs, or is shared by primary jobs and routine jobs with the processing capacity of \( e \) and \( 1 - e \), respectively. The goal is to schedule the primary jobs to optimize some objective functions. The authors showed that for the objective of makespan, any schedule with no unnecessary idle time is optimal; for the objective of total completion time, the optimal schedule can be obtained by scheduling the jobs in non-decreasing order of the processing requirement; for the objective of total weighted completion time, the problem is unary NP-Hard; and for the objectives of maximum lateness and the number of late jobs, the authors presented the optimal algorithms, respectively.

For the related work, Baker and Nuttle (1980) studied the problems of scheduling \( n \) jobs on a single machine subject to the constraint that the availability of the machine varies over time. The motivation for this machine environment comes from the situation in which processing requirements are stated in terms of labor-hours and labor availability varies over time. The example can be found in the applications of rotating Saturday shifts, where the company only maintains a fraction, for example 33\%, of the workforce. In the paper, the authors showed that many existing algorithms for the classical model can be used to solve the corresponding variable machine availability problems with little or no modification. Adiri and Yehudai (1987) studied similar scheduling problems under the concept of service rate. There are one or more parallel identical machines that can be used to process the jobs at certain service rates. The constraint is that if a job is being processed, the service rate of a machine remains constant and it can be changed only when the job is completed.

So far there are no results about the problems studied in this paper. Note that if \( e_{i,k} = 1 \) for all time intervals, that is, there are no routine jobs, our problems become the classical parallel machine scheduling problems \( P_m|C_{\text{max}} \) and \( P_m||\sum C_j \). The problem \( P_m||\sum C_j \) can be solved optimally using SPT (Shortest Processing Time (requirement) First) rule, which schedules the next shortest job to the earliest available machine. The problem \( P_m|C_{\text{max}} \) is an NP-Hard problem and some approximation algorithms have been designed for it. Graham (1969) showed that LS (List Scheduling) rule generates a schedule with an approximation ratio of \( 2 - \frac{1}{m} \). The LS rule schedules the jobs one by one in the given ordered list. Each job is assigned in turn to a machine which is available at the earliest time. If the given list is in non-increasing order of the processing requirements of the jobs, the list schedule rule is called LPT (Longest Processing Time (requirement) First) rule. Graham (1969) showed that the LPT rule generates a schedule with an approximation ratio of \( \left( \frac{4}{3} - \frac{1}{3m} \right) \). Hochbaum and Shmoys (1987) designed a PTAS for this problem in 1987. Additionally, Horowitz and Sahni (1976) developed an FPTAS in 1976 for the case when the number of machines \( m \) is fixed.

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 problems reduce to the problems of parallel machine scheduling with availability constraint where jobs can be resumed on after being interrupted: \( P_m|r - a|C_{\text{max}} \) and \( P_m|r - a|\sum C_j \). The problem \( P_m|r - a|C_{\text{max}} \) is NP-hard and approximation algorithms are developed by Kellerer (1998) and Lee Lee (1991). Lee and Liman (1993) showed that the problem \( P_m|r - a|\sum C_j \) is NP-hard when \( m = 2 \) with one machine continuously available while the other machine becomes unavailable after some finite time. They developed a pseudo-polynomial algorithm to solve the problem and also showed that the SPT rule, with some modifications, leads to a tight relative error of \( \frac{1}{2} \) for the problem. More results can be found in the survey paper by Ma et al. (2010) and references therein.
1.3 New contribution

In this paper, we generalize the shared processing multitasking model proposed by Hall et al. (2016) to parallel machine environment and allow the processing capacity to be different for different routine jobs.

We show that there is no approximation algorithm for the general problem \( P_m, e_{i,k} || C_{\text{max}} \) and \( P_m, e_{i,k} || \sum C_j \) unless \( P = NP \). Then we study the case that, on some machines, the machine capacity for primary jobs may have a constant lower bound. This scenario is also justifiable in some real life applications where a minimum number of members from the team are needed to guarantee the continuous customer service and technical support at any time.

For the objective of makespan, we analyze the performance of the LS (List Scheduling) rule and LS-ECT (List Scheduling - Earliest Completion Time) rule for the problems \( P_m, e_{i,k} \geq e_0 \| C_{\text{max}} \) and \( P_m, e_{i,k} \geq e_0 \| C_{\text{max}} \). We then develop an approximation scheme for the problem \( P_m, e_{i,k} \geq e_0 \| C_{\text{max}} \) whose running time is linear when the number of machines is a constant.

For total completion time minimization problem \( P_m, e_{i,k} \geq e_0 \| \sum C_j \), we show that although SPT (Smallest Processing Time (requirement) - Earliest Completion Time) rule can perform arbitrarily bad, SPT-ECT (Shortest Processing Time (requirement) - Earliest Completion Time) rule is a \( \left\lfloor \frac{m}{m_1} \right\rfloor \cdot \frac{1}{e_0} \)-approximation algorithm. We then develop an approximation scheme for the problem \( P_m, e_{i,k} \geq e_0 \| \sum C_j \) whose running time is polynomial when the number of machines is a constant.

The paper is organized as follows. In Section 2, we show that for both objectives of makespan and total completion time, the problems are inapproximable if the processor sharing intervals and the sharing ratios in these intervals are arbitrary. Then we study problems where the sharing ratios on some machines have a constant lower bound. We present the results for the objective of makespan in Section 3 and the results for the objective of total completion time in Section 4, respectively. Finally, we draw the concluding remarks in Section 5.

2 HARDNESS OF APPROXIMATION

In this section, we show that if the processor sharing intervals and the sharing ratios in these intervals are arbitrary, it is impossible to approximate either of the objectives within any constant factor, or even polynomial.

**Theorem 1.** Let \( \mathbb{N} \) be the set of natural numbers, and \( n \) be the number of the jobs. Let \( f(n) : \mathbb{N} \to \mathbb{N} \) be an arbitrary function such that \( f(n) > 1 \). There is no polynomial time \( f(n) \)-approximation algorithm for \( P_m, e_{i,k} || C_{\text{max}} \) even if \( m = 2 \) unless \( P = NP \).

Proof. We prove the inapproximability by reducing from the partition problem.

Partition Problem: Given a set of positive integers \( \{a_1, \ldots, a_n\} \) where \( A = \sum_{i=1}^{n} a_i \) is even, find if the set can be partitioned into two sets with equal sum \( \frac{1}{2}A \).

Given an instance of partition problem, we can reduce it to an instance of our scheduling problem as follows: There are two machines and \( n \) primary jobs. For each primary job \( j \), \( p_j = a_j \). There are two routine jobs, one for each machine, both of which are processed during the interval \( \left\lfloor \frac{A}{2} + f(n) \cdot A \right\rfloor \) with a sharing ratio \( e = \frac{1}{f(n)A} \).

We will show that if there is an \( f(n) \)-approximation algorithm for the scheduling problem, then the algorithm returns a schedule with makespan at most \( \frac{A}{2} \cdot f(n) \) for the constructed scheduling problem instance if and only if there is a partition for the given partition instance.

Apparently, the makespan of any schedule of the primary jobs is at least \( \frac{A}{2} \). Also if a job cannot finish at \( \frac{A}{2} \), it will then take at least \( f(n)A \) additional time units due to the processing sharing. Thus the makespan of any schedule is either exactly \( \frac{A}{2} \) or at least \( \frac{A}{2} + f(n)A = \frac{A}{2}(1 + 2f(n)) \). Therefore, if the approximation algorithm returns a schedule whose makespan is at most \( \frac{A}{2} \cdot f(n) \), then the makespan of the schedule must be exactly \( \frac{A}{2} \). This implies that there is a partition to the set of integers.

On the other hand, if there is a partition, i.e., the numbers can be partitioned into two sets, each having a sum exactly \( \frac{A}{2} \), then the optimal schedule will schedule the corresponding jobs in each set to a machine and its makespan is exactly \( \frac{A}{2} \). Then the \( f(n) \)-approximation algorithm must return a schedule whose makespan is at most \( \frac{A}{2} \cdot f(n) \).

The above analysis shows that the partition problem has a solution if and only if the algorithm returns a schedule with makespan at most \( \frac{A}{2} \cdot f(n) \) for the corresponding scheduling instance. Since the partition problem is NP-hard, unless \( P=NP \), there is no such approximation algorithm.

