European Journal of Operational Research 306 (2023) 567–578

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

An improved approximation algorithm for scheduling monotonic moldable tasks

Fangfang Wu[a], Xiandong Zhang[a], Bo Chen[b,*]

[a] School of Management, Fudan University, Shanghai 200433, China
[b] Warwick Business School, University of Warwick, Coventry CV4 7AL, UK

ARTICLE INFO

Article history:
Received 6 August 2021
Accepted 24 August 2022

Keywords:
Scheduling
Moldable tasks
Approximation algorithm

ABSTRACT

We are concerned with the problem of scheduling monotonic moldable tasks on identical processors to minimize the makespan. We focus on the natural case where the number m of processors as resources is fixed or relatively small compared with the number n of tasks. We present an efficient $(3/2)$-approximation algorithm with time complexity $O(nm\log(nm))$ (for $m > n$) and $O(n^2\log n)$ (for $m \le n$). To the best of our knowledge, the best relevant known results are: (a) a $(3/2+\epsilon)$-approximation algorithm with time complexity $O(nm\log(n/\epsilon))$, (b) a fully polynomial-time approximation scheme for the case of $m \ge 16n/\epsilon$, and (c) a polynomial-time approximation scheme with time complexity $O(n^{g(1/\epsilon)})$ when m is bounded by a polynomial in n, where $g(\cdot)$ is a super-exponential function. On the other hand, the novel general technique developed in this paper for removing the ϵ-term in the worst-case performance ratio can be applied to improving the performance guarantee of certain dual algorithms for other combinatorial optimization problems.

1. Introduction

A moldable task is one that can be executed on an arbitrary number of processors simultaneously with execution time as a function of the number of processors allotted to it. This task model captures task parallelism in scheduling and allows the scheduler to decide on the degree of parallelism for each task by choosing the number of processors assigned to it. Due to these properties, the model is used to describe some industrial applications (Fotakis, Matuschke, & Papadigenopoulos, 2021). For example, in workforce assignment (Delorme, Dolgui, Kovalev, & Kovalyov, 2019; Dolgui, Kovalev, Kovalyov, Malyutin, & Soukhal, 2018), workers with identical skills need to perform operations. An operation can be performed by one or more workers and the processing time of the operation depends on the number of workers performing it. Also, a berth and quay crane allocation problem, which is to decide the arrival and departure time of ships as well as the number of quay cranes assigned to each ship, was considered in Blazewicz, Cheng, Machowiak, & Oguz (2011). They formulated this problem as scheduling moldable tasks, in which ships are considered as tasks and quay cranes are considered as processors. For each ship, the berthing duration is a function of the number of quay cranes allotted to it. The goal is to minimize the maximum departure time of the ships in berth and quay crane allocation, which is achieved by minimizing the makespan in the model of scheduling moldable tasks. Unsal (2021) considered some specific characteristics of a berth and quay crane allocation problem and developed an extended formulation of moldable task scheduling.

The moldable task model was also proposed by theoretical researchers in the fields of computer science. For example, Feitelson & Rudolph (1996) stated that programs written using the SPMD style, e.g., with the MPI library package, are often moldable. Drozdowski (2009) discussed the practical motivation for moldable tasks involved in parallel applications. Bleuse et al. (2017) considered scheduling independent moldable tasks on multi-cores with GPUs. They assumed that those tasks are parallelizable on CPUs using the moldable model. Also, in Fujiwara, Tanaka, Taura, & Torisawa (2018), deep learning tasks were treated as moldable tasks. Therefore, efficient and sophisticated algorithms are then required for the implementation of these applications to ensure good performance.

1.1. Problem description

We are given a set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of n moldable tasks and a set $M = \{1, 2, \ldots, m\}$ of m identical processors. Each task T_i has

* Corresponding author.
E-mail addresses: ffwu17@fudan.edu.cn (F. Wu), xiandongzhang@fudan.edu.cn (X. Zhang), b.chen@warwick.ac.uk (B. Chen).

https://doi.org/10.1016/j.ejor.2022.08.034

an associated execution time function $t_i : p \to t_{i,p}$ that gives the execution time $t_{i,p} \in \mathbb{Q}^+$ of task T_i if p processors are allotted to executing task T_i, where $\mathbb{Q}^+$ denotes the set of positive rational numbers (Jansen & Porkolab, 2002; Jansen & Thöle, 2010). We define the workload function w_i of task T_i as $w_i : p \to w_{i,p} = p \times t_{i,p}$ for $p \le m$, which gives the workload $w_{i,p}$ of task T_i if p processors are allotted to it. As a key aspect of a moldable task, several types of execution time functions have been introduced (Turek, Wolf, & Yu, 1992; Wang & Cheng, 1991). In this paper, we study a commonly used type — *monotonic*: the execution time function t_i is a non-increasing function while the corresponding workload function w_i is a non-decreasing function: $t_{i,p} \ge t_{i,p'}$ and $w_{i,p} \le w_{i,p'}$ hold for all $i = 1, \ldots, n$ and for any pair $p, p' \in \{1, \ldots, m\}$ with $p < p'$. The monotonicity assumption takes into account both the speedup in execution time and the slowdown due to communication overhead (Bleuse et al., 2017; Mounié, Rapine, & Trystram, 1999). Such an assumption can also be interpreted by the well-known Brent's lemma (Brent, 1974), which states that running tasks in parallel can reduce the overall processing time and leads to sublinear speedups. In addition, this assumption is reasonable in practice as one can find the simulation results of the Fourier Transform benchmark and Block Tridiagonal benchmark in Dutton, Mao, Chen, & Watson III (2008). Note that the execution function $t_{i,p} = t_{i,1}/p + (p-1)c$ used in Dutton et al. (2008), where c is a positive constant, is a widely used function to model execution times (Ghosal, Serazzi, & Tripathi, 1991; Leuze, Dowdy, & Park, 1989), which implies monotonicity when c is small enough.

We assume all tasks are monotone. For any task T_i ($1 \le i \le n$), let P_i be the set of processors allotted to the task. Then these $|P_i|$ processors have to execute task T_i simultaneously without preemption, i.e., they start executing task T_i at some starting time s_i and complete it at $s_i + t_{i,|P_i|}$. Each processor can execute at most one task at a time. A feasible non-preemptive *schedule* is an allotment of processors $P_i \subseteq M$ and a starting time s_i for every task T_i, such that for any time point τ it holds that $\sum_{1 \le i \le n:\, \tau \in \left[s_i, s_i + t_{i,|P_i|}\right)} |P_i| \le m$. Note that we do not require that a task has to be executed by contiguously numbered processors. In other words, we study non-contiguous schedules. Our objective is to find a feasible non-preemptive schedule that minimizes the makespan, which is defined as the maximum completion time $C_{\max} = \max_{1 \le i \le n} \left\{ s_i + t_{i,|P_i|} \right\}$. Since processors are usually to model scarce resources, we are concerned in this paper with the case where m is fixed or relatively small compared with n.

Note that in the above problem description, the number $|P_i|$ of processors allotted to task T_i is a constant during the execution of the task, from s_i to $s_i + t_{i,|P_i|}$. Although moldable tasks are called *malleable* tasks in several studies (Chen & Chu, 2013; Jansen & Porkolab, 2002; Jansen & Thöle, 2010; Mounié, Rapine, & Trystram, 2007; Turek et al., 1992), more recently the latter notion is referred to the case where the number of processors allotted to a task can be changed during the execution of the task (Drozdowski, 2009; Marchal, Simon, Sinnen, & Vivien, 2018).

Before reviewing the literature, let us introduce a few terminologies. A polynomial-time approximation scheme (PTAS) is a collection of $(1+\epsilon)$-approximation algorithms parameterized in arbitrarily small $\epsilon > 0$ with running time being a polynomial in the problem size for any fixed ϵ. A fully polynomial-time approximation scheme (FPTAS) is a collection of $(1+\epsilon)$-approximation algorithms parameterized in arbitrarily small $\epsilon > 0$ with running time polynomial in both problem size and $1/\epsilon$. An asymptotic fully polynomial-time approximation scheme (AFPTAS) is a collection of asymptotic $(1+\epsilon)$-approximation algorithms parameterized in arbitrarily small $\epsilon > 0$ that runs in time polynomial both in the problem size and $1/\epsilon$.

1.2. Related work

The problem of scheduling moldable tasks has been extensively studied. The problem is known to be NP-hard (Du & Leung, 1989) and remains so even with monotonicity assumption (Jansen & Land, 2018). For non-monotonic tasks, there is no polynomial-time algorithm with an approximation guarantee less than 3/2 unless P = NP (Johannes, 2006). An overview of the known results for scheduling moldable tasks is given in Table 1. In the table, "contiguous" (resp., "non-contiguous") indicates that contiguous (resp., non-contiguous) schedules are considered, where each task is required to be executed by contiguously numbered processors (resp., no such requirement).

Other two problems closely related to scheduling moldable tasks are scheduling rigid tasks and strip packing. A rigid task is also called non-malleable or non-moldable task in several studies (Jansen, 2012; Turek et al., 1992), for which the execution time and the number of required processors are known *a priori*. In the problem of strip packing, given a strip with a fixed width and an infinite height, we need to pack a set of rectangular items into the strip without rotation and overlapping. The objective is to find a feasible packing with minimal packing height. The difference between scheduling rigid tasks and strip packing is that processors allotted to a task need to be contiguous in the latter problem.

Scheduling rigid tasks. The problem is strongly NP-hard even if the number m of available processors is a constant at least 4 (Du & Leung, 1989; Henning, Jansen, Rau, & Schmarje, 2020). Garey & Graham (1975) provided a 2-approximation algorithm and the approximation guarantee was later improved to $3/2 + \epsilon$ (Jansen, 2012), which is very close to the lower bound of 3/2 (Johannes, 2006). When m is a constant, a PTAS was presented by Amoura, Bampis, Kenyon, & Manoussakis (2002) and Jansen & Porkolab (2002). Jansen & Thöle (2010) provided a PTAS for the more general case where the number m is bounded by a polynomial in the number of tasks. An overview of the known results for scheduling rigid tasks is given in Table 2.

Strip packing. One of the first results for strip packing was provided by Coffman, Garey, Johnson, & Tarjan (1980). They provided two algorithms with approximation ratios of 3 and 2.7, respectively. The approximation ratio was later improved in a series of studies (Harren & Stee, 2009; Schiermeyer, 1994; Sleator, 1980; Steinberg, 1997) and the best approximation ratio achieved so far is $5/3 + \epsilon$ (Harren, Jansen, Prädel, & van Stee, 2011). Similarly, the asymptotic approximation ratio has also been improved (Baker, Brown, & Katseff, 1981; Baker, Coffman, & Rivest, 1980; Golan, 1981; Jansen & Solis-Oba, 2009; Kenyon & Rémila, 2000; Sviridenko, 2012). When the width of the strip denoted by W is allowed to appear polynomially in the running time, there is no polynomial-time algorithm with approximation guarantee less than 5/4 unless P = NP (Henning et al., 2020). On the other hand, algorithms with approximation ratio $4/3 + \epsilon$ were presented by Gálvez, Grandoni, Ingala, & Khan (2016) and Jansen & Rau (2017). The gap to the lower bound was later closed with a $(5/4 + \epsilon)$-approximation algorithm (Jansen & Rau, 2019). An overview of the known results for scheduling rigid tasks is given in Table 3, where $h_{\max}$ and $w_{\max}$ denote the largest height and width of rectangles, respectively, and W denotes the width of the strip.

1.3. Our contribution

In this paper, we are concerned with scheduling monotonic moldable tasks when the number of processors is fixed or relatively small compared with the number of tasks. We present an efficient algorithm with performance guarantee of 3/2 and with time complexity of $O(nm \log(nm))$ (for $m > n$) and $O(n^2 \log n)$ (for $m \le n$).

