Efficient Approximation Algorithms for Scheduling Moldable Tasks

Xiaohu Wu[a], Patrick Loiseau[b]

[a]*Beijing University of Posts and Telecommunications, Beijing, China*
[b]*Inria, FairPlay team, Palaiseau, France*

Abstract

Moldable tasks allow schedulers to determine the number of processors assigned to each task, thus enabling efficient use of large-scale parallel processing systems. We consider the problem of scheduling independent moldable tasks on processors and propose a new perspective of the existing speedup models: as the number p of processors assigned to a task increases, the speedup is linear if p is small and becomes sublinear after p exceeds a threshold. Based on this, we propose an efficient approximation algorithm to minimize the makespan. As a by-product, we also propose an approximation algorithm to maximize the sum of values of tasks completed by a deadline; this scheduling objective is considered for moldable tasks for the first time while similar works have been done for other types of parallel tasks.

Keywords: Scheduling, approximation algorithms, moldable tasks

1. Introduction

Most computations nowadays are done in a parallelized way on large computers containing many processors. Optimizing the use of processors leads to the problem of scheduling parallel tasks based on their characteristics. In certain cases, the number of processors assigned to a task is predefined by its owner and is said to be rigid. However, in many cases, the scheduler can decide this number before the task execution: if this number cannot be changed during the task execution, the task is said to be moldable; otherwise, it is said to be malleable[1]. Moldable tasks are

Email addresses: xiaohu.wu@bupt.edu.cn (Xiaohu Wu), patrick.loiseau@inria.fr (Patrick Loiseau)

[1]In the earlier literature, moldable tasks was also called malleable tasks. Now, malleable tasks refer to another type of parallel tasks (Drozdowski, 2004).

easier to implement and manage than malleable tasks; the latter require additional system support for task migrations and preemptions (Drozdowski, 2004).

1.1. General Problem Description

We consider the problem of scheduling n independent moldable tasks $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ on m identical processors; all tasks are available at time zero. For every task $T_j \in \mathcal{T}$, its execution time $t_{j,1}$ on one processor is given, as well as the speedup $\eta_{j,p}$ when assigned $p \geq 1$ processors, where p is a positive integer. The execution time of T_j on p processors is $t_{j,p} = \frac{t_{j,1}}{\eta_{j,p}}$; then, its workload is $D_{j,p} = p \times t_{j,p}$. The task T_j can be represented by a rectangle in the processors $\times$ time space. Like (Mounié et al., 1999, 2007; Jansen & Land, 2018), given a real number d, we define a parameter $\gamma(j, d)$ as the minimum number of processors needed to finish task T_j by time d; if T_j cannot be finished by time d on any permissible number of processors, we set by convention $\gamma(j, d) = +\infty$. We often hope to finish all tasks as soon as possible. Sometimes, a task T_j also has a value v_j that can be obtained if it is finished by a deadline τ; then we hope to finish by time τ the most valuable tasks. We will propose algorithms that generate schedules for different objectives: (i) minimize the makespan, *i.e.,* the maximum completion time of all tasks of $\mathcal{T}$ or (ii) choose a subset of tasks and finish them on the m processors by a deadline τ to maximize the throughput, *i.e.,* the aggregate value of tasks finished by time τ. For each task to be executed, a schedule will define *the number of processors assigned to it* and *the time interval in which it is finished*. An algorithm is a ρ-approximation if

- for our minimization problem, it produces a schedule whose makespan is at most ρ times the optimal makespan where $\rho \geq 1$;
- for our maximization problem, it produces a schedule whose throughput is at least ρ times the optimal throughput where $\rho \leq 1$.

It is always desired to have performance bound ρ closer to one, while keeping algorithms simple to run efficiently.

1.2. Typical Speedup Models, and Motivation

For moldable tasks, a key aspect that conditions scheduling is the relation between the task execution time $t_{j,p}$ and the number p of assigned processors. Now, we introduce three typical speedup models in literature and the most related works, as well as the main motivation of this paper. In this paper, our main problem is offline scheduling of independent moldable tasks for makespan minimization. While introducing the related works, if they have any difference with our main problem,