It is well known that the classical problem \( P_m || \sum C_j \) can be solved using SPT rule and it becomes inapproximable when the machines have unavailable periods [see Lee and Liman (1993)]. By our definition, this is the case that the sharing ratio is in the set \( \{0, 1\} \). In the following, we show that the problem \( P_m, e_{i,k} || \sum C_j \) does not admit any approximation algorithm even if there are only two machines and the sharing ratio is always positive.

**Theorem 2.** Let \( \mathbb{N} \) be the set of natural numbers, and \( n \) be the number of the jobs. Let \( f(n) : \mathbb{N} \to \mathbb{N} \) be an arbitrary function such that \( f(n) > 1 \). There is no polynomial time \( f(n) \)-approximation algorithm for \( P_m, e_{i,k} || C_{\text{max}} \) even if \( m = 2 \) unless \( P = NP \).
\( \mathbb{N} \) be an arbitrary function such that \( f(n) > 1 \). There is no polynomial time \( f(n) \)-approximation algorithm for \( P_m, e_{i,k} \) \( \sum C_j \) even if \( m = 2 \) unless \( P = NP \).

**Proof.** We reduce from the partition problem. In the partition problem, we are given a set of positive integers \( \{a_1, a_2, \ldots, a_n\} \), where \( A = \sum_{i=1}^{n} a_i \). The problem is “can the set be partitioned into two subsets with equal sum \( \frac{A}{2} \)?” We construct an instance of the scheduling problem \( P_m, e_{i,k} \) \( \sum C_j \) as follows: there are two machines and \( n \) primary jobs. Job \( j \) has processing requirement \( p_j = a_j \). There are two routine jobs, one for each machine, and both are processed during the interval \( [\frac{A}{2}, \frac{A}{2} + nf(n) \cdot A] \) with a sharing ratio of \( e = \frac{1}{nf(n) \cdot A} \).

It is easy to see that there is a partition of the set if and only if there is a schedule where all jobs could finish at or before \( \frac{A}{2} \), which means the total completion time of all jobs is at most \( \frac{A}{2} \). We can show further, the latter problem can be answered if there is an \( f(n) \)-approximation algorithm.

Suppose there is a schedule in which the total completion time of all jobs is at most \( \frac{A}{2} \), then a \( f(n) \)-approximation algorithm would return a schedule with the total completion time at most \( \frac{A}{2} \cdot f(n) \). This implies that all jobs must finish at or before \( \frac{A}{2} \) because if a job finishes after \( \frac{A}{2} \), its completion time will be at least \( \frac{A}{2} + \frac{1}{e} = \frac{A}{2} + nf(n) \cdot A \), and thus the total completion time is greater than \( \frac{A}{2} \cdot f(n) \). Hence, there exists a schedule whose total completion time at most \( \frac{A}{2} \) if and only if the \( f(n) \)-approximation algorithm returns a schedule such that all jobs finish by \( \frac{A}{2} \). Consequently, we can solve the partition problem, which is impossible unless \( P = NP \).

Given the inapproximability results from Theorem 1 and Theorem 2, from now on, we will focus on the problems such that the sharing ratios on some machines are greater than or equal to a constant \( e_0 \). In the next two sections, we will present the related results for the objectives of makespan and total completion time, respectively.

### 3 | MAKESPAN MINIMIZATION

In this section, we study the makespan minimization problems \( P_m, e_{i,k} \geq e_0 \| C_{max} \) and \( P_m, e_{i \leq m, k} \geq e_0 \| C_{max} \). We first give some preliminary results. Then for a fixed \( e_0, 0 < e_0 < 1 \), we analyze the performance of the LS (List Scheduling) rule and LS-ECT (List Scheduling - Earliest Completion Time) rule. We then develop an approximation scheme for the problem \( P_m, e_{i \leq m, k} \geq e_0 \| C_{max} \) whose running time is linear when the number of machines is a constant.

#### 3.1 | Preliminary results for \( P_m, e_{i,k} \geq e_0 \| C_{max} \)

For problem \( P_m, e_{i,k} \geq e_0 \| C_{max} \), the sharing ratio is bounded below by a constant \( e_0 \) for all intervals, i.e. \( e_{i,k} \geq e_0 \) on all machines \( M_i \), \( 1 \leq i \leq m \). Let \( I \) be an instance for \( P_m, e_{i,k} \geq e_0 \| C_{max} \). Based on \( I \), we create a corresponding instance \( I' \) such that \( I' \) has the same set of primary jobs as \( I \) but does not have any routine job. Therefore, for instance \( I' \), each machine has full capacity available all the time for processing primary jobs. For each pair of corresponding instances \( I \) and \( I' \), let \( S' \) be a schedule for \( I' \), and we can construct a corresponding schedule \( S \) for \( I \) as follows: for each machine \( M_i (1 \leq i \leq m) \), we obtain the sequence of jobs that are scheduled on \( M_i \) in \( S' \), and schedule them in the same order as in \( S' \) to machine \( M_i \) in \( S \) where the routine jobs have been pre-scheduled during specific intervals with given fractions of machine capacity. As a result, the primary jobs may be completed later in \( S \) compared with \( S' \) due to the reduced machine capacity, but not more than a factor of \( \frac{1}{e_0} \) where \( e_0 \) is the lower bound of the sharing ratios. This implies that the makespan of \( S \) is at most \( \frac{1}{e_0} \) times that of \( S' \). Since the optimal makespan for \( I \) must be greater than or equal to that of \( I' \), we get the following observation.

**Observation 3.** An \( \alpha \)-approximation algorithm for \( P_m || C_{max} \) is an \( \frac{\alpha}{e_0} \)-approximation algorithm for \( P_m, e_{i,k} \geq e_0 || C_{max} \).

Observation 3 and the existing literature (Graham 1969; Hochbaum and Shmoys 1987; Horowitz and Sahni 1976) imply the following results for \( P_m, e_{i,k} \geq e_0 || C_{max} \):

1. LS rule gives a \( \left( 2 - \frac{1}{m} \right) \frac{1}{e_0} \) approximation.
2. LPT rule gives a \( \left( \frac{4}{3} - \frac{1}{3m} \right) \frac{1}{e_0} \) approximation.
3. There is a \( \frac{1+e}{e_0} \) approximation algorithm whose running time is \( O(n^{2m} \log_2 \frac{1}{e_0}) \), where \( k = \left\lfloor \log_{1+e} \frac{1}{e_0} \right\rfloor \).
4. There is a \( \frac{1+e}{e_0} \) approximation algorithm whose running time is \( O(n^2 / (e)k^{n-1}) \).

#### 3.2 | PERFORMANCE OF CLASSICAL SCHEDULING ALGORITHMS

In the following, we will give a tighter bound for LS rule and its variant.

3.2.1 | LS Rule for \( P_m, e_{i,k} \geq e_0 || C_{max} \)

For an instance of \( P_m, e_{i,k} \geq e_0 || C_{max} \), let \( C_{max}(LS) \) and \( C_{max}^* \) be the makespan of an arbitrary list schedule and the optimal schedule, respectively. Then we have the following theorem.
For $P_m, e_{i,k} \geq e_0 || C_{\text{max}}$, the LS rule generates a schedule with the approximation ratio of $(1 + \frac{1}{e_0} - \frac{1}{m})$ in $O(n \log m + n)$ time.

Proof. Without loss of generality, we assume $C_{\text{max}}(LS) = C_j$ for some job $j$. Assume that job $j$ starts at time $t$ in the schedule generated by the LS rule. This implies that there is no idle time before time $t$ in the schedule, and thus we can obtain a lower bound on the minimum makespan $C_{\text{max}}$, i.e., $C_{\text{max}} \geq t + \frac{p_j}{e_0}$. On the other hand, we can derive an upper bound on the makespan of the list schedule as:

$$C_{\text{max}}(LS) = C_j \leq t + \frac{p_j}{e_0} = (t + \frac{p_j}{m}) + p_j \cdot \left(1 - \frac{1}{e_0} - \frac{1}{m}\right).$$

Now we analyze the running time. Given a list of jobs, the list scheduling can be implemented as follows. For a partial schedule, we maintain the completion time of the last job $f_i$, $1 \leq i \leq m$, on all machines using a min-heap. The next job $j$ in the list will be assigned to the machine with minimum $f_i$, which can be found and updated in $O(k + \log m)$ time, where $k$ is the number of intervals during which the job $j$ is scheduled. The total running time to assign all $n$ jobs will be $O(n \log m + n)$.