Table 1
Known results for scheduling moldable tasks.

	Tasks	Processors	Ratio	Remarks
Turek et al. (1992)	non-monotonic	non-contiguous	2	$O(nm\log n)$
Turek et al. (1992)	non-monotonic	contiguous	5/2	$O(n^2 m\log n)$
Ludwig & Tiwari (1994)	non-monotonic	non-contiguous	2	$O(nm)$
Ludwig & Tiwari (1994)	non-monotonic	contiguous	5/2	$O(nm)$
Jansen & Porkolab (2002)	non-monotonic	non-contiguous	PTAS	$O(n)$ with m being constant
Jansen (2004)	non-monotonic	non-contiguous	AFPTAS	$O(n(1/\epsilon^2+\log n)\log(1/\epsilon)\max(n\log\log(n/\epsilon), n/\epsilon^4)+\max(\log(1/\epsilon)/\epsilon^3, \mathcal{M}(\epsilon^{-2}))/\epsilon^4+nm)$, where $\mathcal{M}(N)$ is the time to invert an $(N\times N)$ matrix
Jansen & Thöle (2010)	non-monotonic	non-contiguous	PTAS	$O(n^{f(1/\epsilon)})$ for some exponential function f; m is polynomially bounded by n
Jansen (2012)	non-monotonic	non-contiguous	$3/2+\epsilon$	$O(n\log n)+f(1/\epsilon)$ for some exponential function f
Jansen & Rau (2019)	non-monotonic	contiguous	$5/4+\epsilon$	$O((nm)^{1/\epsilon^{2^{O(1/\epsilon^{13})}}})$
Ludwig & Tiwari (1994)	monotonic	non-contiguous	2	$O(n\log^2 m)$
Ludwig & Tiwari (1994)	monotonic	contiguous	5/2	$O(n\log^2 m)$
Mounié et al. (1999)	monotonic	contiguous	$\sqrt{3}+\epsilon$	$O((\min\{nm, n^3\}+n\log m)\log(1/\epsilon)+n\log^2 m)$
Mounié et al. (2007)	monotonic	non-contiguous [a]	$3/2+\epsilon$	$O(nm\log(n/\epsilon))$
Jansen & Land (2018)	monotonic	non-contiguous	FPTAS	$O(n\log^2 m(\log m+\log(1/\epsilon)))$; $m\geq 16n/\epsilon$ for any $\epsilon>0$
Jansen & Land (2018)[b]	monotonic	non-contiguous	$3/2+\epsilon$	$O(n\log^2 m+\log(T(n,m,\epsilon)/\epsilon))$, where $T(n,m,\epsilon)=O(n(\log m)+n\log(\epsilon m))$, $O(n(\frac{1}{\epsilon^2}\log m(\frac{\log m}{\epsilon}+\log^3(\epsilon m))+\log n))$ and $O(n\frac{1}{\epsilon^2}\log m(\frac{\log m}{\epsilon}+\log^3(\epsilon m)))$
Jansen & Rau (2019)	monotonic	contiguous	$5/4+\epsilon$	$O(n^{1/\epsilon^{2^{O(1/\epsilon^{13})}}})$; $m<8n/\epsilon$
This paper	monotonic	non-contiguous	3/2	$O(nm\log(nm))$ for $m>n$; $O(n^2\log(n))$ for $m\leq n$

[a] Mounié et al. (2007) claimed that they constructed a contiguous schedule, but the claim was unjustified. In their algorithm, if a task satisfies some rules, it will be split into two parts. One part is allocated to a processor in a shelf, the other to a processor in another shelf. Since only one pair of contiguously numbered processors exists between two shelves and it is possible to have more than one task satisfying the rule, this process results in a non-contiguous schedule.
[b] Jansen & Land (2018) provided three different approximation algorithms with the same approximation ratio of $3/2+\epsilon$. Hence, three different time complexity estimations are given in the remarks.

Table 2
Known results for scheduling rigid tasks.

	Ratio	Remarks
Garey & Graham (1975)	2	$O(n)$; m constant
Amoura et al. (2002)	PTAS	$O(n)$; m constant
Jansen & Porkolab (2002)	PTAS	$O(n)$; m constant
Jansen & Thöle (2010)	PTAS	$O(n^{f(1/\epsilon)})$ for some exponential function f; m polynomially bounded by n
Jansen (2012)	$3/2+\epsilon$	$O(n\log n)+f(1/\epsilon)$ for some exponential function f

Table 3
Known results for strip packing.

	Ratio	Time Complexity	Remarks
Coffman et al. (1980)	3	$O(n\log n)$	
Coffman et al. (1980)	2.7	$O(n\log n)$	
Sleator (1980)	2.5	$O(n\log n)$	
Schiermeyer (1994)	2	$O(n\log n)$	
Steinberg (1997)	2	$O((n\log^2 n)/\log\log n)$	
Harren & Stee (2009)	1.9369	$O(T_{\text{PTAS}}+(n\log^2 n)/\log\log n)$	T_{PTAS} is the running time of PTAS presented by Bansal, Caprara, Jansen, Prädel, & Sviridenko (2009)
Harren et al. (2011)	$5/3+\epsilon$	$O(T_{\text{PTAS}}+n\log^2 n)$	
Baker et al. (1980)	3	$O(n\log n)$	asymptotic approximation ratio
Golan (1981)	4/3	$O(n\log n)$	
Baker et al. (1981)	5/4	$O(n\log n)$	
Kenyon & Rémila (2000)	AFPTAS	$O(n\log n+\epsilon^{-6}\log^3 n\log^3(\frac{1}{\epsilon}))$	
Jansen & Solis-Oba (2009)	APTAS	$O(n^{1/\epsilon^{O(1/\epsilon)}})$	
Sviridenko (2012)	AFPTAS	$O(n\log n+\epsilon^{-6}\log^3 n\log^3(\frac{1}{\epsilon}))$	
Nadiradze & Wiese (2016)	$7/5+\epsilon$	$O((\max\{w_{\max}, h_{\max}\}n)^{O(1)})$	W is polynomially bounded by n
Gálvez et al. (2016)	$4/3+\epsilon$		
Jansen & Rau (2017)	$4/3+\epsilon$	$O((nW)^{1/\epsilon^{O(2^{1/\epsilon})}})$	
Jansen & Rau (2019)	$5/4+\epsilon$	$O(n\log(n))W^{1/\epsilon^{2^{O(1/\epsilon^{13})}}}$	

The best polynomial time algorithm before this paper has a performance guarantee of $3/2+\epsilon$ with running time a polynomial in problem size and $1/\epsilon$ (Jansen & Land, 2018; Mounié et al., 2007). We develop a new technique in this paper to remove the ϵ-term in the performance ratio as well as from the running time. The technique can be used in eliminating the ϵ-term in the running time of certain dual algorithms for other combinatorial optimization problems.

Although a PTAS exists for this problem, its time complexity $O(n^{h(1/\epsilon)})$ is prohibitively high for some super-exponential function h (Jansen & Thöle, 2010). On the other hand, while an FPTAS can achieve an approximation ratio of $1+\epsilon$, it requires that $m\geq 16n/\epsilon$ (Jansen & Land, 2018; Mounié et al., 2007). Therefore, our (3/2)-approximation algorithm offers a much better time complexity compared with the PTAS and a better approximation guarantee compared with the FPTAS when $m<32n$, which is the case if m is fixed or relatively small compared with n.

1.4. Organization

This paper is organized as follows. We start with some preliminaries in Section 2, which include notation and some known

Table 4
Notation dashboard.

Variable	Definition
n	Number of tasks
m	Number of processors
$\mathcal{T}$	Set of all n tasks
$C^*_{\max}$	Optimal makespan
W^*	Workload of an optimal schedule
$t_{i,p}$	Execution time of task T_i if p processors are allotted to task T_i
$w_{i,p}$	Workload of task T_i if p processors are allotted to task T_i
$\gamma(i,h)$	Minimal number of processors needed to execute task T_i within h time units
X	$=\{t_{j,p} : j=1,\ldots,n; p=1,\ldots,m\}$
Y	$=X\cup\{t_{i,1}+t_{j,1}, t_{i,1}+t_{j,2} : i,j=1,\ldots,n\}$
τ_1	Largest value in X, but no more than $C^*_{\max}$, used as estimate of $C^*_{\max}$
τ_2	Largest value in $X\cup\{0\}$, but no more than $\frac{1}{2}C^*_{\max}$, used as estimate of $\frac{1}{2}C^*_{\max}$
τ_3	Largest value in Y, but no more than $\frac{3}{2}C^*_{\max}$, used as estimate of $\frac{3}{2}C^*_{\max}$
$\mathcal{T}_L$	Set of "large" tasks with $t_{i,1}>\tau_2$
$\mathcal{T}_S$	Set of "small" tasks with $t_{i,1}\le\tau_2$
S_1	Set of m_1 processors with available period $[0,\tau_1]$ (see Fig. 3)
S_2	Set of m_1 processors with available period $(\tau_1,\tau_1+\tau_2]$ (see Fig. 3)
S_0	Set of m_0 processors with available period $[0,\tau_3]$ (see Fig. 3)
q	Number of idle processors in S_1
$\sigma_1,\sigma_2,\sigma_3$	Partial schedules obtained in Procedure $A(\mathcal{T},m,\tau_1,\tau_2,\tau_3)$ under different scenarios
σ_0	Exactly one of $\sigma_1,\sigma_2,\sigma_3$.
$\tilde{\sigma}$	Possible full schedule output of Procedure $F(\mathcal{T},m)$
σ	Full schedule output of Procedure $A(\mathcal{T},m,\tau_1,\tau_2,\tau_3)$

results in the literature that our work is built on. An outline of our approximation algorithm, followed by its detailed description is provided in Section 3. The theoretical analysis of the algorithm is provided in Sections 4 and 5. We make some concluding remarks in Section 6.

2. Preliminaries

We summarize our notations in the following Table 4, most of which will appear afterwards.

2.1. Dual algorithm

A λ-dual approximation algorithm (Hochbaum & Shmoys, 1987) takes a number d as an input, and either delivers a schedule with makespan at most λd or answers correctly that there exists no schedule whose makespan is at most d. Hence d is called a guess of the makespan. For scheduling monotonic moldable tasks, a $(3/2)$-dual approximation algorithm with time complexity $O(nm)$ is proposed in Mounié et al. (2007). For the convenience of our exposition, we denote this dual algorithm by Algorithm $\mathbf{D}(\mathcal{T},m,d)$. For a makespan-minimization problem, an *estimation* algorithm (Jansen & Land, 2018) with estimation ratio ρ computes a value w that estimates the optimal makespan within an estimation ratio ρ, i.e., $w\le C^*_{\max}\le\rho w$. Given an estimation algorithm with time complexity $f(n,m)$ and a λ-dual algorithm with time complexity $g(n,m)$, using binary search for a correct guess of the makespan within any specified precision $\epsilon>0$, from these two algorithms one can develop a $\lambda(1+\epsilon)$-approximation algorithm with running time $O(f(n,m)+g(n,m)\log(1/\epsilon))$. For scheduling monotonic tasks, as mentioned before, we have a 2-approximation algorithm with time complexity $O(n\log^2 m)$ (Ludwig & Tiwari, 1994), which is actually an estimation algorithm with estimation ratio 2. Therefore, the dual algorithm $\mathbf{D}(\mathcal{T},m,d)$ can be converted to a $(3/2+\epsilon)$-approximation algorithm with time complexity $O(nm\log(n/\epsilon))$.

Given a number h, we define for each task T_i its *canonical number of processors*, $\gamma(i,h)$, as the minimal number of processors needed to execute task T_i in time at most h. We set by convention $\gamma(i,h)=+\infty$ if T_i cannot be executed in time at most h on m processors. Two important technical results established in Mounié et al. (2007) are stated in the following two lemmas.