Table 1: The Most Relevant Algorithmic Results for the Linear-speedup Model

	Optimality or Approximation Ratio	Remarks
Wang & Cheng (1992)	$3 - \frac{2}{m}$	Dependent
Drozdowski (1996)	Exact	Malleable, Polynomial time solvable
Jain et al. (2012)	$\frac{m-k}{m}\frac{s-1}{s}$	Malleable, Throughput maximization
Lucier et al. (2013)	$2+\mathcal{O}\left(\frac{1}{(\sqrt[3]{s}-1)^2}\right)$	Malleable, Online, Throughput maximization
Wu & Loiseau (2015)	$\frac{s-1}{s}$ & exact respectively	Malleable, Throughput maximization
Guo & Shen (2017)	$\frac{m-k}{m}$ & exact respectively	Malleable, Throughput maximization
Benoit et al. (2022a)	2	for failure-prone platforms
Benoit et al. (2022b)	2.62	Dependent, Online

we only clarify their difference with ours; otherwise, they consider the same problem as our main problem.

Linear-Speedup Model. An ideal speedup model is linear when p does not exceed a threshold δ_j (Drozdowski, 2004): $t_{j,p} = \frac{t_{j,1}}{p}$ where $\eta_{j,p} = p$; the workload of T_j is independent of p since $D_{j,p} = pt_{j,p} = t_{j,1}$. Benoit et al. (2022a) propose a 2-approximation algorithm, called LPA-LIST, for failure-prone platforms with additional constraints in the process of executing jobs. We note that LPA-LIST is applicable to the main problem of this paper by setting the number of job execution failures in its model to zero. Like ours, the other related works of this paper are directly for failure-free platforms. When there are precedence constraints among moldable tasks, Wang & Cheng (1992) propose a $\left(3 - \frac{2}{m}\right)$-approximation algorithm while Benoit et al. (2022b) give a 2.62-approximation algorithm in the online setting. Besides, the case of scheduling independent malleable tasks has already been studied well, e.g., Drozdowski (1996) gives a polynomial time exact algorithm with a time complexity of $\mathcal{O}(n^2)$. Table 1 summarizes the most relevant works under this model and their differences with our main problem are clarified in the third column; here, the works whose objectives are throughput maximization will be introduced in Section 2.2.

Communication Time Model. The communication time model is defined by a

Table 2: Algorithmic Results for the Communication Time Model

	Approximation Ratio	Remarks
Dutton & Mao (2007)	$\frac{30}{13}$ when $m \to \infty$; 2, $\frac{9}{4}$ and $\frac{20}{9}$ for $m = 2$, 3 and 4 respectively	Online, $c_j = c$
Havill & Mao (2008)	$4 - \frac{4}{m}$ for even $m \geq 2$; $4 - \frac{4}{m+1}$ for odd $m \geq 3$	Online, $c_j = c$
Guo & Kang (2010)	$\frac{1+\sqrt{5}}{2}$ for $m = 2$	Online
Kell & Havill (2015)	1.5 for $m = 2$; 2 for $m = 3$	Online, $c_j = c$
Benoit et al. (2022a)	3	For failure-prone platforms
Benoit et al. (2022b)	3.61	Dependent, Online

function:

$$t_{j,p} = \frac{t_{j,1}}{p} + (p-1)c_j, \tag{1}$$

where c_j is a positive real number; the term $(p-1)c_j$ is used to model the communication overhead among different parts of a task. As more processors are assigned, the overhead and workload $D_{j,p}$ increase; if p is too large, $t_{j,p}$ will not decrease and even increase as p increases, due to the effect of $(p-1)c_j$. Like Table 1, Table 2 summarizes the related works. Specifically, when all tasks $T_j \in \mathcal{T}$ have the same $c_j = c$, Dutton & Mao (2007) give an online algorithm whose approximation ratio is 2, $\frac{9}{4}$, and $\frac{20}{9}$ for $m = 2$, 3, and 4 respectively, and is $\frac{30}{13}$ when $m \to \infty$. Havill & Mao (2008) propose an online algorithm with an approximation ratio $\frac{4(m-1)}{m}$ for even $m \geq 2$ and $\frac{4m}{m+1}$ for odd $m \geq 3$. Kell & Havill (2015) improve the work of (Dutton & Mao, 2007) by giving online algorithms whose approximation ratio are 1.5 and 2 for $m = 2$ and 3. The following works consider the case that each task T_j has a specific c_j. Guo & Kang (2010) give an online algorithm whose approximation ratio is $(1+\sqrt{5})/2$ for $m = 2$, and show that $(1+\sqrt{5})/2$ is a lower bound on the approximation ratio of any online algorithm for the problem with $m \geq 2$. In the offline setting, Benoit et al. (2022a) show that LPA-LIST is a 3-approximation for failure-prone platforms. Benoit et al. (2022b) consider online scheduling of moldable tasks with precedence constraints and give a 3.61-approximation algorithm.