Theorem 4 implies that when $e_0$ is close to 1, the ratio of LS is close to $(2 - \frac{2}{m})$ which agrees with the tight bound of LS rule for the classical scheduling problem $P || C_{\text{max}}$.

On the other hand for a specific $e_0$, the worst case example for LS that we can find is as follows: there are $n = (m-1)n + 1$ jobs, where one job has a processing requirement of $m$ units, while the remaining jobs have a processing requirement of 1 unit each. There is a routine job that has been scheduled during the time interval $(m, m + m/e_0]$ on the first machine, leaving a machine capacity of $e_0$ for primary jobs on this machine during this interval. In the optimal schedule, the large job with a processing requirement of $m$ is scheduled to a single machine, while the remaining $m - 1$ machines each has $m$ jobs, resulting in a makespan of $m$. In contrast, the LS may schedule $m - 1$ small jobs on each of the $m$ machines, with the large job scheduled on the first machine after the $m - 1$ small jobs. The makespan of the LS schedule is given by $m + \frac{m-1}{e_0}$, resulting an approximation of $(1 + \frac{1}{e_0} - \frac{1}{e_0 m})$. The difference between this ratio and the bound in Theorem 4 leaves room for further exploration in future work.

If we relax the constraint on sharing ratios from the problem $P_m, e_{i,k} \geq e_0 || C_{\text{max}}$ so that only the intervals on the first $m_1$ ($m_1 < m$) machines have bounded sharing ratios, i.e., $P_m, e_{i \leq m_1,k} \geq e_0 || C_{\text{max}}$, LS rule could perform very badly. Consider the example of 2 machines, the first machine has a sharing interval $(0, \infty)$ with a sharing ratio of $e_0$, the second machine has a sharing interval $(0, \infty)$ with a sharing ratio of $x << e_0$, and we have two jobs, each of which has a processing requirement of 1. The LS schedule will schedule one job on each machine and the makespan will be $1/x$, but the optimal schedule will schedule both jobs on the first machine and has a makespan of $\frac{1}{e_0}$. The approximation ratio of LS is $\frac{e_0}{2x}$, which will be arbitrarily large as $x$ gets close to 0. To tackle this problem, we consider a modified list scheduling rule, LS-ECT, which is presented in the next subsection.

3.2.2 LS-ECT Rule for $P_m, e_{i \leq m_1,k} \geq e_0 || C_{\text{max}}$ and $P_m, e_{i,k} \geq e_0 || C_{\text{max}}$

In this subsection, we analyze the performance of the modified list scheduling rule, LS-ECT, which schedules the jobs such that when the next job $j$ in the list needs to be scheduled, instead of scheduling it to the machine so that it can start as early as possible, schedule it on the machine so that it can complete the earliest. Before we do further analysis, we first give a claim that we will use frequently later.

Claim 5. For $P_m, e_{i \leq m_1,k} \geq e_0 || C_{\text{max}}$, LS-ECT algorithm produces a schedule in which at most $(m-1)$ jobs finish after time $C_{\text{max}}^*$ where $C_{\text{max}}^*$ is the optimal makespan for the problem.

Proof. Suppose, on the contrary, that there are $m$ jobs finishing after $C_{\text{max}}^*$. Let $x$ be the number of machines each have at least one job completed after $C_{\text{max}}^*$. If $x = m$, we would have all machines busy after $C_{\text{max}}^*$, which is impossible. Now we consider the case $x < m$. Among the $m$ jobs finishing after $C_{\text{max}}^*$, we remove $(m-x)$ jobs from the end so that the last job on these $x$ machines still completes after $C_{\text{max}}^*$. We then reschedule each of the removed $(m-x)$ jobs to one of the remaining $(m-x)$ machines at the end.

According to the LS-ECT rule, when these $(m-x)$ jobs were initially scheduled, they were placed on machines so that they can be completed at the earliest time. Therefore, if these $(m-x)$ jobs are rescheduled to any of the remaining $(m-x)$ machines, their completion times will not decrease. Consequently, their completion times will still remain greater than $C_{\text{max}}^*$. Thus, all machines are busy after $C_{\text{max}}^*$, which contradicts the fact that $C_{\text{max}}^*$ is the optimal makespan. This concludes the proof of the claim.

Figure 2 illustrates the proof of Claim 5. In the example, $m = 5$. There are 5 jobs that finish after $C_{\text{max}}^*$ and they are scheduled on the first 2 machines. Among these 5 jobs, if we reschedule jobs 2, 4, 5 to the last three machines, respectively, their completion times would not decrease and thus all the
In the following, we consider the performance of LS-ECT (LS-ECT) to denote the makespan of the schedule produced by LS-ECT.

**Theorem 6.** For \( P_m, e_i \geq e_0 \), LS-ECT generates a schedule with an approximation ratio of \( \left( 1 + \frac{1}{e_0} \cdot \left( \frac{m-1}{m} \right) + 1 \right) \) in \( O(\bar{n} + n(m + m \log(\bar{n}/m))) \) time.

**Proof.** Let \( I \) be the smallest instance for which LS-ECT has the worst performance. Assume the jobs are ordered 1, 2, \ldots, \( n \), then we must have \( C_{\text{max}}(\text{LS-ECT}) = C_p \). Suppose not, then we can remove job \( n \) to get a new instance. For the new instance, the performance ratio of LS-ECT will be the same or worse because the makespan of LS-ECT schedule will be the same, but the optimal makespan may be smaller, which contradicts the fact that \( I \) is the smallest instance for which LS-ECT has the worst performance.

Let \( S \) be the schedule obtained for \( I \) using LS-ECT rule where \( f_i, 1 \leq i \leq m \), is the completion time of the last job on machine \( M_i \) in \( S \). It is easy to see that there exists at least one \( f_i \) such that \( f_i \leq C_{\text{max}} \).

If \( f_i \leq C_{\text{max}} \) for some \( i \leq m_1 \), then we reschedule job \( n \) to \( M_i \), its new completion time will be at most \( f_i + p_n/e_0 \leq \left( 1 + \frac{1}{e_0} \right) C_{\text{max}} \).

By LS-ECT rule, this will not be better than its original completion time. Thus in this case we have \( C_{\text{max}}(\text{LS-ECT}) = C_p \leq \left( 1 + \frac{1}{e_0} \right) C_{\text{max}} \). See Figure 3 for an illustration.

Otherwise, we have \( f_i > C_{\text{max}} \) for all \( 1 \leq i \leq m_1 \), and for some machine \( M_i', i' > m_1, f_i < C_{\text{max}} \). By Claim 5, there can be at most \((m - 1)\) jobs complete after \( C_{\text{max}} \), which implies that at most \((m - 1)\) jobs complete after \( C_{\text{max}} \) on the first \( m_1 \) machines. By pigeon hole principle, among the first \( m_1 \) machines, there exists a machine on which there are at most \[ \lceil (m - 1)/m_1 \rceil \] jobs finishing after \( C_{\text{max}} \). Without loss of generality, suppose \( n \) is not scheduled on this machine, then moving job \( n \) to this machine will not decrease its completion time. Let \( p_{\text{max}} = \max_{1 \leq i \leq n} p_i \), then we have

\[
C_{\text{max}}(\text{LS-ECT}) = C_p \leq C_{\text{max}} + \frac{m - 1}{m_1} \frac{p_{\text{max}}}{e_0} + \frac{p_n}{e_0} \leq \left( 1 + \left( \frac{m - 1}{m_1} + 1 \right) \cdot \frac{1}{e_0} \right) C_{\text{max}}.
\]

Now we analyze the running time of LS-ECT rule. Considering a partial schedule where jobs 1, 2, \ldots, \( j - 1 \) have been scheduled, we maintain a pair \( (P_i, f_i) \) for each machine \( M_i (1 \leq i \leq m) \), where \( P_i \) is the total processing requirement of the jobs that have been assigned to \( M_i \), and \( f_i \) is the completion time of the last job on \( M_i \). Moreover, for each interval on machine \( M_i \), \( I_{i,1} = (0, t_{i,1}], I_{i,2} = (t_{i,1}, t_{i,2}], \ldots \), we maintain a quadruple \( (t_{i,k-1}, t_{i,k}, e_{i,k}, A_i(t_{i,k})) \) where \( e_{i,k} \) is the sharing ratio of the interval \( t_{i,k-1}, t_{i,k} \) and \( A_i(t_{i,k}) \) is the total job processing requirement that can be scheduled before \( t_{i,k} \). For convenience of scheduling, we pre-calculate \( A_i(t_{i,k}) \) for every \( t_{i,k} \) on machine \( M_i \). Note that \( A_i(t_{i,k}) = A_i(t_{i,k-1}) + e_{i,k}(t_{i,k} - t_{i,k-1}) \). Assuming the intervals are given in sorted order, the calculation can be done in \( O(k_i) \) time for machine \( M_i \) where \( k_i \) is the number of shared intervals on \( M_i \), and in \( O(\bar{n}) \) time for all \( m \) machines.