Lemma 1 (Mounié et al., 2007). *Given any fixed number d, define $\mathcal{T}_S(d)=\{T_i\in\mathcal{T} : t_{i,1}\le d\}$ and $\mathcal{T}_L(d)=\mathcal{T}\backslash\mathcal{T}_S(d)$. Let $W_S(d)=\sum_{T_i\in\mathcal{T}_S(d)}t_{i,1}$. Define knapsack problem* $\mathrm{KP}(\mathcal{T}_L(d),m,d)$ *as follows:*

$$W(d)=\min_{\mathcal{T}_1\subseteq\mathcal{T}_L(d)}\left(\sum_{T_i\in\mathcal{T}_1}w_{i,\gamma(i,d)}+\sum_{T_i\in\mathcal{T}_L(d)\backslash\mathcal{T}_1}w_{i,\gamma(i,d/2)}\right)$$

subject to $\sum_{T_i\in\mathcal{T}_1}\gamma(i,d)\le m$. *If* $W(d)+W_S(d)>md$, *then* $d<C^*_{\max}$. *Otherwise, Algorithm* $\mathbf{D}(\mathcal{T},m,d)$ *constructs a feasible schedule with makespan at most* $3d/2$.

Lemma 1 helps us to determine whether a guess d is smaller than $C^*_{\max}$. This is frequently used in the first part of our algorithm defined as Algorithm $\mathbf{F}(\mathcal{T},m)$ (see Section 3.1). The details of how Algorithm $\mathbf{D}(\mathcal{T},m,d)$ constructs a feasible schedule of makespan at most $3d/2$ can be found in Mounié et al. (2007).

Lemma 2 (Mounié et al., 2007). *Given any number h, if $\gamma(i,h)<+\infty$, then*

$$h\ge t_{i,\gamma(i,h)}>\frac{\gamma(i,h)-1}{\gamma(i,h)}h.$$

If $\gamma(i,h)\in[2,m]$, *then*

$$2t_{i,\gamma(i,h)}\ge t_{i,\gamma(i,h)-1}>h\ge t_{i,\gamma(i,h)}>\frac{1}{2}h.$$

Lemma 2 will be used in analyzing one part of our $(3/2)$-approximation algorithm in Section 5.

2.2. Algorithm outline

Before we indulge into the algorithmic details, let us outline the main intuitions and ideas behind our algorithm. As concluded in the abstract and related works, the best relevant known results are: (a) an efficient $(3/2+\epsilon)$-approximation algorithm with time complexity $O(nm\log(n/\epsilon))$ (Mounié et al., 2007), (b) an FPTAS for the case of $m\ge 16n/\epsilon$ (Jansen & Land, 2018), and (c) a PTAS with time complexity $O(n^{g(1/\epsilon)})$ (Jansen & Porkolab, 2002; Jansen & Thöle, 2010), where $g(\cdot)$ is a super-exponential function. Concentrated on the case that m is fixed or relatively small compared with n, we are looking for fast and direct algorithms since the PTAS requires high running time. As we target at an efficient approximation algorithm with approximation ratio less than $3/2+\epsilon$, it is

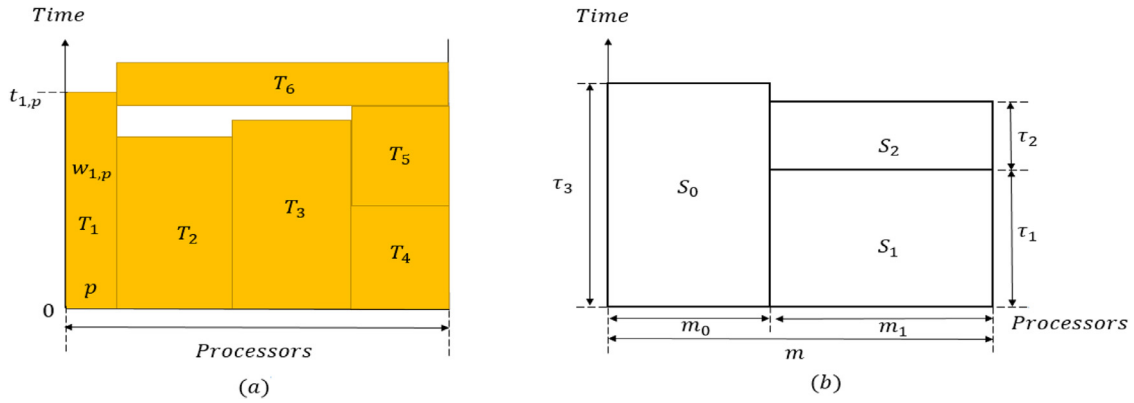

Fig. 1. (a) a schedule in a processor-time diagram; (b) a shelf-based schedule.

natural to consider whether we can remove the ϵ-term in Mounié et al. (2007).

In Mounié et al. (2007), the ϵ-term comes from the binary search of the optimal makespan. Unfortunately, there is no way to remove ϵ in Algorithm $\mathbf{D}(\mathcal{T}, m, d)$ since only an input $d = C^*_{\max} + \epsilon$ always guarantees a feasible schedule with makespan $\frac{3}{2}d$, while an input $d = C^*_{\max} - \epsilon$ cannot for any $\epsilon > 0$.

Motivated by the study in Ludwig & Tiwari (1994), which takes the largest task execution time as the makespan of their schedule, we construct a schedule of a makespan represented by execution times of some tasks. Rather than guessing the exact value of $C^*_{\max}$ as Algorithm $\mathbf{D}(\mathcal{T}, m, d)$ does, our novel strategy here is to find approximate values for $C^*_{\max}$ from some discrete sets of execution times of some tasks, each having a cardinality polynomially bounded by n and m. We first construct three sub-schedules of makespans bounded by these approximate values, and then concatenate these three sub-schedules into a full schedule with a makespan of a desired bound.

3. Algorithm description

As shown in Fig. 1(a), a schedule to moldable task scheduling can be depicted in a processor-time diagram. Each rectangle represents a task T_i, with the height corresponding to the execution time $t_{i,p}$ of the task and the width to the number of p processors allotted to it. Recall that the workload $w_{i,p}$ of a task is the product of p and $t_{i,p}$, which corresponds to the area of the rectangle in the diagram. We define the *workload* of a schedule as the total of workloads of all tasks in the schedule, which corresponds to the total area of all rectangles.

Our algorithm uses a shelf-based approach: Define a *shelf* as a set of processors with available processing period. Define the height of a shelf as the length of available processing period and the width as the number of processors. We construct three shelves S_1, S_2 and S_0 of respective heights τ_1, τ_2 and τ_3, as illustrated in Fig. 1(b). More specifically, define shelf S_1 with m_1 processors and available in period $[0, \tau_1]$, shelf S_2 with m_1 processors and available in period $(\tau_1, \tau_1 + \tau_2]$, and shelf S_0 with m_0 processors and available in period $[0, \tau_3]$. We then try to determine a good processor allotment of all tasks and pack them into these shelves. All values of τ_1, τ_2, τ_3 and m_0, m_1 will be defined and computed accordingly.

Suppose such a shelf-based schedule with makespan $\max\{\tau_1 + \tau_2, \tau_3\}$ exists. As we target for a 3/2 approximation, it is natural to consider setting the values of τ_1, τ_2 and τ_3 in such a way that the makespan of the schedule is no more than $\frac{3}{2}C^*_{\max}$. To this end, we let τ_1, τ_2 and τ_3 be approximate values of $C^*_{\max}$, $\frac{1}{2}C^*_{\max}$ and $\frac{3}{2}C^*_{\max}$, respectively. More specifically, let

$$X = \left\{t_{j,p} : j = 1, \ldots, n;\ p = 1, \ldots, m\right\},$$

$$Y = X \cup \left\{t_{i,1} + t_{j,1}, t_{i,1} + t_{j,2} : i, j = 1, \ldots, n\right\}.$$

and define

$$\tau_1 = \max\{x : x \le C^*_{\max}, x \in X\}, \tag{1}$$

$$\tau_2 = \max\left\{x : x \le \tfrac{1}{2}C^*_{\max}, x \in X \cup \{0\}\right\}, \tag{2}$$

$$\tau_3 = \max\left\{y : y \le \tfrac{3}{2}C^*_{\max}, y \in Y\right\}, \tag{3}$$

It is clear that $|X| \le nm$ and $|Y| \le nm + 2n^2$. Since $C^*_{\max} \ge \min\{x : x \in X\}$, values of (τ_1, τ_2, τ_3) always exist.

Then the following questions arise: As $C^*_{\max}$ is unknown, how can we find the values of τ_1, τ_2 and τ_3? Even if τ_1, τ_2 and τ_3 are found, does there exist such a shelf-based schedule? If yes, how can we construct it? These questions are answered by our (3/2)-approximation algorithm, which we call FA Algorithm and consists of two parts: the first part, written as Procedure $\mathbf{F}(\mathcal{T}, m)$, finds the values of τ_1, τ_2 and τ_3 (see Section 3.1), while the second part, written as Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$, constructs a feasible schedule dependent on the values of τ_1, τ_2 and τ_3 (see Section 3.2).

The FA Algorithm

1. Run Procedure $\mathbf{F}(\mathcal{T}, m)$ to obtain the values of τ_1, τ_2 and τ_3 and, possibly, to output a feasible schedule $\tilde{\sigma}$ of makespan at most $\frac{3}{2}\tau_0$ (see Section 3.1).
2. Given input (τ_1, τ_2, τ_3), run Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ to output a feasible schedule σ of makespan $C(\tau_1, \tau_2, \tau_3)$ (see Section 3.2).

End of the FA Algorithm

3.1. Procedure $\mathbf{F}(\mathcal{T}, m)$

The main idea behind this procedure is that one can identify whether a value is smaller than $C^*_{\max}$ by solving a knapsack problem. Take τ_1 as an example. As we know, τ_1 is the largest value no more than $C^*_{\max}$ in X. According to Lemma 1, if $W(d) + W_S(d) > md$, we then obtain $d < C^*_{\max}$. Then Procedure $\mathbf{F}(\mathcal{T}, m)$ sets the value of τ_1 as the largest among all those $x_i \in X$ satisfying $W(x_i) + W_S(x_i) > mx_i$. We discuss and confirm that $\mathbf{F}(\mathcal{T}, m)$ succeeds in finding the right values of τ_1, τ_2 and τ_3 in Section 4.

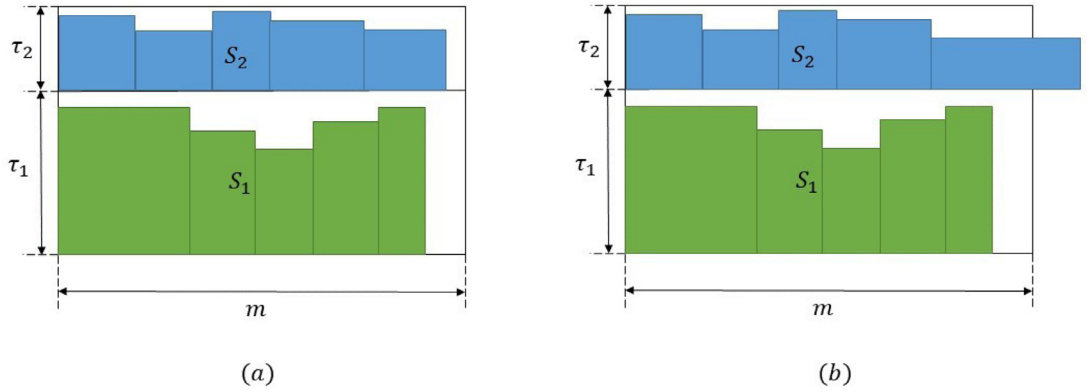

Fig. 2. Possible schedules resulted from Step 2.