Monotonic Model. To date, the best algorithm for our problem is designed by simply using a general monotonic assumption: $t_{j,p}$ is non-increasing and $D_{j,p}$ is non-decreasing in $p \in [1, m]$, where $\eta_{j,p} \leq p$. Fig. 1 illustrates the major algorithm improvements over the past three decades, where m is independent of n. Specifically, Belkhale & Banerjee (1990) give a $\frac{2}{1+1/m}$-approximation algorithm. Mounié

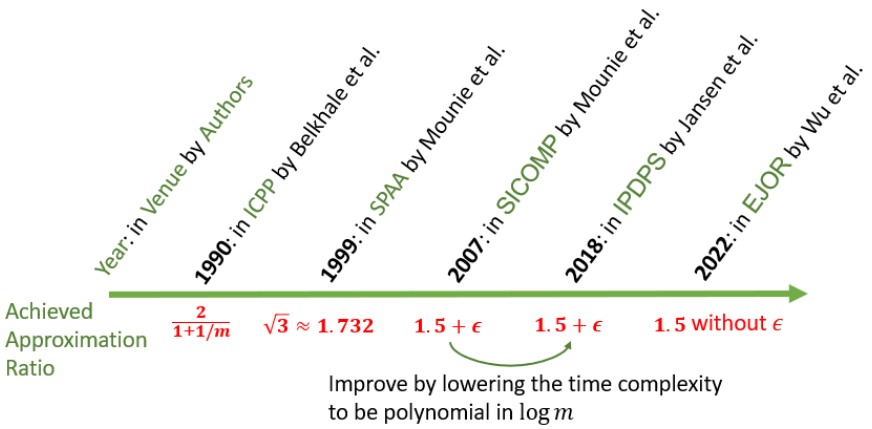

Figure 1: Major Algorithmic Improvements for the Monotonic Model over the Past Three Decades.

et al. (1999, 2007) first propose a $(\sqrt{3}+\epsilon)$-approximation algorithm and then a $(\frac{3}{2}+\epsilon)$-approximation algorithm with a complexity $\mathcal{O}(mn \log \frac{n}{\epsilon})$ where ϵ is arbitrarily small. Jansen & Land (2018) achieve an improved complexity polynomial in $\log m$ and $\frac{1}{\epsilon}$ and linear in n, although the algorithm is still a $(\frac{3}{2}+\epsilon)$-approximation. Additionally, in the special case where $m \geq 8\frac{n}{\epsilon}$, they give a FPTAS with a complexity $\mathcal{O}(n \log^2 m(\log m + \log \frac{1}{\epsilon}))$. The FPTAS requires a specific relation between n and m. Wu et al. (2023) give a $\frac{3}{2}$-approximation algorithm without ϵ and its time complexity is $\mathcal{O}(mn \log(mn))$ for $m > n$ and $\mathcal{O}(n^2 \log n)$ for $m \leq n$. As illustrated in Fig. 1, the three recent algorithmic results all have approximation ratios of around 1.5, and it is difficult to lower the best known approximation ratio 1.5. In this paper, we aim to sacrifice the generality of the monotonic model for a better performance guarantee.

In the case where n is independent of m, we hope to develop a ρ-approximation algorithm with $\rho < \frac{3}{2}$ and will revisit the related speedup models. Under the monotonic assumption, we have the following bounds of the execution time $t_{j,\gamma(j,d)}$ when a task T_j is assigned $\gamma(j,d)$ processors, which will also hold in this paper:

$$d \geq t_{j,\gamma(j,d)} > d(\gamma(j,d)-1)/\gamma(j,d). \tag{2}$$

By the definition of $\gamma(j,d)$, $D_{j,\gamma(j,d)}$ is the minimum workload needed to complete T_j by time d. Suppose that an algorithm produces a schedule of a makespan d. We observe that it is a $\frac{1}{\theta}$-approximation to makespan minimization if every task $T_j \in \mathcal{T}$ has the minimum workload and the aggregate workload processed on the m processors in $[0,d]$ is $\geq \theta md$ where θ is a lower bound of the processor utilization. Our objective is to make θ large (e.g., $\theta > \frac{2}{3}$). For each task T_j with large $\gamma(j,d)$ (e.g.,