To assign a job \( j \) to this partial schedule, for each machine \( M_i \), we can use binary search on the continuous intervals \( I_{i,1}, I_{i,2}, \ldots \), to find the interval \( t_{i,k-1}, t_{i,k} \) such that \( A_i(t_{i,k-1}) < P_j + p_j \leq A_i(t_{i,k}) \). Then job \( j \)'s completion time on machine \( M_i \) can be calculated as \( t = t_{i,k-1} + \frac{P_j + p_j - A_i(t_{i,k-1})}{e_{i,k}} \).

After all machines are considered, we assign job \( j \) to the machine so it completes the earliest. In total, it takes \( O(\sum_{i=1}^{m} \log k_i) + m \) time to assign a job to the machine so it completes the earliest.

So the overall time for scheduling \( n \) jobs using LS-ECT is \( O(\bar{n} + n(m + \sum_{i=1}^{m} \log k_i)) \). Note that \( \sum k_i = O(\bar{n}) \), so the running time is bounded by \( O(\bar{n} + n(m + m \log(\bar{n}/m))) \).

Theorem 6 implies that if \( m_1 = m - 1 \), the approximation ratio becomes \( 1 + \frac{1}{e_0} \). In the following we will show that in this case, the approximation ratio is actually bounded by \( \left( 1 + \frac{1}{e_0} \right)^2 \). We will also show that for the case of \( m_1 = m \),
i.e., $P_m, e_{i,k} \geq e_0 ||C_{max}$, LS-ECT provides an approximation of $(1 + \frac{1}{e_0} - \frac{1}{m})$ which is the same as the LS rule.

**Theorem 7.** For $P_m, e_{i,k} \geq e_0 ||C_{max}$,

$$\frac{C_{max}(LS-ECT)}{C_{max}^{*}} \leq 1 + \frac{1}{e_0},$$

and for $P_m, e_{i,k} \geq e_0 ||C_{max}$,

$$\frac{C_{max}(LS-ECT)}{C_{max}^{*}} \leq 1 + \frac{1}{e_0} - \frac{1}{m}.$$

**Proof.** We first consider the problem $P_m, e_{i,k} \geq e_0 ||C_{max}$. Let $I$ be the smallest instance for which LS-ECT has the worst performance. Assume the jobs are ordered $1, 2, \ldots, n$, then we must have $C_{max}(LS-ECT) = C_n$.

Let $S$ be the schedule obtained for $I$ using LS-ECT where $f_i, 1 \leq i \leq m$, is the completion time of the last job on machine $M_i$ in $S$. It is easy to see that there exists at least one $f_i$ such that $f_i \leq C_{max}$.

If $i \leq m - 1$, then as in the proof of Theorem 6, we have $C_{max}(LS-ECT) = C_n \leq (1 + \frac{1}{e_0}) C_m$.

Otherwise, there is no $i$ such that $i \leq m - 1$ and $f_i \leq C_{max}$. This implies (1) on each of the first $(m - 1)$ machines, at least one job finishes after $C_m$, (2) $f_m < C_{max}$, and (3) job $n$ must be scheduled on one of the first $(m - 1)$ machines.

By Claim 5, there are at most $(m - 1)$ jobs that complete after $C_{max}$ on the first $m - 1$ machines. Combining with (1), there is exactly one job that completes after $C_{max}$ for each $M_i, i \leq m - 1$. Therefore, $n$ must be the only job that finishes after $C_{max}$ on the machine it is scheduled, and it starts at or before $C_{max}$. So we have $C_{max}(LS-ECT) = C_n \leq C_{max} + p_n/e_0 \leq \left(1 + \frac{1}{e_0}\right) C_{max}$.

Next we consider the performance of LS-ECT when $e_{i,k} \geq e_0$ for all the intervals on all $m$ machines, i.e. $P_m, e_{i,k} \geq e_0 ||C_{max}$. With the same argument as above, we can assume that $C_{max}(LS-ECT) = C_n$. Let $f_i = \min_{k=1}^{m} \{f_k\}$. Then there will be no idle time before $f_i$, and thus we must have $C_{max} \geq f_i + p_n/e_0$. On the other hand, if we reschedule job $n$ to machine $i$, its new completion time is at most $f_i + p_n/e_0$ and it will not be less than its original completion time $C_n$. Thus,

$$C_{max}(LS-ECT) = C_n \leq f_i + \frac{p_n}{e_0} = f_i \left(1 + \frac{p_n}{m} \right) + \frac{p_n}{e_0} - \frac{p_n}{m} \leq \left(1 + \frac{1}{e_0} - \frac{1}{m}\right) C_{max}.$$ 

A natural question follows from Theorem 7 is if the performance of LS-ECT for $P_m, e_{i,k} \geq e_0 ||C_{max}$ is still $(1 + \frac{1}{e_0})$ when $m_1 = m - 2$. We show by an example that this is unfortunately not the case. Let $m = 3$ and $m_1 = 1$, and $x$ is an integer greater than 1. The first machine has a sharing ratio of $e_0$ during interval $(x + 2, \infty)$, the other 2 machines have sharing ratio $\frac{2}{3x}$ during interval $(x, \infty)$, and all other intervals have sharing ratio 1; there are 5 jobs whose processing requirements are $x, 1, 1, x, x$. If we use this list for LS-ECT, the first machine has three large jobs, and the other two machines each has one small job, and the makespan is $x + 2 \frac{2x-2}{e_0}$; however in the optimal schedule, one large job and the two small jobs are scheduled on the first machine, and the other two large jobs are scheduled on the second and third machines, respectively. 

The makespan of the optimal schedule is $(x + 2)$. The performance ratio of LS-ECT approaches $\left(1 + \frac{x}{e_0}\right)$ as $x$ increases.

### 3.3 Approximation scheme for $P_m, e_{i,k} \geq e_0 ||C_{max}$

In this section, we develop an approximation scheme for $P_m, e_{i,k} \geq e_0 ||C_{max}$. Our technique is quite standard. We first partition the jobs into two groups, one for large jobs and the other for small jobs. Then we schedule the large jobs using enumeration and schedule the small jobs using LS-ECT. Let $d$ be the number of large jobs which depends on the error ratio of the approximation scheme and will be specified later. Our algorithm is formally presented as follows (Algorithm 1).
Algorithm 1.

**Input:**
- Parameters $m, n, m_1,$ and $\epsilon_0$
- The intervals $(0, t_{1,1}], (t_{1,1}, t_{1,2}], \ldots$ on machine $M_i, 1 \leq i \leq m$, and their sharing ratios $e_{i1}, e_{i2}, \ldots$, respectively
- The processing requirement for all jobs, $p_j, 1 \leq j \leq n$
- Integer parameter $d$ that determines the accuracy of approximation

**Output:** a schedule of the $n$ jobs

**Steps:**
1. Find the $d$ largest jobs
2. For each possible assignment of the large jobs, schedule the remaining jobs using LS-ECT
3. Return the schedule $S$ obtained from previous step that has the minimum makespan

For ease of analysis, we first analyze the performance of Algorithm 3.3 for $m_1 = m$, i.e., $P_m, e_{i,k} \geq \epsilon_0 (1 - |C_{\text{max}}|$ for the general case $m_1 \leq m$, i.e., $P_m, e_{i\leq m_1,k} \geq \epsilon_0 |C_{\text{max}}$. Let $S^*$ be an optimal schedule and $C_{\text{max}}^*$ be the makespan of $S^*$.

**Lemma 8.** For an instance of $P_m, e_{i,k} \geq \epsilon_0 |C_{\text{max}}$, Algorithm 3.3 returns a schedule with the makespan at most $(1 + \frac{m}{d \epsilon_0}) \cdot C_{\text{max}}^*$.