Procedure F$(\mathcal{T}, m)$

Step 1. Reindex the elements in $X = \{x_1, \ldots, x_{|X|}\}$ and in $Y = \{y_1, \ldots, y_{|Y|}\}$ in ascending order of their values. Let $\bar{x} = \max\{x : x \in X\}$ and $\bar{y} = \max\{y : y \in Y\}$.

Step 2. Solve knapsack problem $\mathrm{KP}(\mathcal{T}_L(d), m, d)$ with $d = \bar{x}$. If $W(\bar{x}) + W_S(\bar{x}) > m\bar{x}$, set $\tau_1 := \bar{x}$. Otherwise, identify $x_j \in X$ such that $W(x_j) + W_S(x_j) > mx_j$ and $W(x_{j+1}) + W_S(x_{j+1}) \le mx_{j+1}$. (NB: This can be done with binary search, where for each searched value x_i, solve knapsack problem $\mathrm{KP}(\mathcal{T}_L(d), m, d)$ with $d = x_i$. Similar comment applies to the second part of Steps 3 and 4.) Set $\tau_1 := x_j$ and run Algorithm **D**$(\mathcal{T}, m, d)$ with $d = x_{j+1}$ to obtain a feasible schedule $\tilde{\sigma}_1$.

Step 3. Solve knapsack problem $\mathrm{KP}(\mathcal{T}_L(d), m, d)$ with $d = 2\bar{x}$. If $W(2\bar{x}) + W_S(2\bar{x}) > m2\bar{x}$, set $\tau_2 := \bar{x}$. Otherwise, identify $x_k \in X$ such that $W(2x_k) + W_S(2x_k) > 2mx_k$ and $W(2x_{k+1}) + W_S(2x_{k+1}) \le 2mx_{k+1}$. Set $\tau_2 := x_k$ and run Algorithm **D**$(\mathcal{T}, m, d)$ with $d = 2x_{k+1}$ to obtain a feasible schedule $\tilde{\sigma}_2$.

Step 4. Solve knapsack problem $\mathrm{KP}(\mathcal{T}_L(d), m, d)$ with $d = \frac{2}{3}\bar{y}$. If $W(\frac{2}{3}\bar{y}) + W_S(\frac{2}{3}\bar{y}) > m(\frac{2}{3}\bar{y})$, set $\tau_3 := \bar{y}$. Otherwise, identify $y_i \in Y$ such that $W(\frac{2}{3}y_i) + W_S(\frac{2}{3}y_i) > m(\frac{2}{3}y_i)$ and $W(\frac{2}{3}y_{i+1}) + W_S(\frac{2}{3}y_{i+1}) \le m(\frac{2}{3}y_{i+1})$. Set $\tau_3 := y_i$ and run Algorithm **D**$(\mathcal{T}, m, d)$ with $d = \frac{2}{3}y_{i+1}$ to obtain a feasible schedule $\tilde{\sigma}_3$.

Step 5. If $\{\tilde{\sigma}_1, \tilde{\sigma}_2, \tilde{\sigma}_3\} \neq \emptyset$, then let $\tau_0 = \min\{x_{j+1}, 2x_{k+1}, \frac{2}{3}y_{i+1}\}$ (NB: some element(s) may not be applicable) and let $\tilde{\sigma} \in \{\tilde{\sigma}_1, \tilde{\sigma}_2, \tilde{\sigma}_3\}$ be the feasible schedule of the minimum makespan.

End of Procedure F$(\mathcal{T}, m)$

3.2. Procedure **A**$(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$

The main steps of Procedure **A**$(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ are as follows. The first step divides tasks into large and small. The latter tasks are ignored temporarily until the last step, in which they are added to the schedule. In the second step, large tasks are divided into two subsets $\mathcal{T}_1$ and $\mathcal{T}_2$ by solving a knapsack problem. Tasks in subset $\mathcal{T}_1$ are scheduled in shelf S_1 with height τ_1 and the others are scheduled in shelf S_2 with height τ_2. This is illustrated in Fig. 2. If tasks of subset $\mathcal{T}_2$ use more than m processors in shelf S_2, we select some large tasks in $\mathcal{T}_1$ and $\mathcal{T}_2$ to be reallocated to shelf S_0 with height τ_3. This is done in Step 3 and is illustrated in Fig. 3. Finally, all small tasks identified in Step 1 are added into the schedule in Step 4, which is illustrated in Fig. 4.

We refer the sum of execution times of the tasks allotted to a processor as the *load* of the processor. Initially, set $m_1 := m$ and $m_0 := 0$.

Procedure A$(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$

Step 1. Partition the task set $\mathcal{T}$ into two subsets:

$$\mathcal{T}_S = \{T_i \in \mathcal{T} : t_{i,1} \le \tau_2\} \text{ (small tasks)}, \tag{4}$$

$$\mathcal{T}_L = \{T_i \in \mathcal{T} : t_{i,1} > \tau_2\} \text{ (large tasks)}. \tag{5}$$

If $\mathcal{T}_L = \emptyset$, then let σ_1 be the empty schedule and go to Step 4; otherwise, go to Step 2.

Step 2. Solve the knapsack problem $\mathrm{KP}(\mathcal{T}_L, m, \tau_1, \tau_2)$ below for an allotment of tasks in $\mathcal{T}_L$:

$$W(\tau_1, \tau_2) = \min_{\mathcal{T}' \subseteq \mathcal{T}_L} \left(\sum_{T_i \in \mathcal{T}'} w_{i,\gamma(i,\tau_1)} + \sum_{T_i \in \mathcal{T}_L \setminus \mathcal{T}'} w_{i,\gamma(i,\tau_2)} \right)$$

subject to $\sum_{T_i \in \mathcal{T}'} \gamma(i, \tau_1) \le m$. Let $\mathcal{T}' = \mathcal{T}_1$ be an optimal solution to the above problem $\mathrm{KP}(\mathcal{T}_L, m, \tau_1, \tau_2)$. Then allot $\gamma(i, \tau_1)$ processors to any task $T_i \in \mathcal{T}_1$ and $\gamma(i, \tau_2)$ processors to any task $T_i \in \mathcal{T}_2 \equiv \mathcal{T}_L \setminus \mathcal{T}_1$. Schedule all tasks in $\mathcal{T}_1$ in shelf S_1 and all tasks in $\mathcal{T}_2$ in shelf S_2. If $\sum_{T_i \in \mathcal{T}_2} \gamma(i, \tau_2) \le m$ (as illustrated in Fig. 2(a)), let S_1 and S_2 use the same m processors. The resulting schedule is denoted by σ_2 and go to Step 4. If $\sum_{T_i \in \mathcal{T}_2} \gamma(i, \tau_2) > m$ (as illustrated in Fig. 2(b)), go to Step 3.

Step 3. Transform tasks in S_1 and S_2 to S_0 or S_1 in the following ways.

Step 3.1. For any task T_i in S_1 with $\gamma(i, \tau_1) > 1$ and $t_{i,\gamma(i,\tau_1)-1} \le \tau_3$, let $m_0 := m_0 + \gamma(i, \tau_1) - 1$, $m_1 := m_1 - \gamma(i, \tau_1) + 1$, and reallocate T_i to the $\gamma(i, \tau_1) - 1$ processors in shelf S_0.

Step 3.2. While there exists at least one pair of $\{T_i, T_j\}$ in S_1 such that $\gamma(i, \tau_1) = \gamma(j, \tau_1) = 1$ and $t_{i,1} + t_{j,1} \le \tau_3$, pick such a pair $\{T_i, T_j\}$ and do the following transformation: Let $m_0 := m_0 + 1$, $m_1 := m_1 - 1$, and reschedule T_i and T_j sequentially on the newly added processor in S_0.
Repeat Step 3.2 if after transforming the pair $\{T_i, T_j\}$ there still exists at least one such pair.

Step 3.3. While there exists at least one pair of $\{T_i, T_j\}$ in S_1 such that $\gamma(i, \tau_1) = 1$ and $\gamma(j, \tau_1) = 2$ and $t_{i,\gamma(i,\tau_1)} + t_{j,\gamma(j,\tau_1)} \le \tau_3$, pick such a pair $\{T_i, T_j\}$ and do the following transformation: Change one processor used by T_j from S_1 to S_0 and reschedule T_i after T_j on this processor and let $m_0 := m_0 + 1$ and $m_1 := m_1 - 1$.(NB: In this

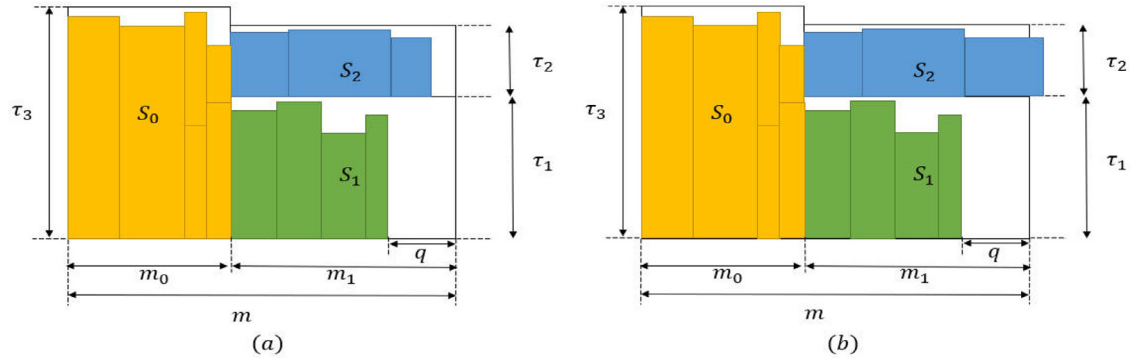

Fig. 3. Possible schedules resulted from Step 3.

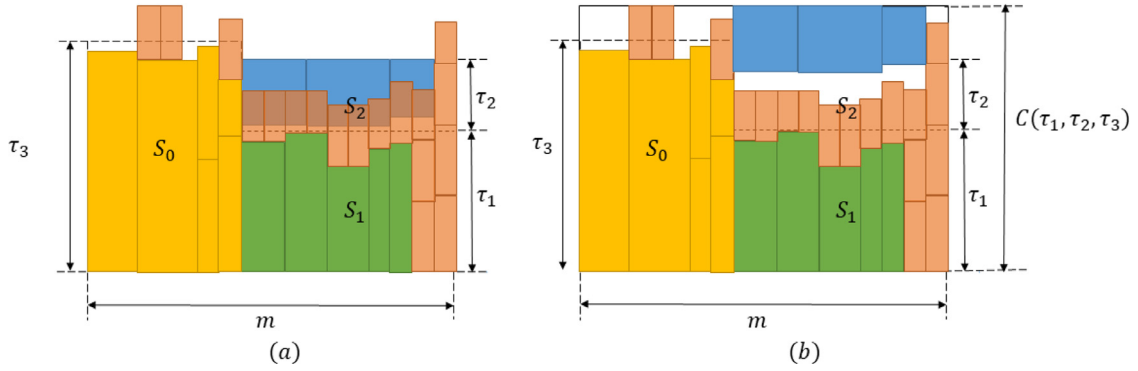

Fig. 4. Schedule σ obtained after Step 4.

case, T_j uses one processor in S_0 and one processor in S_1.)

Repeat Step 3.3 if after transforming the pair $\{T_i, T_j\}$ there still exists at least one such pair.

Step 3.4. Let q be the number of idle processors in S_1. For each task T_i in S_2 with $\gamma(i, \tau_3) \le q$, reallocate T_i to $\gamma(i, \tau_3)$ processors in S_0 if $t_{i,\gamma(i,\tau_3)} > \tau_1$ and to S_1 otherwise. Let $m_0 := m_0 + \gamma(i, \tau_3)$ and $m_1 := m_1 - \gamma(i, \tau_3)$, if T_i is reallocated to S_0.