$\gamma(j,d) \geq 4$), we have by Inequality (2) that executing it on $\gamma(j,d)$ processors alone can make these processors achieve a high utilization in $[0,d]$. One main challenge comes from tasks with smaller $\gamma(j,d)$. Then, a more precise speedup description than monotonicity could help, which is fortunately available in literature; it allows quantitatively characterizing the execution time reduction while keeping the workload constant, when the number p of processors assigned to a task T_j changes from $\gamma(j,d)$ to a larger value. We can thus obtain some desired properties and design a schedule under which the m processors achieve a high overall utilization in $[0,d]$ under some additional constraints (see Section 3).

The Proposed Speedup Model. While the linear-speedup model is studied, (Drozdowski, 1996) points out that it is typical of parallel applications that the speedup is linear when p is within a relatively small δ_j; assigning more than δ_j processors to execute T_j becomes less efficient. This model sets the parallelism bound of T_j to be δ_j, although it may be worth exploring the opportunity of assigning more processors to each task T_j to get better resource efficiency. Complementarily, the function (1) of the communication time model is also tested on widely used NAS parallel benchmarks and HPLinpack, which embody various computations with typical communication patterns for evaluating the performance of parallel systems (John & Eeckhout, 2018); here, an instance of a type of computation represents a task. The benchmarking results of Dutton et al. (2008) show that the function (1) can well approximate the execution times of tasks and also indicate that the factor c_j is far smaller than $t_{j,1}$: when p is small (up to a threshold δ_j), the effect of $(p-1)c_j$ on $t_{j,p}$ is negligible compared with the term $\frac{t_{j,1}}{p}$ and the speedup coincides accurately with the linear-speedup model (Drozdowski, 1996); assigning more than δ_j processors to execute T_j becomes less efficient: its execution time still decreases as p increases but its workload starts to increase, similarly to monotonic tasks; finally, there may be a larger threshold k_j such that when $p > k_j$, its execution time does not decrease any longer and even increases as p increases, since parallelizing on too many processors incurs an unacceptable overhead. Thus, we associate every task T_j with two thresholds δ_j and k_j to distinguish the speedup modes of T_j when p is in different ranges where $\delta_j \leq k_j$; then, we make the following definition on which we will base the algorithmic design of this paper.

Definition 1. *A task $T_j \in \mathcal{T}$ is (δ_j, k_j)-monotonic if it is moldable and satisfies*

1. *When $p \in [1, \delta_j]$, its workload remains constant and the speedup is linear,* i.e., *$D_{j,1} = D_{j,p} = p \times t_{j,p}$;*
2. *If $\delta_j < k_j$, its workload is increasing and its execution time is decreasing in $p \in [\delta_j, k_j]$, i.e., $D_{j,p} < D_{j,p+1}$ and $t_{j,p} > t_{j,p+1}$ for $p \in [\delta_j, k_j - 1]$.*
3. *The parameter k_j is a parallelism bound,* i.e., *the maximum number of processors allowed to be assigned to T_j.*

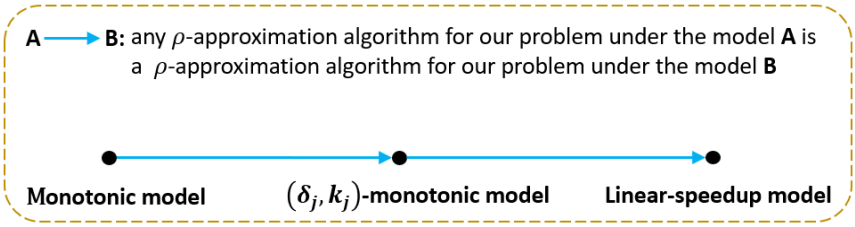

Figure 2: Relations Between Different Speedup Models

In Definition 1, the second point implies that assigning more than δ_j for executing T_j is less efficient but its execution time is decreasing in $p \in [\delta_j, k_j]$. The third point is used to reflect that when $p > k_j$, the workload begins to increase to an unacceptable extent such that the execution time does not decrease any more (i.e., $\eta_{j,k_j} \geq \eta_{j,p}$); then, assigning more than k_j processors to T_j cannot bring any benefit. Overall, when $p \in [1, k_j]$, $t_{j,p}$ is non-increasing in p while $D_{j,p}$ is non-decreasing in p.