**Proof.** Let $p_d$ be the processing requirement of the $d$-th largest job. Then we must have $C_{\text{max}}^* \geq \frac{p_d}{m}$. Thus, $p_d \leq \frac{m}{d} C_{\text{max}}^*$. Let $S'$ be the schedule obtained from step 2 of Algorithm 3.3 that has the same large job assignment as $S^*$. Let $j$ be the job such that $C_j(S') = C_{\text{max}}(S')$. Without loss of generality, we can assume $C_{\text{max}}(S') > C_{\text{max}}^*$. Then $j$ must be a small job. It is easy to see that there is at least one machine such that the last job on this machine finishes at or before $C_{\text{max}}^*$. Let $M_i$ be such a machine in $S'$. Then job $j$ must be scheduled on a machine other than $M_i$ in $S'$. If job $j$ is rescheduled to $M_i$ in $S'$, then its new completion time is at most $C_{\text{max}}^* + \frac{p_d}{\epsilon_0}$, which, by LS-ECT, would not be less than the original completion time $C_{\text{max}}(S')$ (see Figure 4). Therefore, we have

$$C_{\text{max}}(S') \leq C_{\text{max}}^* + \frac{p_d}{\epsilon_0} \leq C_{\text{max}}^* + \frac{1}{\epsilon_0} p_d \leq C_{\text{max}}^* + \frac{1}{\epsilon_0} \cdot \frac{m}{d} C_{\text{max}}^* \leq \left(1 + \frac{m}{d \epsilon_0}\right) \cdot C_{\text{max}}^*.$$

Since Algorithm 3.3 returns a schedule $S$ with minimum makespan, the above bound is also an upper bound of $C_{\text{max}}(S)$.

**Lemma 9.** Algorithms 3.3 can be implemented in $O(n + n + (m^d (n - d)(m + m \log (n/m)))$ time.

**Proof.** In step 1, we first find the $d$-th largest job using linear selection algorithm, and then extract the $d$ largest jobs in $O(n)$ time. In step 2, there are at most $m^d$ ways to assign these large jobs to $m$ machines. For each large job assignment, the algorithm schedules the remaining small jobs using LS-ECT rule. As we described in Theorem 6 for the time complexity analysis, with $A_t(i,j)$ pre-calculated in $O(n \tilde{e})$ time, it takes $O \left( (\sum_{i=1}^{m} \log k_i) + m \right)$ time to assign a job to the machine so it completes the earliest. So for each large job assignment, the overall time for scheduling $(n - d)$ small jobs using LS-ECT in step 2 is $O \left( (n - d) \left( m + \sum_{i=1}^{m} \log k_i \right) \right)$. Adding all the time, we get the total running time of Algorithm 3.3 $O(n + n + m^d (n - d)(m + m \log (n/m)))$, which is $O(n + m^d (n - d)(m + m \log (n/m)))$.

Given an instance of $P_m, e_{i,k} \geq \epsilon_0 |C_{\text{max}}$, and a real number $\epsilon \in (0, 1)$. Let $d = \left\lceil \frac{m}{\epsilon \epsilon_0} \right\rceil$. Since $m, \epsilon$, and $\epsilon_0$ are constants, $d$ is also a constant. Without loss of generality, we can assume that $n > d$. If we apply Algorithm 3.3, then by Lemma 8, we get a schedule $S$ whose makespan is at most $(1 + \epsilon) C_{\text{max}}^*$. Combining Lemma 8 and Lemma 9, we get the following theorem.

**Theorem 10.** For any given instance of $P_m, e_{i,k} \geq \epsilon_0 |C_{\text{max}}$ and an error parameter $\epsilon, 0 < \epsilon < 1$, Algorithm 3.3 can return a schedule with makespan at most $(1 + \epsilon) C_{\text{max}}^*$ in $O(\tilde{n} + n + m^\epsilon (n - e_{\epsilon}) (n - \frac{m}{\epsilon \epsilon_0}) (m + m \log (n/m)))$ time.

Next we analyze the performance of Algorithm 3.3 for the more general problem $P_m, e_{i \leq m_1,k} \geq \epsilon_0 |C_{\text{max}}$. We will show that by choosing $d$ appropriately, we can still get a $(1 + \epsilon)$ approximation.

**Theorem 11.** For any given instance of $P_m, e_{i \leq m_1,k} \geq \epsilon_0 |C_{\text{max}}$ and a parameter $d$, Algorithm 1 returns a schedule with makespan at most $\left(1 + \frac{m(m+1-\epsilon^d)}{d \epsilon_0 m_1} \right) C_{\text{max}}^*$ in $O(\tilde{n} + n + m^d (n - d)(m + m \log (n/m)))$ time. In particular, if $d = \left\lceil \frac{m(m+1-\epsilon^d)}{\epsilon \epsilon_0 m_1} \right\rceil$, it is a $(1 + \epsilon)$-approximation.

**Proof.** As in the proof of Lemma 8, we consider the schedule $S'$ from step 2 of Algorithm 1.
that has the same assignment of large jobs as the optimal schedule \( S' \). Let \( j \) be the job such that \( C_j(S') = C_{\text{max}}(S') \). Without loss of generality, we assume \( C_{\text{max}}(S') > C_{\text{max}}(S) \), then \( j \) must be a small job. Let \( p_d \) be the processing requirement of the \( d \)-th largest job. Then we must have \( C_{\text{max}} \geq \frac{d \cdot p_d}{m} \), that is, \( p_d \leq \frac{m}{d} C_{\text{max}} \). It is easy to see that there is at least one machine in \( S' \) where the last job finishes at or before \( C_{\text{max}} \). If there exists one such machine \( M_i \) with \( i \leq m_1 \), we can use similar argument as that of Lemma 8, to show that

\[
C_{\text{max}}(S') = C_j \leq C_{\text{max}} + \frac{p_j}{e_0} \\
\leq \left( 1 + \frac{m}{d \cdot e_0} \right) \cdot C_{\text{max}}.
\]

Otherwise, for all machines \( M_i \), \( 1 \leq i \leq m_1 \), the last job finishes after \( C_{\text{max}} \). By Claim 5, there are at most \( m - 1 \) jobs that finish after \( C_{\text{max}} \) on these \( m_1 \) machines and there must exist one machine where at most \( \left\lfloor \frac{m-1}{m_1} \right\rfloor \) jobs finish after \( C_{\text{max}} \). Similarly, by LS-ECT, moving job \( j \) to this machine does not decrease its completion time. Therefore,

\[
C_{\text{max}}(S') = C_j(S') \leq C_{\text{max}} + \left( \left\lfloor \frac{m-1}{m_1} \right\rfloor + 1 \right) \frac{p_d}{e_0} \\
\leq C_{\text{max}} + \frac{m + m_1 - 1}{m_1} \cdot \frac{m}{d} C_{\text{max}} \cdot \frac{1}{e_0} \\
\leq \left( 1 + \frac{m(m + m_1 - 1)}{d \cdot e_0 \cdot m_1} \right) \cdot C_{\text{max}}.
\]

Algorithm 1 returns a schedule \( S \) that is at least as good as \( S' \), so the above bound is also an upper bound of \( C_{\text{max}}(S) \). Let \( e \) be a real number in \((0, 1)\), if we select \( d = \left\lfloor \frac{m(m+e)-1}{e_0 \cdot m_1} \right\rfloor \), then \( C_{\text{max}}(S) \leq (1 + e) C_{\text{max}} \).

Finally, the analysis of running time remains the same as in Lemma 9.

4 | TOTAL COMPLETION TIME MINIMIZATION

In this section, we study the total completion time minimization problem when there exist one or more machines such that the sharing ratios for all intervals on these machines have a constant lower bound, that is, \( P_m, e_{i \leq m_1, k} \geq e_0 \| \sum C_j \). We first analyze the performance of SPT and its variant SPT-ECT for our problem, and then we develop a PTAS.

4.1 | Performance of classical scheduling algorithms for \( P_m, e_{i \leq m_1, k} \geq e_0 \| \sum C_j \)

It is well known that for the classical problem \( P_m \| \sum C_j \), SPT generates an optimal schedule where the jobs complete in SPT order. Its variant, SPT-ECT rule, which schedules the next shortest job on the machine so it completes the earliest, generates the same optimal schedule as SPT.