Step 3.5. If some task was reallocated to S_1 from S_2 in Step 3.4, then go back to Step 3.2.

Step 3.6. Regardless of whether shelf S_2 uses more than m_1 processors or not, let S_1 and S_2 use the same m_1 processors, which together with S_0 form a schedule denoted by σ_3. If S_2 uses more than m_1 processors (as illustrated in Fig. 3(b)), then terminate with output "exit" (due to infeasibility of σ_3). Otherwise (as illustrated in Fig. 3(a)), go to Step 4 (with feasible σ_3).

Step 4. On the basis of partial schedule σ_1, σ_2, or σ_3 of large tasks only, add small tasks in $\mathcal{T}_S$ (if any) to the schedule in the following ways to produce a full schedule σ of makespan $C(\tau_1, \tau_2, \tau_3) < +\infty$.

Step 4.1. Reindex tasks $T_i \in \mathcal{T}_S$ in the order of non-increasing execution time of $t_{i,1}$.

Step 4.2. For $i = 1, \ldots, |\mathcal{T}_S|$, iteratively allocate task T_i in $\mathcal{T}_S$ to the least loaded processor and update the processor load after each job allocation.

Step 4.3. Compute the total load $C(\tau_1, \tau_2, \tau_3)$ of the most loaded processor.

Step 4.4. Postpone all tasks in S_2 (if any), so that their completion times are equal to $C(\tau_1, \tau_2, \tau_3)$. Small tasks allocated to a processor can be scheduled in any order between tasks in S_1 and S_2 (if any) or after tasks in S_0 (if any) scheduled in the processor. (See Fig. 4 for illustration.)

End of Procedure A$(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$

4. Analysis of procedure F$(\mathcal{T}, m)$

Now let us discuss and confirm that Procedure $\mathbf{F}(\mathcal{T}, m)$ succeeds in identifying the right values of τ_1, τ_2 and τ_3, unless schedule $\tilde{\sigma}$ has a makespan at most $\frac{3}{2}C^*_{\max}$. Let us first focus on Step 2 of the procedure, which aims at finding the value of τ_1.

Case (a). Suppose $W(\bar{x}) + W_S(\bar{x}) > m\bar{x}$. According to Lemma 1, we obtain $C^*_{\max} > \bar{x}$. Then by the definition of τ_1 given in (1) and $\bar{x}$, we have $\tau_1 = \bar{x}$, which implies Procedure $\mathbf{F}(\mathcal{T}, m)$ succeeds in finding τ_1.

Case (b). Suppose $W(\bar{x}) + W_S(\bar{x}) \le m\bar{x}$. Then there exists a pair (x_j, x_{j+1}) such that $W(x_j) + W_S(x_j) > mx_j$ and $W(x_{j+1}) + W_S(x_{j+1}) \le mx_{j+1}$. According to Lemma 1, we have $x_j < C^*_{\max}$ and the schedule $\tilde{\sigma}_1$ constructed in Step 2 of Procedure $\mathbf{F}(\mathcal{T}, m)$ is a feasible schedule with makespan at most $\frac{3}{2}x_{j+1}$. Then we discuss the relationship between x_{j+1} and $C^*_{\max}$.

(i) If $x_{j+1} \le C^*_{\max}$, by the definition of τ_1 it is not correct to set $\tau_1 := x_j$, which implies Procedure $\mathbf{F}(\mathcal{T}, m)$ fails to find τ_1. However, the makespan of the feasible schedule is at most $\frac{3}{2}x_{j+1} \le \frac{3}{2}C^*_{\max}$. (ii) If $x_{j+1} > C^*_{\max}$, by the definition of τ_1 and the fact that $x_j < C^*_{\max}$, it is correct to set $\tau_1 := x_j$, which implies Procedure $\mathbf{F}(\mathcal{T}, m)$ succeeds in finding τ_1.

In conclusion, Procedure $\mathbf{F}(\mathcal{T}, m)$ either finds the right value of τ_1 or a feasible schedule with makespan at most $\frac{3}{2}C^*_{\max}$. The following lemma hence follows.

Lemma 3. *In $O(nm\log(nm))$ time, Procedure $\mathbf{F}(\mathcal{T}, m)$ either finds the right value of τ_1 or a feasible schedule with makespan at most $\frac{3}{2}C^*_{\max}$.*

Proof. Sorting elements in X needs $O(nm\log(nm))$ time. The number of binary searches is bounded by $\log(nm)$ and for each binary search we need to solve a knapsack problem in $O(nm)$. On the other hand, the feasible schedule can be generated by Algorithm $\mathbf{D}(\mathcal{T}, m, d)$ in $O(nm)$ time, from which the lemma follows. □

Similar arguments to those in the proof of Lemma 3 apply to the analysis of Steps 3 and 4 of Procedure $\mathbf{F}(\mathcal{T}, m)$ and lead to the following two lemmas.

Lemma 4. *In $O(nm\log(nm))$ time, Procedure $\mathbf{F}(\mathcal{T}, m)$ either finds the value of τ_2 or a feasible (full) schedule with makespan at most $\frac{3}{2}C^*_{\max}$.* □

Lemma 5. *In $O(nm + n^2)\log(nm + n^2))$ time, Procedure $\mathbf{F}(\mathcal{T}, m)$ either finds the value of τ_3 or a (full) feasible schedule with makespan at most $\frac{3}{2}C^*_{\max}$.* □

Combining Lemmas 3–5, we obtain our first theorem.

Theorem 1. *Procedure $\mathbf{F}(\mathcal{T}, m)$ either finds a feasible schedule within a factor 3/2 of optimum, or outputs the correct values of τ_1, τ_2 and τ_3. It runs in $O(nm\log(nm))$ time for $m > n$ and in $O(n^2\log n)$ time for $m \le n$.*

5. Analysis of procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$

In this section we analyze Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ presented in Section 3.2. As one can see in our FA Algorithm, Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ constructs a schedule dependent on the values of τ_1, τ_2 and τ_3 given by Procedure $\mathbf{F}(\cdot)$. Note that $\mathbf{F}(\cdot)$ may find wrong values for parameters τ_1, τ_2 and τ_3. Taken incorrect values as input, Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ may deliver an infeasible schedule, or may exit at Step 3.6, and the makespan of the final schedule may exceed $\frac{3}{2}C^*_{\max}$. However, Theorem 1 allows us either (a) to use the schedule produced by procedure $\mathbf{F}(\cdot)$ as our final output or (b) to assume that $\mathbf{F}(\cdot)$ computes the values τ_1, τ_2 and τ_3 correctly. Therefore, in this section, we assume that the values of τ_1, τ_2 and τ_3 used in Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ are correct as defined. With this assumption, we establish that Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ delivers a feasible schedule σ with makespan $C(\tau_1, \tau_2, \tau_3) \le \frac{3}{2}C^*_{\max}$.

As one can easily see from Procedure $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$, there is exactly one of σ_1, σ_2 and σ_3 when the procedure enters Step 4. For convenience of our exposition, we denote the only one partial schedule by σ_0. We first establish in Section 5.1 that if the partial schedule σ_0 satisfies the following three conditions, then the final full schedule σ is feasible and its makespan $C(\tau_1, \tau_2, \tau_3)$ is at most $\frac{3}{2}C^*_{\max}$. Then we show in Sections 5.2.1–5.2.3 that σ_0 indeed satisfies the following three conditions, where $W_S = \sum_{T_i \in \mathcal{T}_S} t_{i,1}$ denotes the sum of the workloads of tasks in $\mathcal{T}_S$ when each task in $\mathcal{T}_S$ is allotted to one processor:

- **Feasibility:** Each processor executes at most one task at a time; all the processors allotted to a task execute the task simultaneously without preemption; at any time point, the number of non-idle processors does not exceed m.
- **Workload:** The schedule workload is at most $mC^*_{\max} - W_S$.
- **Makespan:** The makespan is at most $\frac{3}{2}C^*_{\max}$.

For convenience, a (partial) schedule that satisfies the above three conditions will be called a *good* schedule.

According to the monotonicity assumption, workload of a task is minimum when it is allocated to a single processor, which implies that in any schedule the total workload of small tasks is at least W_S. Let W^* denote the workload of an optimal schedule for all tasks of $\mathcal{T}$. It follows that $W_S \le W^* \le mC^*_{\max}$.

5.1. Quality of full schedule σ

Let us establish the following result as our starting point.

Lemma 6. *If σ_0 is a good schedule, then schedule σ is feasible with makespan at most $\frac{3}{2}C^*_{\max}$.*

Proof. Note that in schedule σ_0 each processor is scheduled to execute at most two tasks, each of which may use one or more than one processor. The two tasks (if any) are executed as the first and the last task in the processor. In Step 4, each small task is scheduled to a single existing processor between two tasks (if any). No new processor is added. Therefore, Step 4 always generates a feasible schedule σ if σ_0 is feasible.

Notice that in Lemma 6 σ_0 is a good schedule and hence the workload of σ_0 is at most $mC^*_{\max} - W_S$ according to the definition of "goodness". Adding small tasks into σ_0 forms σ. Since the total workload of small tasks added by Procedure $\mathbf{A}(\cdot)$ is exactly W_S, the total workload of σ is at most $mC^*_{\max}$.

On the makespan of σ, suppose to the contrary that its makespan is more than $\frac{3}{2}C^*_{\max}$. Consider a processor that is loaded until the makespan point. Then the last task scheduled onto the processor denoted by T_p must be a small task defined in (4), because σ_0 satisfies the makespan condition. Note that the small task T_p has a processing time at most $\frac{1}{2}C^*_{\max}$ due to the definition of τ_2 given in (2) and its completion time is $\frac{3}{2}C^*_{\max}$. Then the load of the processor before scheduling task T_p must be larger than $C^*_{\max}$. According to Step 4.2, the processor was the least loaded before the task T_p is scheduled. Therefore, we conclude that all processors have their loads greater than $C^*_{\max}$. In other words, the workload of σ is greater than $mC^*_{\max}$, which contradicts that the total workload of σ is at most $mC^*_{\max}$. □

5.2. Goodness of partial schedule σ_0

Given that exactly one of σ_1, σ_2 and σ_3 exists, which we denote by partial schedule σ_0, we consider that σ_0 is respectively σ_1, σ_2 and σ_3 in Sections 5.2.1–5.2.3, and then show the goodness of σ_1, σ_2 and σ_3, respectively. Before we show the goodness of partial schedule σ_0, let us first establish some properties for tasks.

Lemma 7. *For any task T_i, we have $\gamma(i, \tau_1) = \gamma(i, C^*_{\max})$, $\gamma(i, \tau_2) = \gamma(i, \frac{1}{2}C^*_{\max})$, and $\gamma(i, \tau_3) = \gamma(i, \frac{3}{2}C^*_{\max})$.*

Proof. For any task T_i, we have $\gamma(i, C^*_{\max}) \le \gamma(i, \tau_1)$ since $\tau_1 \le C^*_{\max}$. Suppose there exists a task T_j such that $\gamma(j, C^*_{\max}) < \gamma(j, \tau_1)$. This implies $C^*_{\max} \ge t_{j,\gamma(j,C^*_{\max})} \ge t_{j,\gamma(j,\tau_1)-1} > \tau_1$, which contradicts the definition of τ_1. The same argument can be applied on τ_2 and τ_3. □

Corollary 1. *For any task T_i, we have $t_{i,\gamma(i,\tau_1)} = t_{i,\gamma(i,C^*_{\max})}$, $t_{i,\gamma(i,\tau_2)} = t_{i,\gamma(i,\frac{1}{2}C^*_{\max})}$, and $t_{i,\gamma(i,\tau_3)} = t_{i,\gamma(i,\frac{3}{2}C^*_{\max})}$; on the other hand, we also have $w_{i,\gamma(i,\tau_1)} = w_{i,\gamma(i,C^*_{\max})}$, $w_{i,\gamma(i,\tau_2)} = w_{i,\gamma(i,\frac{1}{2}C^*_{\max})}$ and $w_{i,\gamma(i,\tau_3)} = w_{i,\gamma(i,\frac{3}{2}C^*_{\max})}$.* □

5.2.1. Schedule $\sigma_0 = \sigma_1$

Let us show in this subsection that schedule σ_1 is good. According to the algorithm, schedule σ_1 is empty. Hence we only need to check the workload condition. As we already have $mC^*_{\max} \ge W^* \ge W_S$, the workload condition is then satisfied by the empty schedule σ_1 with zero workload. Consequently, we have the following lemma.

Lemma 8. *If there is no large task in $\mathcal{T}$, then $mC^*_{\max} - W_S \ge 0$.* □

5.2.2. Schedule $\sigma_0 = \sigma_2$

Now let us consider the goodness of schedule σ_2, which is delivered by the algorithm in its Step 2 for large tasks $\mathcal{T}_L$ with a makespan at most $\tau_1 + \tau_2$. Since $\tau_1 \le C^*_{\max}$ and $\tau_2 \le \frac{1}{2}C^*_{\max}$, the schedule satisfies makespan condition. According to the algorithm, σ_2 satisfies the feasibility condition. Now consider the workload condition.

According to Lemma 7 and Corollary 1, problem $\mathrm{KP}(\mathcal{T}_L, m, \tau_1, \tau_2)$ in Step 2 of the procedure is equivalent to the following knapsack problem $\mathrm{KP}(\mathcal{T}_L, m, C^*_{\max}, \frac{1}{2}C^*_{\max})$, i.e., $W(\tau_1, \tau_2) = W(C^*_{\max}, \frac{1}{2}C^*_{\max})$:

$$W\left(C^*_{\max}, \frac{1}{2}C^*_{\max}\right) = \min_{\mathcal{T}' \subseteq \mathcal{T}_L} \left(\sum_{T_i \in \mathcal{T}'} w_{i,\gamma(i,C^*_{\max})} + \sum_{T_i \in \mathcal{T}_L \setminus \mathcal{T}'} w_{i,\gamma(i,\frac{1}{2}C^*_{\max})} \right)$$

subject to $\sum_{T_i \in \mathcal{T}'} \gamma(i, C^*_{\max}) \le m$.

Fix any optimal schedule of our problem. In an optimal schedule, let p_i denote the number of processors allotted to task T_i. Let $\mathcal{T}^*_1 = \{T_i \in \mathcal{T}_L : t_{i,p_i} > \frac{1}{2}C^*_{\max}\}$. Clearly, we have $\sum_{T_i \in \mathcal{T}^*_1} p_i \le m$ as the optimal schedule is first feasible. Since $p_i \ge \gamma(i, C^*_{\max})$ for $T_i \in \mathcal{T}^*_1$ according to the definition of $\gamma(i, C^*_{\max})$, we have $\sum_{T_i \in \mathcal{T}^*_1} \gamma(i, C^*_{\max}) \le m$. Therefore, $\mathcal{T}^*_1$ is a feasible solution of knapsack problem $\mathrm{KP}(\mathcal{T}_L, m, C^*_{\max}, \frac{1}{2}C^*_{\max})$.

In an optimal schedule, the total workload of $\mathcal{T}_S$ is at least W_S, which implies that the total workload of the remaining large tasks is at most $W^* - W_S$: $\sum_{T_i \in \mathcal{T}_L} w_{i,p_i} \le W^* - W_S$. For any task $T_i \in \mathcal{T}^*_1$, we have $p_i \ge \gamma(i, C^*_{\max})$ and hence $w_{i,p_i} \ge w_{i,\gamma(i,C^*_{\max})}$. For any task $T_i \in \mathcal{T}_L \setminus \mathcal{T}^*_1$, we have $t_{i,p_i} \le \frac{1}{2}C^*_{\max}$ and hence $p_i \ge \gamma(i, \frac{1}{2}C^*_{\max})$, which implies $w_{i,p_i} \ge w_{i,\gamma(i,\frac{1}{2}C^*_{\max})}$. We then have $\sum_{T_i \in \mathcal{T}_L} w_{i,p_i} = \sum_{T_i \in \mathcal{T}^*_1} w_{i,p_i} + \sum_{T_i \in \mathcal{T}_L \setminus \mathcal{T}^*_1} w_{i,p_i} \ge \sum_{T_i \in \mathcal{T}^*_1} w_{i,\gamma(i,C^*_{\max})} + \sum_{T_i \in \mathcal{T}_L \setminus \mathcal{T}^*_1} w_{i,\gamma(i,\frac{1}{2}C^*_{\max})}$. As $\mathcal{T}^*_1$ is a feasible solution of knapsack problem $\mathrm{KP}(\mathcal{T}_L, m, C^*_{\max}, \frac{1}{2}C^*_{\max})$ and $W(C^*_{\max}, \frac{1}{2}C^*_{\max})$ is the optimal value of the knapsack problem, we then have

$$\begin{aligned} W\left(C^*_{\max}, \frac{1}{2}C^*_{\max}\right) &\le \sum_{T_i \in \mathcal{T}^*_1} w_{i,\gamma(i,C^*_{\max})} + \sum_{T_i \in \mathcal{T}_L \setminus \mathcal{T}^*_1} w_{i,\gamma(i,\frac{1}{2}C^*_{\max})} \\ &\le \sum_{T_i \in \mathcal{T}^*_1} w_{i,p_i} + \sum_{T_i \in \mathcal{T}_L \setminus \mathcal{T}^*_1} w_{i,p_i} = \sum_{T_i \in \mathcal{T}_L} w_{i,p_i} \le W^* - W_S. \end{aligned}$$

Then, we have the following lemma with $W^* \le mC^*_{\max}$.

Lemma 9. *Any optimal solution to knapsack problem $\mathrm{KP}(\mathcal{T}_L, m, \tau_1, \tau_2)$ satisfies $W(\tau_1, \tau_2) \le mC^*_{\max} - W_S$.* □

The total workload of any schedule constructed at Step 2 is $W(\tau_1, \tau_2)$. As a direct consequence of Lemma 9, schedule σ_2 satisfies the workload condition.

5.2.3. Schedule $\sigma_0 = \sigma_3$

It is clear that schedule σ_3, which is constructed in Step 3 of the algorithm, has a makespan at most $\max\{\tau_1 + \tau_2, \tau_3\}$, which implies that the makespan condition is satisfied by σ according to the definition of τ_1, τ_2 and τ_3. We know the total workload of any schedule constructed in Step 2 is $W(\tau_1, \tau_2)$, which is at most $mC^*_{\max} - W_S$ according to Lemma 9. Note in Step 3, the algorithm can only decrease the number of processors of some tasks, and the total workload of the schedule decreases accordingly due to the monotonicity assumption. Therefore, schedule σ_3 satisfies the workload condition.

In schedule σ_3, m_0 is the number of processors allotted to tasks in S_0 and $m_1 = m - m_0$. Notice that in Step 3, the number of used processors in S_1 does not exceed m_1, but the number of used processors in S_2 can exceed m_1. Then if tasks in shelf S_2 use at most m_1 processors, schedule σ_3 is feasible (as illustrated in Fig. 3(a)). In what follows, we show that this is indeed the case.

The idea of our proof is by contradiction. Assume tasks in S_2 use more than m_1 processors in schedule σ_3. First, we obtain a lower bound on the workload of tasks in S_0. Then we establish some properties of tasks in S_1 and S_2. These properties help us to find some lower bounds on the workloads of tasks in S_1 and S_2. Using the workload condition for schedule σ_3, we then obtain a contradiction.

Lemma 10. *In schedule σ_3, the total workload in S_0 is greater than $m_0C^*_{\max}$.*

Proof. Since all processors in S_0 are scheduled in Step 3, we consider tasks scheduled in those processors in different cases.

For task T_i in Step 3.1, we have $t_{i,\gamma(i,\tau_1)-1} = t_{i,\gamma(i,C^*_{\max})-1} > C^*_{\max}$ according to the definition of the canonical number of processors and Lemma 7.

For tasks T_i and T_j in Step 3.2, we have $t_{i,\gamma(i,\tau_1)} = t_{i,1}$ and $t_{j,\gamma(j,\tau_1)} = t_{j,1}$. As $T_i \in \mathcal{T}_L$ and $T_j \in \mathcal{T}_L$, we then have $t_{i,1} > \tau_2$ and $t_{j,1} > \tau_2$ according to the definition of $\mathcal{T}_L$ given in (5). Since $t_{i,1} \in X$ and τ_2 is the largest value in X but no more than $\frac{1}{2}C^*_{\max}$, then $t_{i,1} > \frac{1}{2}C^*_{\max}$. Similarly, we can obtain $t_{j,1} > \frac{1}{2}C^*_{\max}$. Hence, $t_{i,1} + t_{j,1} > \frac{1}{2}C^*_{\max} + \frac{1}{2}C^*_{\max} = C^*_{\max}$.

For tasks T_i and T_j in Step 3.3, we have $t_{i,\gamma(i,\tau_1)} = t_{i,1}$ and $t_{j,\gamma(j,\tau_1)} = t_{j,2}$. Since $T_i \in \mathcal{T}_L$, then $t_{i,1} > \tau_2$ according to the definition of $\mathcal{T}_L$. Since $t_{i,1} \in X$ and τ_2 is the largest value in X but no more than $\frac{1}{2}C^*_{\max}$, then $t_{i,1} > \frac{1}{2}C^*_{\max}$. Also, we have $t_{j,\gamma(j,\tau_1)} = t_{j,\gamma(j,C^*_{\max})} > \frac{1}{2}C^*_{\max}$ by Corollary 1 and Lemma 2. Hence, $t_{i,1} + t_{j,2} > \frac{1}{2}C^*_{\max} + \frac{1}{2}C^*_{\max} = C^*_{\max}$.

For task T_i in Step 3.4, because of the test in Step 3.4, we have $t_{i,\gamma(i,\tau_3)} > \tau_1$. Since $t_{i,\gamma(i,\tau_3)} \in X$ and τ_1 is the largest value in X but no more than $C^*_{\max}$, we then have $t_{i,\gamma(i,\tau_3)} > C^*_{\max}$.

Therefore, the workload of any processor in S_0 is larger than $C^*_{\max}$, which implies the workload in S_0 is greater than $m_0C^*_{\max}$. □

Corollary 2. *In schedule σ_3, the total workload of tasks in S_1 and S_2 is bounded by $m_1C^*_{\max} - W_S$.* □

According to Lemma 10, the total workload of tasks in S_0 is greater than $mC^*_{\max}$ if $m_1 = 0$, which is a contradiction to the workload constraint. We then assume $m_1 \ge 1$ in the remainder of this subsection. Let us establish more properties of tasks scheduled in S_1 and S_2.