Now we consider SPT and SPT-ECT rules for our problem \( P_m, e_{i \leq m_1, k} \geq e_0 \| \sum C_j \). First of all, SPT and SPT-ECT may generate different schedules. Consider two machines where the first machine has a sharing ratio of 1 during the interval \((0, 1]\) and \(\frac{1}{2}\) during \((1, \infty)\), and the second machine has a sharing ratio of 1 all the time. There are 3 jobs with processing requirements of 1, 2, and 2. SPT will schedule one job with processing requirement 2 on \( M_2 \), and two other jobs on \( M_1 \). The total completion time is \( 1 + 5 + 2 = 8 \). SPT-ECT may schedule the first job on \( M_1 \) and the other two jobs on \( M_2 \), resulting in a total completion time of \( 1 + 2 + 4 = 7 \). Moreover, SPT and SPT-ECT do not dominate each other, i.e., for some cases, SPT-ECT generates a better schedule (see the above example), while for other cases, SPT generates a better schedule.

For the above example, if we add one more job with a processing requirement of 3, SPT will generate a better schedule that schedules jobs with processing requirements of 1 and 2 on \( M_1 \), and the other two jobs on \( M_2 \). The total completion time is \( 1 + 5 + 2 + 5 = 13 \). On the other hand, SPT-ECT may schedule the jobs with processing requirements of 1 and 3 on \( M_1 \), and the other two jobs on \( M_2 \). The total completion time is then \( 1 + 2 + 4 + 7 = 14 \).
Next we show that the approximation ratio of SPT rule for our problem is unbounded. Consider two machines where sharing ratio \(e_{1,1} = 1\) during \((0, \infty)\) and \(e_{2,1} = 0\) during \((0, \infty)\), where \(\alpha < 1\). Given two jobs with processing requirement 1 and 1, SPT rule schedules two jobs one on each machine with the total completion time of 1 + \(\frac{1}{\alpha}\) while in the optimal schedule, both jobs are on \(M_1\) with the total completion time of 3. The approximation ratio of SPT for this instance is \(\frac{1 + (1/\alpha)}{3}\). The ratio approaches infinity when \(\alpha\) is close to 0.

Finally we prove that the approximation ratio of SPT-ECT rule for our problem is bounded. For convenience, we first prove the following claim before we give the approximation ratio of SPT-ECT.

**Claim 12.** Given a set of \(n\) primary jobs and 0 routine jobs, let \(I_m\) be the instance consisting of these \(n\) jobs and \(m_1\) identical machines with full capacity, and \(I_m^{*}\) be the instance consisting of these \(n\) jobs and \(m\) identical machines with full capacity. Then the minimum total completion time for \(I_m\) is at most \(\left\lfloor \frac{m}{m_1} \right\rfloor\) times that for \(I_m^{*}\).

**Proof.** Suppose the jobs are indexed by SPT order. We first consider the case that \(m\) is a multiple of \(m_1\), i.e. \(\frac{m}{m_1} = c\) for some integer \(c\).

Let \(S_m^{*}\) be the optimal schedule for \(I_m^{*}\) generated by SPT. Then in \(S_m^{*}\), the indices of jobs scheduled on \(M_i\) will be in the order of \(i, m_1 + i, 2m_1 + i, \ldots\), and so on. Let \(j = \lambda m_1 + i\). Then job \(j\) must be scheduled on \(M_i\) after \(\lambda\) jobs whose indices are \(km_1 + i, 0 \leq k < \lambda\), and its completion time will be

\[
C_j(S_m^{*}) = \sum_{0 \leq k \leq \lambda} (P_{km_1+i}).
\]

Now let us consider the instance \(I_m\) which consists of \(m = cm_1\) machines. Let \(S_m\) be the optimal schedule for \(I_m\) which is generated by SPT. The \(n\) jobs that were scheduled on \(m_1\) machine in \(S_m^{*}\) are now scheduled on \(m\) machines in \(S_m\). For each machine \(M_i\) (1 \(\leq i \leq m\)), the jobs that were originally scheduled on it are now rescheduled in SPT order on a subset of \(\frac{m}{m_1} = c\) machines. So job \(j = \lambda m_1 + i\) will be scheduled after all \(\lambda\) jobs \(km_1 + i, 0 \leq k < \lambda\), have been scheduled onto these \(c\) machines. By SPT rule, job \(j\) has the largest completion time, whose lower bound is the average machine load of all these \(\lambda + 1\) jobs on \(c\) machines. So we have

\[
C_j(S_m) \geq \sum_{0 \leq k \leq \lambda} (P_{km_1+i}) \geq c \cdot C_j(S_m^{*}) = \frac{m_1}{m} C_j(S_m^{*}),
\]

which means

\[
\sum_{j=1}^{n} C_j(S_m^{*}) \leq m \sum_{j=1}^{n} C_j(S_m).
\]

For the case that \(m\) is not a multiple of \(m_1\), let \(m' > m\) be the smallest multiple of \(m_1\). Using the above argument, the minimum total completion time on \(m_1\) machines is at most \(\frac{m'}{m_1} = \left\lfloor \frac{m}{m_1} \right\rfloor\) times the minimum total completion time on \(m'\) machines; the latter is a lower bound on the total completion time on \(m\) machines. This completes the proof.

Now we give the approximation ratio of SPT-ECT for our problem.

**Theorem 13.** For \(P_{m_1}, e_{i \leq m_1,k} \geq e_0\|\sum C_j\), SPT-ECT is a \(\left\lfloor \frac{m}{m_1} \right\rfloor\) approximation algorithm with running time \(O(n \log n + \tilde{n} + n(m + m \log (\tilde{n}/m))\).

**Proof.** Let \(I_m\) be an instance of \(P_{m_1}, e_{i \leq m_1,k} \geq e_0\|\sum C_j\) and \(S_m^{*}\) be the optimal schedule for \(I_m\). Let \(I_{m_1}\) be the corresponding instance that has the same set of primary jobs as \(I_m\) but does not have any routine job, and let \(\tilde{S}_m^{*}\) be the optimal schedule for \(I_{m_1}\). It is obvious that \(\sum C_j(S_m^{*}) \leq \sum C_j(S_m^{*})\). Let \(I_{m_1}\) be the instance that has the same set of primary jobs and no routine job as \(I_m\) but has \(m_1\) machines. And let \(\tilde{S}_m\) be the schedule obtained by applying SPT-ECT to \(I_{m_1}\), which is the same as the schedule obtained by applying SPT. By Theorem 12,

\[
\sum_{j=1}^{n} C_j(\tilde{S}_m^{*}) \leq \frac{m}{m_1} \sum_{j=1}^{n} C_j(\tilde{S}_m^{*}).
\]

Now we analyze the running time of SPT-ECT algorithm. With \(O(n \log n)\) time to sort the jobs and using an implementation similar as LS-ECT for the makespan minimization problem, the total time will be \(O(n \log n + \tilde{n} + n(m + \sum_{i=1}^{m} \log k_i))\).

### 4.2 Approximation Scheme

For \(P_{m_1}, e_{i \leq m_1,k} \geq e_0\|\sum C_j\)

In this section, we develop a PTAS for our problem when the sharing ratios on \((m - 1)\) machines have a lower bound
$e_0$, i.e., $P_m,e_{i\in S_{m-1,k}} \geq e_0||C_j$. For convenience, we introduce the following two notations that will be used in this section:

$P_i(S)$: the total processing requirement of the jobs assigned to machine $M_i$ in $S$.

$\sigma_i(S)$: the total completion time of the jobs scheduled to $M_i$ in $S$.

The idea of our algorithm is to schedule the jobs one by one in SPT order; for each job $j$ to be scheduled, we enumerate all the possible assignments of job $j$ to all machines $M_i$, $1 \leq i \leq m$, and then we prune the set of schedules so that no two schedules are “similar”. Two schedules $S_1$ and $S_2$ are “similar” with respect to a given parameter $\delta$ if for every $1 \leq i \leq m$, $P_i(S_1)$ and $P_i(S_2)$ are both in an interval $[(1+\delta)^i, (1+\delta)^{i+1})$ for some integer $x$, and $\sigma_i(S_1)$ and $\sigma_i(S_2)$ are both in an interval $[(1+\delta)^i, (1+\delta)^{i+1})$ for some integer $y$. We use $S_1 \approx S_2$ to denote that $S_1$ and $S_2$ are “similar” with respect to $\delta$. Our algorithm is formally presented as follows (Algorithm 2).