Lemma 11. *Schedule σ_3 has the following properties:*

(a) *Any task in S_1 uses at most two processors, i.e., $\gamma(i, \tau_1) \in \{1, 2\}$ for T_i in shelf S_1.*
(b) *Any task T_i with $\gamma(i, \tau_1) = 2$ in S_1 has $t_{i,\gamma(i,\tau_1)} > \frac{3}{4}C^*_{\max}$.*
(c) *Among all the tasks T_i with $\gamma(i, \tau_1) = 1$ in S_1, there exists at most one task with execution time less than or equal to $\frac{3}{4}C^*_{\max}$.*
(d) *Any task in S_2 uses at least two processors and its execution time is larger than $\frac{1}{4}C^*_{\max}$.*
(e) *The workload of any task in S_2 is larger than $\frac{3}{2}qC^*_{\max}$, where q is the number of idle processors in S_1.*

Proof. (a) Suppose to the contrary that there exists a task T_i in S_1 such that $\gamma(i, \tau_1) \ge 3$. According to the monotonicity assumption and the definition of τ_1, we have $w_{i,\gamma(i,\tau_1)-1} \le w_{i,\gamma(i,\tau_1)} \le \tau_1\gamma(i, \tau_1) \le C^*_{\max}\gamma(i, \tau_1)$. Therefore,

$$t_{i,\gamma(i,\tau_1)-1} = \frac{w_{i,\gamma(i,\tau_1)-1}}{\gamma(i,\tau_1) - 1} \le \frac{C^*_{\max}\gamma(i,\tau_1)}{\gamma(i,\tau_1) - 1},$$

which is at most $\frac{3}{2}C^*_{\max}$ when $\gamma(i, \tau_1) \ge 3$. Hence, task T_i can be rescheduled to S_0 in Step 3.1.

(b) Since T_i cannot be rescheduled to S_0, we have $t_{i,\gamma(i,\tau_1)-1} > \tau_3$. As $t_{i,\gamma(i,\tau_1)-1} \in Y$ and τ_3 is the largest value in Y but no more

than $\frac{3}{2}C^*_{\max}$, we must have $t_{i,\gamma(i,\tau_1)-1} > \frac{3}{2}C^*_{\max}$. Then,

$$w_{i,\gamma(i,\tau_1)} = \gamma(i,\tau_1)t_{i,\gamma(i,\tau_1)} = 2t_{i,\gamma(i,\tau_1)} \geq w_{i,\gamma(i,\tau_1)-1} = (\gamma(i,\tau_1)-1)t_{i,\gamma(i,\tau_1)-1} > \tfrac{3}{2}C^*_{\max}.$$

Hence, $t_{i,\gamma(i,\tau_1)} > \frac{3}{4}C^*_{\max}$.

(c) If there exist two tasks T_i and T_j with $t_{i,1} \leq \frac{3}{4}C^*_{\max}$ and $t_{j,1} \leq \frac{3}{4}C^*_{\max}$, then these two tasks can be scheduled in S_0 in Step 3.2 since $t_{i,1} + t_{j,1} \leq \tau_3 \leq \frac{3}{2}C^*_{\max}$ by the definition of τ_3.

(d) According to the definition of large tasks and Lemma 2, the claim holds.

(e) Since any task in S_2 can not be rescheduled to S_0 or S_1, any task T_i in S_2 has $\gamma(i,\tau_3) > q$, which implies $t_{i,q} > \tau_3$. As $t_{i,q} \in Y$ and τ_3 is the largest value in Y but no more than $\frac{3}{2}C^*_{\max}$, we must have $t_{i,q} > \frac{3}{2}C^*_{\max}$. Due to monotony, we have $w_{i,\gamma(i,\tau_2)} \geq w_{i,\gamma(i,\tau_3)} \geq w_{i,q} = qt_{i,q} > \frac{3}{2}qC^*_{\max}$. □

In schedule σ_3, let W_1 denote the workload of tasks in S_1 and W_2 denote the workload of tasks in S_2. If $m_1 = q$, we have $W_1 = 0$ and $W_2 > \frac{3}{2}m_1C^*_{\max}$ by (e) of Lemma 11, which implies $W_1 + W_2 > m_1C^*_{\max}$. This contradicts that $W_1 + W_2 \leq m_1C^*_{\max} - W_S$ according to Corollary 2. Therefore, we assume $m_1 - q \geq 1$ in the remainder of this subsection.

If schedule σ_3 is not feasible, number of processors used in S_2 is more than m_1. Lemmas 12 and 13 deliver some lower bounds of the workload of tasks in S_1 and S_2. Lemma 14 shows that this leads to a contradiction.

Lemma 12. *If tasks in S_2 use at least $m_1 + 1$ processors in schedule σ_3, then the workload W_1 of tasks in S_1 is larger than $\frac{3}{4}(m_1 - q)C^*_{\max}$, while the workload W_2 of tasks in S_2 is larger than $\frac{1}{4}(m_1 + 1)C^*_{\max}$.*

Proof. We have $W_2 > \frac{1}{4}(m_1 + 1)C^*_{\max}$ since any task in S_2 has an execution time larger than $\frac{1}{4}C^*_{\max}$ by (d) in Lemma 11 and tasks in S_2 use at least $m_1 + 1$ processors.

In S_1, if all tasks have execution times larger than $\frac{3}{4}C^*_{\max}$, we have $W_1 > \frac{3}{4}(m_1 - q)C^*_{\max}$.

Then we concentrate on the case that there exists one task T_i in S_1 such that $t_{i,\gamma(i,\tau_1)} \leq \frac{3}{4}C^*_{\max}$. According to (a), (b) and (c) of Lemma 11, T_i is the only task in S_1 with execution time less than or equal to $\frac{3}{4}C^*_{\max}$. And we have $\gamma(i,\tau_1) = 1$. We branch into several subcases. Recall that $m_1 - q \geq 1$.

If $m_1 - q = 1$ and $q = 0$, i.e., task T_i is the only task in S_1, we have $W_1 > \frac{1}{2}C^*_{\max}$ and $W_2 > \frac{1}{4}(m_1 + 1)C^*_{\max} = \frac{1}{2}C^*_{\max}$. Hence, $W_1 + W_2 > C^*_{\max}$, which contradicts $W_1 + W_2 \leq m_1C^*_{\max} - W_S$ by Corollary 2.

If $m_1 - q = 1$ and $q \geq 1$, we have $W_1 > \frac{1}{2}C^*_{\max}$ and $W_2 > \frac{3}{2}qC^*_{\max} = \frac{3}{2}(m_1 - 1)C^*_{\max}$ by (e) of Lemma 11. Then, $W_1 + W_2 > (\frac{3}{2}m_1 - 1)C^*_{\max} = (m_1 + \frac{1}{2}m_1 - 1)C^*_{\max} \geq m_1C^*_{\max}$ as $m_1 \geq 2$. This also contradicts $W_1 + W_2 \leq m_1C^*_{\max} - W_S$.

Therefore, we must have $m_1 - q \geq 2$, i.e., there exists at least another task T_j scheduled in S_1 with T_i. According to (a) in Lemma 11, we have $\gamma(j,\tau_1) = 1$ or 2. If $\gamma(j,\tau_1) = 1$, we have $t_{i,1} + t_{j,1} > \tau_3$. By the definition of τ_3, $t_{i,1} + t_{j,1} > \frac{3}{2}C^*_{\max}$, which implies that the average load of the two processors executing T_i and T_j is larger than $\frac{3}{4}C^*_{\max}$. If $\gamma(j,\tau_1) = 2$, then $t_{i,1} + t_{j,2} > \tau_3$. We also have $t_{i,1} + t_{j,2} > \frac{3}{2}C^*_{\max}$. Then the average load of the processors executing T_i and T_j is also larger than $\frac{3}{4}C^*_{\max}$. And all non-idle processor(s) in S_1 execute task(s) with execution time larger than $\frac{3}{4}C^*_{\max}$. Then, we obtain $W_1 > \frac{3}{4}(m_1 - q)C^*_{\max}$. □

Lemma 13. *If tasks in S_2 use at least $m_1 + 1$ processors in schedule σ_3, the workload W_2 of tasks in S_2 is greater than $\frac{3}{2}qkC^*_{\max}$ and $\frac{1}{2}(m_1 + 1 - k)C^*_{\max}$, where k is the number of tasks in S_2.*

Proof. We have

$$W_2 > \frac{3}{2}qkC^*_{\max}, \tag{6}$$

since any task in S_2 has a work area larger than $\frac{3}{2}qC^*_{\max}$ by (e) in Lemma 11. Due to the monotonicity assumption and Corollary 1, task T_i in S_2 has a workload $w_{i,\gamma(i,\tau_2)} = w_{i,\gamma(i,\frac{1}{2}C^*_{\max})} \geq w_{i,\gamma(i,\frac{1}{2}C^*_{\max})-1} > \frac{1}{2}C^*_{\max}(\gamma(i,\frac{1}{2}C^*_{\max}) - 1)$, since $\gamma(i,\frac{1}{2}C^*_{\max}) \geq 2$ by (d) in Lemma 11. Then we have

$$W_2 > \frac{1}{2}C^*_{\max}\left(\sum_{T_i \in S_2}\gamma\left(i,\frac{1}{2}C^*_{\max}\right) - k\right) \geq \frac{1}{2}C^*_{\max}(m_1 + 1 - k). \tag{7}$$

□

Lemma 14. *In schedule σ_3, tasks in S_2 use at most m_1 processors.*

Proof. By contradiction. Assume that tasks in S_2 use more than m_1 processors.

If $q = 0$, we have $W_1 + W_2 > \frac{3}{4}(m_1 - q)C^*_{\max} + \frac{1}{4}(m_1 + 1)C^*_{\max} = m_1C^*_{\max}$ according to Lemma 12, which contradicts $W_1 + W_2 \leq m_1C^*_{\max} - W_S$ by Corollary 2.

If $q \geq 1$. We have

$$W_2 < \frac{1}{4}m_1C^*_{\max} + \frac{3}{4}qC^*_{\max}, \tag{8}$$

since $W_1 + W_2 \leq m_1C^*_{\max} - W_S$ and $W_1 > \frac{3}{4}(m_1 - q)C^*_{\max}$ according to Lemma 12. Combining (6) and (8), we obtain

$$6qk < m_1 + 3q \Leftrightarrow 3q(2k-1) < m_1.$$

Combining (7) and (8), we obtain

$$2(m_1 + 1 - k) < m_1 + 3q \Leftrightarrow m_1 < 3q + (2k-2).$$

By transitivity we have the following inequality:

$$3q(2k-1) < 3q + (2k-2) \Leftrightarrow (3q-1)(k-1) < 0.$$

However, we have $k \geq 1$ and $q \geq 1$, which is a contradiction. □

From Lemma 6 and the goodness of schedule σ_0 established in Sections 5.2.1–5.2.3, we obtain the following theorem.

Theorem 2. *Procedure* $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$ *outputs in time* $O(nm + n\log n)$ *a feasible schedule that is* $(3/2)$*-approximate.*

Proof. We are left to analyze the time complexity. Initially, for any task T_i, $\gamma(i,\tau_1)$, $\gamma(i,\tau_2)$ and $\gamma(i,\tau_3)$ can be computed in time $O(\log m)$ by a binary search since tasks are monotonic. For n tasks, it requires $O(n\log m)$ time. Step 1 requires $O(n)$ steps. In Step 2, it is well known that knapsack problem is NP-hard and cannot be solved in a time bounded by a polynomial in n unless P = NP. However, it admits a pseudo-polynomial algorithm whose time complexity is bounded by a polynomial in n and capacity m (Martello & Toth, 1990). In fact, dynamic programming recursions solve the knapsack problem exactly in $O(nm)$. Hence, Step 2 requires $O(nm)$ steps.

In Step 3.1, we check at most n tasks. In Step 3.2, we need to sort at most n tasks in the nondecreasing order of $t_{i,1}$ for $T_i \in S_1$ and $\gamma(i,\tau_1) = 1$, and check first two of them at most $n/2$ times. The time complexity is $O(n\log n)$. In Step 3.3, the analysis of time complexity is similar to that of Step 3.2. In Step 3.4, we need to check at most n tasks in S_2 and reallocate it to S_0 or S_1. If the tasks is reallocated to S_1, we add the task into the list of tasks sorted on $t_{i,1}$ or $t_{i,2}$ according to $\gamma(i,\tau_1)$. This takes $O(\log n)$ time. Then, the time complexity is $O(n\log n)$. In Step 3.5, we need to go back to Steps 3.2, 3.3 and 3.4. But the maximum times we need to run Steps 3.2, 3.3 and 3.4 are already counted in those steps. Therefore, Step 3 requires $O(n\log n)$ steps.