**Algorithm 2.**

**Input:**
- $e_0$, $e$
- the intervals $(0,t_{i,1})$, $(t_{i,1},t_{i,2})$, … on machine $M_i$, $1 \leq i \leq m$, and their sharing ratios $e_{i,1}, e_{i,2}, …$, $1 \leq i \leq m-1$ and $1 \leq k \leq k_i$, $e_{i,k} \geq e_0$, respectively.
- $n$ jobs with the processing requirements, $p_1, \cdots, p_n$

**Output:** A schedule $S$ whose total completion time is at most $(1 + \epsilon)$ times the optimal.

**Steps:**

1. Reindex the jobs in SPT order.
2. Let $\delta = \frac{e_0}{6n}$
3. Let $U_0 = \{\text{empty schedule}\}$
4. For $j = 1, \ldots, n$, compute $U_j$ which is a set of schedules of the first $j$ jobs:
   a. $U_j = \emptyset$
   b. For each schedule $S_{j-1} \in U_{j-1}$
      - If $i = 1 \ldots m$
        - Add job $j$ to the end of $M_i$ in $S_{j-1}$, let the schedule be $S_j$
        - $U_j = U_j \cup \{S_j\}$
      c. Prune $U_j$ by repeating the following until $U_j$ cannot be reduced
         - If there are two schedules $S_1$ and $S_2$ in $U_j$ such that $S_1 \approx S_2$
           - If $P_m(S_1) \leq P_m(S_2)$, $U_j = U_j \setminus \{S_2\}$
           - Else $U_j = U_j \setminus \{S_1\}$

5. Return the schedule $S \in U_n$ that minimizes $\sum_{i=1}^{m} \sigma_i$

**Theorem 14.** Algorithm 4.2 is a $(1 + \epsilon)$-approximation scheme for $P_m,e_{i\in S_{m-1,k}} \geq e_0||C_j$, and it runs in time $O(n \log n + n(m+n)) (\frac{1}{e_0^2}(\log P)(\log n^2)(\log n^2 \log n^2))$, where $P = \sum p_j$.

**Proof.** Let $S^*$ be the optimal schedule. We use $S^*_i$ to denote the partial schedule of the first $j$ jobs in $S^*$. We first prove by induction the following claim:

For each job $1 \leq j \leq n$, there is a partial schedule $S_j \in U_j$ such that

- Property (1): $P_m(S_j) \leq P_m(S^*_j)$.
- Property (2): $P_i(S_j) \leq (1 + \delta)^i P_i(S^*_j)$ for $1 \leq i \leq m - 1$, and
- Property (3): $\sigma_i(S_j) \leq (1 + \delta)^i (1 + \frac{2n\delta}{e_0}) \sigma_i(S^*_j)$ for $1 \leq i \leq m$.

It is trivial for $j = 1$. Assume the hypothesis is true for $j$, so we have a schedule $S_j \in U_j$ with properties (1)-(3). Consider the schedule of job $j + 1$ in $S^*$.

**Case 1.** In $S^*$, job $j + 1$ is scheduled on $M_m$. Then $P_m(S^*_{j+1}) = P_m(S^*_j) + p_{j+1}$. Let $S_{j+1}$ be the schedule obtained from $S_j$ by scheduling job $(j+1)$ on $M_m$. Then we have

$$P_m(S_{j+1}) = P_{m}(S_j) + p_{j+1} \leq P_{m}(S^*_j) + p_{j+1} \quad \text{by Property (1)}$$

$$= P_{m}(S^*_j) + p_{j+1}.$$ (4)

For $1 \leq i \leq m - 1$, $S_j$ and $S_{j+1}$ are the same on $M_i$, so are schedules $S^*_j$ and $S^*_j$. Thus, we have

$$P_i(S_{j+1}) = P_i(S_j) \leq (1 + \delta)^i P_i(S^*_j) = (1 + \delta)^i P_i(S^*_j),$$ (5)

and for $1 \leq i \leq m - 1$,

$$\sigma_i(S_{j+1}) = \sigma_i(S_j) \leq (1 + \delta)^i \left(1 + \frac{2n\delta}{e_0}\right) \sigma_i(S^*_j)$$

$$= (1 + \delta)^i \left(1 + \frac{2n\delta}{e_0}\right) \sigma_i(S^*_j).$$ (6)

For $i = m$, note that (4) implies $C_{j+1}(S_{j+1}) \leq C_{j+1}(S^*_j)$,

$$\sum_{i=1}^{m} \sigma_i = \sum_{i=1}^{m} \sigma_j + C_{j+1}(S_{j+1}) \leq (1 + \delta)^i \left(1 + \frac{2n\delta}{e_0}\right) \sigma_m(S^*_j) + C_{j+1}(S^*_j) \quad \text{by Property (3)}$$

$$\leq (1 + \delta)^i \left(1 + \frac{2n\delta}{e_0}\right) \left(\sigma_m(S^*_j) + C_{j+1}(S^*_j)\right)$$

$$\leq (1 + \delta)^i \left(1 + \frac{2n\delta}{e_0}\right) \left(\sigma_m(S^*_j) + C_{j+1}(S^*_j)\right).$$ (7)

If $S_{j+1}$ is not pruned, i.e., $S_{j+1} \in U_{j+1}$, inequalities (4)-(7) mean $S_{j+1}$ is the schedule in $U_{j+1}$ with
Properties (1)-(3). Otherwise, there must exist another schedule $S'_{j+1} \in U_{j+1}$ such that $S'_{j+1} \approx$ $S_{j+1}$ which implies

$$P_m(S'_{j+1}) \leq P_m(S_{j+1}),$$

$$P_i(S'_{j+1}) \leq (1 + \delta)P_i(S_{j+1}) \text{ for } 1 \leq i \leq m - 1,$$

and

$$\sigma_i(S'_{j+1}) \leq (1 + \delta)\sigma_i(S_{j+1}) \text{ for } 1 \leq i \leq m.$$

Combining with the above inequalities (4)-(7), we get:

$$P_m(S'_{j+1}) \leq P_m(S_{j+1}),$$

$$P_i(S'_{j+1}) \leq (1 + \delta^j+1P_i(S_{j+1}) \text{ for } 1 \leq i \leq m - 1,$$

and

$$\sigma_i(S'_{j+1}) \leq (1 + \delta^j+1(1 + \frac{2n\delta}{e_0})\sigma_i(S_{j+1}) \text{ for } 1 \leq i \leq m.$$

Thus, $S'_{j+1}$ is the schedule in $U_{j+1}$ with Properties (1)-(3) in this case.

**Case 2.** In $S^*$, $j + 1$ is scheduled on machine $M_k$, $1 \leq k \leq m - 1$. Let $S_{j+1}$ be the schedule obtained from $S_j$ by scheduling job $(j + 1)$ on $M_k$. Then for $M_k$ in $S_{j+1}$, we have

$$P_k(S_{j+1}) = P_k(S_j) + p_{j+1}$$

$$\leq (1 + \delta^j)P_k(S_j) + p_{j+1}$$

$$\leq (1 + \delta^j(1 + p_{j+1})$$

$$= (1 + \delta^j)P_k(S_{j+1}).$$

Thus, $P_k(S_{j+1}) - P_k(S_{j+1}) \leq ((1 + \delta^j - 1)P_k(S_{j+1}).$

Since $(j + 1)$ is scheduled on $M_k$ in both $S_j$ and $S_{j+1}$, the difference of the completion time of $j + 1$ in these schedules will be

$$C_{j+1}(S_{j+1}) - C_{j+1}(S_{j+1}) \leq \frac{P_k(S_{j+1}) - P_k(S_{j+1})}{e_0} \leq \frac{(1 + \delta^j - 1)P_k(S_{j+1})}{e_0}.$$

Obviously $P_k(S_{j+1}) \leq C_{j+1}(S_{j+1})$; and since $\delta < \frac{1}{n}$, we can show that

$$(1 + \delta^j - 1) \leq (1 + \delta^n - 1 = \sum_{i=1}^{n} (\begin{pmatrix} n \\ i \end{pmatrix})\delta^i$$

$$\leq n\delta \left( \sum_{i=1}^{n} \frac{1}{n!} \right) \leq 2n\delta,$$

thus

$$C_{j+1}(S_{j+1}) - C_{j+1}(S_{j+1}) \leq \frac{2n\delta}{e_0} C_{j+1}(S_{j+1}).$$

Therefore,

$$\sigma_k(S_{j+1}) = \sigma_k(S_{j}) + C_{j+1}(S_{j+1}) \leq (1 + \delta^j(1 + \frac{2n\delta}{e_0})\sigma_k(S_{j}) + C_{j+1}(S_{j+1})$$

$$\leq (1 + \delta^j(1 + \frac{2n\delta}{e_0})\sigma_k(S_j) + C_{j+1}(S_{j+1})$$

$$< (1 + \delta^j(1 + \frac{2n\delta}{e_0})\sigma_k(S_j)$$

$$+ (1 + \frac{2n\delta}{e_0})C_{j+1}(S_{j+1}).$$

Since for other machines, the schedules $S_j$ and $S_{j+1}$ are same and so are the schedules $S'_j$ and $S'_{j+1}$, we have

$$P_m(S_{j+1}) < P_m(S'_{j+1})$$

$$P_i(S_{j+1}) < (1 + \delta^j)P_i(S_{j+1}), \text{ for } i \neq m, k,$$

and

$$\sigma_i(S_{j+1}) < (1 + \delta^j)(1 + \frac{2n\delta}{e_0})\sigma_i(S_{j+1}), \text{ for } 1 \leq i \leq m.$$

Same as Case 1, we can show the properties (1)-(3) hold no matter $S_{j+1} \in U_{j+1}$ or not.

Thus at the end of the algorithm, after all $n$ jobs have been processed,

$$\sum_{j=1}^{n} C_j(S) = \sum_{i=1}^{m} \sigma_i(S)$$

$$\leq (1 + \delta^n)(1 + \frac{2n\delta}{e_0})\sum_{i=1}^{m} \sigma_i(S^*)$$

$$= (1 + \delta^n)(1 + \frac{2n\delta}{e_0})\sum_{j=1}^{n} C_j(S^*)$$

$$\leq (1 + 2n\delta)(1 + \frac{2n\delta}{e_0})\sum_{j=1}^{n} C_j(S^*)$$

$$\leq \left( 1 + \frac{2n\delta}{e_0} \right)^2 \sum_{j=1}^{n} C_j(S^*)$$

By step 2 of Algorithm 4.2, $\delta = \frac{e_0}{6n}$, then

$$\sum_{j=1}^{n} C_j(S) \leq \left( 1 + \frac{\varepsilon}{3} \right)^3 \sum_{j=1}^{n} C_j(S^*) \leq (1 + \varepsilon)\sum_{j=1}^{n} C_j(S^*).$$

Now we consider the running time. Let $P = \sum P_j$, then after $U_j$ is pruned, for any schedule $S \in U_j$ and for any machine $M_i$, $1 \leq i \leq m$, $P_i(S)$ can take at most $\log_{1+\delta} P$ values; $\sigma_i(S)$ can take at most $\log_{1+\delta}(\frac{n\sigma}{e_0})$ values. Thus after pruning, there are at most $(\log_{1+\delta} P \log_{1+\delta} n^p \frac{e_0}{\varepsilon})^m$ schedules in $U_j$. In each iteration when job $j$ is added, at most $n$ sharing intervals are considered for all machines, so the total time for Step 4 of Algorithm 4.2 will be

$$O \left( n(m + n)(\log_{1+\delta} P \log_{1+\delta} n^p \frac{e_0}{\varepsilon})^m \right).$$

Plugging in $\delta = \frac{e_0}{6n}$ and adding the sorting time to get the jobs in SPT order, the total time is $O \left( n \log n + n(m + n) \left( \frac{36}{e_0^2} \log \frac{n^p}{e_0} \right)^2 \right)^m$. 


5 | CONCLUSIONS

In this paper we studied the problem of multitasking scheduling with shared processing in the parallel machine environment and with different fractions of machine capacity assigned to routine jobs. The objectives are minimizing makespan and minimizing the total completion time of the primary jobs. For both criteria, we proved the inapproximability of the problem with arbitrary sharing ratios for all machines. Then we focused on the problems where some machines have a positive constant lower bound for the sharing ratios. For makespan minimization problem, we analyzed the approximation ratios of LS and LS-ECT and developed an approximation scheme that runs in linear time when the number of machines is a constant. For total completion time minimization, we studied the performance of SPT and SPT-ECT and developed an approximation scheme that runs in polynomial time when the number of machines is a constant.

Our research leaves one unsolved case for the total completion time minimization problem: is there an approximation scheme when $m > 2$ and more than one machines have arbitrary sharing ratios? For the future work, it is also interesting to study other performance criteria including maximum tardiness, the total number of tardy jobs and other machine environments such as uniform machines, flowshop, etc. Moreover, in our work we assume that all the routine jobs have been predetermined and have fixed release times and duration as well as fixed processing capacity. In some applications, routine jobs may have relaxed time windows to be processed and flexible processing capacity that could be changed depending on where in the window the routine jobs are processed. For this scenario, one needs to consider the schedules for both primary jobs and routine jobs simultaneously.

REFERENCES

Adiri, I., & Yehudai, Z. (1987). Scheduling on machines with variable service rates. *Computers & Operations Research, 14*(4), 289–297.

Baker, K. R., & Nuttle, H. L. W. (1980). Sequencing independent jobs with a single resource. *Naval Research Logistics Quarterly, 27*, 499–510.

Coviello, D., Ichino, A., & Persico, N. (2014). Time allocation and task juggling. *American Economic Review, 104*(2), 609–623.

Denning, P. J. (1971). Third generation computer systems. *Computing Surveys, 3*(4), 175–216.

González, V. M., & Mark, G. (2005). Managing currents of work: Multi-tasking among multiple collaborations. In H. Gellersen, K. Schmidt, M. Beaudouin-Lafon, W. Mackay (Eds.), *ECSCW* (pp. 143–162). Springer, Dordrecht. https://doi.org/10.1007/1-4020-4023-7_8

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics, 17*, 416–429.

Graham, R. L., Lawler, E. L., Lenstra, J. K. L., Kan, A. H. G. R. K., & Optmization, R. (1979). *approximation in deterministic sequencing and scheduling: A survey*. In P. Hammer, E. Johnson, & B. Korte (Eds.), *Discrete Optimization II Annals of Discrete Mathematics* (Vol. 5, pp. 287–326). Elsevier.

Hall, N. G., Leung, J. Y.-T., & Li, C.-L. (2015). The effects of multitasking on operations scheduling. *Production and Operations Management, 24*, 1248–1265.

Hall, N. G., Leung, J. Y.-T., & Li, C.-L. (2016). Multitasking via alternate and shared processing: Algorithms and complexity. *Discrete Applied Mathematics, 208*, 41–58.

Hochbaum, D. S., & Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM, 34*(1), 141–162.

Horowitz, E., & Sahni, S. (1976). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM, 23*, 317–327.

Kellerer, H. (1998). Algorithms for multiprocessor scheduling with machine release times. *IEEE Transactions, 30*, 991–999.

Lee, C.-Y. (1991). Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics, 30*, 53–61.

Lee, C.-Y., & Liman, S. D. (1993). Capacitated two-parallel machine scheduling to minimize sum of job completion times. *Discrete Applied Mathematics, 41*, 211–222.

Legros, B., Jouini, O., Akşin, O. Z., & Koole G. (2020). Front-office multitasking between service encounters and back-office tasks. *European Journal of Operational Research, 287*(3), 946–963. https://doi.org/10.1016/j.ejor.2020.04.048.

Ma, Y., Chu, C., & Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering, 58*(2), 199–211.

ÖLeary, K. J., Liebovitz, D. M., & Baker, D. W. (2006). How hospitalists spend their time: insights on efficiency and safety. *Journal of Hospital Medicine, 1*, 88–93.

Sun, J., & Ho, K. (2015). Analysis on the effect of multitasking. In 2015 IEEE international conference on systems, man, and cybernetics (pp. 204–209). https://doi.org/10.1109/SMC.2015.48.

Zhu, Z., Zheng, F., & Chu, C. (2017). Multitasking scheduling problems with a rate-modifying activity. *International Journal of Production Research, 55*, 296–312.

How to cite this article: Fu, B., Huo, Y., & Zhao, H. (2024). Multitasking scheduling with shared processing. *Naval Research Logistics (NRL), 71*(4), 595–608. https://doi.org/10.1002/nav.22167