In Step 4.1, the time complexity is $O(n\log n)$. In Step 4.2, there are at most n steps. In Step 4.4, we need to modify at most m

ending times of all tasks and decide at most n starting times for small tasks. The greedy algorithm in Step 4 can be computed in $O(m + n\log n)$.

Therefore, for a given input $(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$, the algorithm requires $O(nm + n\log n)$ steps to deliver a schedule. □

Remarks. The performance ratio 3/2 in Theorem 2 is tight as shown in the following example. Let $n = 3, m = 4$. $t_{1,1} = t_{1,2} = t_{1,3} = t_{1,4} = 1$; $t_{2,1} = 1$, $t_{2,2} = \frac{1}{2}$, $t_{2,3} = t_{2,4} = \frac{1}{3}$; $t_{3,1} = 2$, $t_{3,2} = 1$, $t_{3,3} = t_{3,4} = \frac{2}{3}$. Clearly, the optimal makespan is $C^*_{\max} = 1$. According to our algorithm, we have $\tau_1 = 1$, $\tau_2 = \frac{1}{2}$ and $\tau_3 = \frac{3}{2}$. Therefore, all three tasks are large tasks. One optimal solution to $\mathrm{KP}(\mathcal{T}_L, m, \tau_1, \tau_2)$ may lead to the assignment of task T_1 with one processor and task T_3 with two processors in shelf S_1 and task T_2 with two processors in shelf S_2, which results in a makespan of 3/2.

6. Conclusions and remarks

To summarize, our FA Algorithm, which combines procedures $\mathbf{F}(\mathcal{T}, m)$ and $\mathbf{A}(\mathcal{T}, m, \tau_1, \tau_2, \tau_3)$, outputs schedule $\tilde{\sigma}$ of makespan at most $\frac{3}{2}\tau_0$ if $\tau_0 \le C^*_{\max}$. Otherwise, by correctly computing (τ_1, τ_2, τ_3), it outputs schedule σ of makespan $C(\tau_1, \tau_2, \tau_3) \le \frac{3}{2}C^*_{\max}$. Combining Theorems 1 and 2, we obtain the following main theorem.

Theorem 3. *The FA Algorithm for scheduling monotonic modable tasks is $\frac{3}{2}$-approximate. Its time complexity is $O(nm\log(nm))$ (if $m > n$) and $O(n^2\log n)$ (if $m \le n$).* □

For non-monotonic moldable tasks, it is known that no approximation algorithm has an approximation ratio smaller than $\frac{3}{2}$ unless P = NP. For monotonic moldable tasks, we believe it is also the case although a formal proof is still unavailable.

We remark that the novel general technique developed in the paper for removing the ϵ-term in the worst-case performance ratio can be applied to improving the performance guarantee of dual algorithms for other combinatorial optimization problems. In fact, the technique has already been applied successfully in another study (Wu, Jiang, Zhang, & Zhang, 2022).

Acknowledgments

This work is supported to the first two authors in part by the National Natural Science Foundation of China (Grant nos. 71971065 and 71531005), and to the third author in part by a Visiting Professorship of the School of Management, Fudan University.

References

Amoura, A. K., Bampis, E., Kenyon, C., & Manoussakis, Y. (2002). Scheduling independent multiprocessor tasks. *Algorithmica, 32*(2), 247–261.

Baker, B., Brown, D., & Katseff, H. (1981). A 5/4 algorithm for two-dimensional packing. *Journal of algorithms, 2*, 348–368.

Baker, B. S., Coffman, E. G., Jr., & Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on Computing, 9*(4), 846–855.

Bansal, N., Caprara, A., Jansen, K., Prädel, L., & Sviridenko, M. (2009). A structural lemma in 2-dimensional packing, and its implications on approximability. In *International symposium on algorithms and computation* (pp. 77–86). Springer.

Blazewicz, J., Cheng, T. E., Machowiak, M., & Oguz, C. (2011). Berth and quay crane allocation: A moldable task scheduling model. *Journal of the Operational Research Society, 62*(7), 1189–1197.

Bleuse, R., Hunold, S., Kedad-Sidhoum, S., Monna, F., Mounié, G., & Trystram, D. (2017). Scheduling independent moldable tasks on multi-cores with GPUs. *IEEE Transactions on Parallel and Distributed Systems, 28*(9), 2689–2702.

Brent, R. P. (1974). The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM), 21*(2), 201–206.

Chen, C., & Chu, C. (2013). A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. *IEEE Transactions on Parallel and Distributed Systems, 24*(8), 1479–1488. https://doi.org/10.1109/TPDS.2012.258.

Coffman, E. G., Jr., Garey, M. R., Johnson, D. S., & Tarjan, R. E. (1980). Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing, 9*(4), 808–826.

Delorme, X., Dolgui, A., Kovalev, S., & Kovalyov, M. Y. (2019). Minimizing the number of workers in a paced mixed-model assembly line. *European Journal of Operational Research, 272*(1), 188–194. https://doi.org/10.1016/j.ejor.2018.05.072.

Dolgui, A., Kovalev, S., Kovalyov, M. Y., Malyutin, S., & Soukhal, A. (2018). Optimal workforce assignment to operations of a paced assembly line. *European Journal of Operational Research, 264*(1), 200–211. https://doi.org/10.1016/j.ejor.2017.06.017.

Drozdowski, M. (2009). *Scheduling for parallel processing*: vol. 18. Springer.

Du, J., & Leung, J. Y.-T. (1989). Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics, 2*(4), 473–487.

Dutton, R. A., Mao, W., Chen, J., & Watson III, W. (2008). Parallel job scheduling with overhead: A benchmark study. In *2008 international conference on networking, architecture, and storage* (pp. 326–333). IEEE.

Feitelson, D. G., & Rudolph, L. (1996). Toward convergence in job schedulers for parallel supercomputers. In *Workshop on job scheduling strategies for parallel processing* (pp. 1–26). Springer.

Fotakis, D., Matuschke, J., & Papadigenopoulos, O. (2021). Assigning and scheduling generalized malleable jobs under subadditive or submodular processing speeds. 10.48550/arXiv.2111.06225.

Fujiwara, I., Tanaka, M., Taura, K., & Torisawa, K. (2018). Effectiveness of moldable and malleable scheduling in deep learning tasks. In *2018 ieee 24th international conference on parallel and distributed systems (ICPADS)* (pp. 389–398). https://doi.org/10.1109/PADSW.2018.8644536.

Gálvez, W., Grandoni, F., Ingala, S., & Khan, A. (2016). Improved pseudo-polynomial–time approximation for strip packing. In *36th IARCS annual conference on foundations of software technology and theoretical computer science (FSTTCS 2016): vol. 65* (pp. 9:1–9:14). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Leibniz International Proceedings in Informatics (LIPIcs)

Garey, M. R., & Graham, R. L. (1975). Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing, 4*(2), 187–200.

Ghosal, D., Serazzi, G., & Tripathi, S. K. (1991). The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering, 17*(5), 443.

Golan, I. (1981). Performance bounds for orthogonal oriented two-dimensional packing algorithms. *SIAM Journal on Computing, 10*(3), 571–582.

Harren, R., Jansen, K., Prädel, L., & van Stee, R. (2011). A (5/3 + ϵ)-approximation for strip packing. In *Algorithms and data structures* (pp. 475–487). Berlin, Heidelberg: Springer Berlin Heidelberg.

Harren, R., & Stee, R. (2009). Improved absolute approximation ratios for two-dimensional packing problems. In *12th international workshop on approximation algorithms for combinatorial optimization problems* (pp. 177–189).

Henning, S., Jansen, K., Rau, M., & Schmarje, L. (2020). Complexity and inapproximability results for parallel task scheduling and strip packing. *Theory of Computing Systems, 64*(1), 120–140.

Hochbaum, D. S., & Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM), 34*(1), 144–162.

Jansen, K. (2004). Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. *Algorithmica, PP*(39), 187–200.

Jansen, K. (2012). A (3/2+ϵ) approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *Proceedings of the twenty-fourth annual ACM symposium on parallelism in algorithms and architectures, SPAA 2012* (pp. 224–235). New York, NY, USA: Association for Computing Machinery.

Jansen, K., & Land, F. (2018). Scheduling monotone moldable jobs in linear time. In *2018 ieee international parallel and distributed processing symposium (IPDPS)* (pp. 172–181).

Jansen, K., & Porkolab, L. (2002). Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica, 32*(3), 507–520.

Jansen, K., & Rau, M. (2017). Improved approximation for two dimensional strip packing with polynomial bounded width. In *International workshop on algorithms and computation* (pp. 409–420). Springer.

Jansen, K., & Rau, M. (2019). Closing the gap for pseudo-polynomial strip packing. In *27th annual european symposium on algorithms (ESA 2019): vol. 144* (pp. 62:1–62:14). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Leibniz International Proceedings in Informatics (LIPIcs)

Jansen, K., & Solis-Oba, R. (2009). Rectangle packing with one-dimensional resource augmentation. *Discrete Optimization, 6*(3), 310–323.

Jansen, K., & Thöle, R. (2010). Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing, 39*(8), 3571–3615.

Johannes, B. (2006). Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling, 9*(5), 433–452.

Kenyon, C., & Rémila, E. (2000). A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research, 25*(4), 645–656.

Leuze, M. R., Dowdy, L. W., & Park, K. H. (1989). Multiprogramming a distributed-memory multiprocessor. *Concurrency: Practice and Experience, 1*(1), 19–33.

Ludwig, W., & Tiwari, P. (1994). Scheduling malleable and nonmalleable parallel tasks. In *Proceedings of the fifth annual ACM-SIAM symposium on discrete algorithms* (pp. 167–176). Society for Industrial and Applied Mathematics.

Marchal, L., Simon, B., Sinnen, O., & Vivien, F. (2018). Malleable task-graph scheduling with a practical speed-up model. *IEEE Transactions on Parallel and Distributed Systems, 29*(6), 1357–1370. https://doi.org/10.1109/TPDS.2018.2793886.

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. Wiley.

Mounié, G., Rapine, C., & Trystram, D. (1999). Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of the eleventh annual ACM symposium*

on parallel algorithms and architectures, SPAA 1999 (pp. 23–32). New York, NY, USA: Association for Computing Machinery.

Mounié, G., Rapine, C., & Trystram, D. (2007). A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM Journal on Computing, 37*(2), 401–412.

Nadiradze, G., & Wiese, A. (2016). On approximating strip packing with a better ratio than 3/2. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on discrete algorithms* (pp. 1491–1510). SIAM.

Schiermeyer, I. (1994). Reverse-fit: A 2-optimal algorithm for packing rectangles. In *European symposium on algorithms* (pp. 290–299).

Sleator, D. D. (1980). A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters, 10*(1), 37–40.

Steinberg, A. (1997). A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing, 26*(2), 401–409.

Sviridenko, M. (2012). A note on the Kenyon–Remila strip-packing algorithm. *Information Processing Letters, 112*(1–2), 10–12.

Turek, J., Wolf, J. L., & Yu, P. S. (1992). Approximate algorithms for scheduling parallelizable tasks. In *Proceedings of the fourth annual ACM symposium on parallel algorithms and architectures* (pp. 323–332).

Unsal, O. (2021). An extended formulation of moldable task scheduling problem and its application to quay crane assignments. *Expert Systems with Applications, 185*, 115617.

Wang, Q., & Cheng, K. H. (1991). List scheduling of parallel tasks. *Information Processing Letters, 37*(5), 291–297.

Wu, F., Jiang, Z., Zhang, R., & Zhang, X. (2022). Approximation algorithms for scheduling monotonic moldable tasks on multiple platforms. Submitted for journal publication.