Relations with the Monotonic and Linear-Speedup Models. For each task $T_j \in \mathcal{T}$, the speedup model defines the way that $t_{j,p}$ and $D_{j,p}$ change with the number p of allocated processors, where $t_{j,1}$ is known and $D_{j,p} = p \times t_{j,p}$. We consider the problem of offline scheduling of independent moldable tasks on identical machines, and the objective is either makespan minimization or throughput maximization. By Definition 1, a task whose speedup is linear is also a (δ_j, k_j)-monotonic task when $k_j = \delta_j$. Thus, the linear-speedup model is a special case of the (δ_j, k_j)-monotonic model; thus, for a given objective, any ρ-approximation algorithm for the problem under the (δ_j, k_j)-monotonic model of this paper is also a ρ-approximation algorithm for the problem under the linear-speedup model, as illustrated in Fig. 2. A problem A is S-reducible to a problem B if any instance of A can be transformed into an instance of B with the same optimal objective function value, and any solution for B can be transformed into a solution for A with the same objective function value (Crescenzi et al., 2016). There exists a S-reduction from the problem under the (δ_j, k_j)-monotonic model to the problem under the monotonic model, which is proved in Appendix A, and thus any ρ-approximation algorithm for the monotonic model can be transformed into a ρ-approximation algorithm for the (δ_j, k_j)-monotonic model.

Algorithms for scheduling problems with a more general speedup model have more extensive applicability, as illustrated in Fig. 2. However, algorithms under specific models are still important since they may be designed more finely to have better approximation ratios. For example, for online scheduling of moldable task

graphs to minimize the makespan, (Benoit et al., 2022b) give a 2.62-approximation algorithm for the linear-speedup model and a 3.61-approximation algorithm for the communication time model; they also generalize these speedup models and give a 5.72-approximation algorithm under the generalized model.

1.3. Algorithmic Results

Consider a set $\mathcal{T}$ in which each task T_j is (δ_j, k_j)-monotonic. Given a task T_j, its parameters δ_j and k_j are fixed; as reported in (Dutton et al., 2008), δ_j and k_j typically range in $[25, 150]$ and $[250, 512]$, depending on the types of computation embodied in the tasks of $\mathcal{T}$. We denote by δ the minimum linear-speedup threshold of all tasks and by k the maximum parallelism bound of all tasks, i.e.,

$$\delta = \min_{T_j \in \mathcal{T}}\{\delta_j\} \text{ and } k = \max_{T_j \in \mathcal{T}}\{k_j\}. \tag{3}$$

The number m of processors is large since our problem arises in large-scale parallel systems such as supercomputers and cloud computing clusters (Jain et al., 2012; Aridor et al., 2005), *e.g.,* supercomputers can have $m = 2^{16}$ processors inside (Aridor et al., 2005). Like (Jain et al., 2012), we assume in this paper that m is much larger than the maximum parallelism bound of tasks, i.e., $m \gg k$.

Let $u = \left\lceil \sqrt[2]{\delta} \right\rceil - 1$, that is, u is the unique integer such that $\delta \in [u^2 + 1, (u + 1)^2]$. Let t_m denote the maximum execution time of tasks when they are executed on one processor, i.e., $t_m = \max_{T_j \in \mathcal{T}}\{t_{j,1}\}$. In this paper, for any $\delta \geq 5$, *the main algorithmic result* is a $\frac{1}{\theta(\delta)}(1 + \epsilon)$-approximation algorithm for makespan minimization with a complexity $\mathcal{O}(n \log m \log (nmt_m/\epsilon))$ where

$$\theta(\delta) = \frac{u+1}{u+2}\left(1 - \frac{k}{m}\right).$$

The algorithm achieves an approximation ratio $\theta(\delta)$ close to $\frac{u+2}{u+1}$ since $m \gg k$. Typically, the minimum linear-speedup threshold δ has an effective range of $[25, 150]$ (Dutton et al., 2008). In the worst case that $\delta = 25$, $\theta(\delta)$ is close to $\frac{6}{5}$; when $\delta = 150$, $\frac{u+2}{u+1} = \frac{14}{13} \approx 1.077$, which is close to 1. The larger the threshold δ, the better the proposed algorithm. Under mild assumptions, we realize our goal to sacrifice the generality of the monotonic model for a better approximation ratio.

For throughput maximization with a given deadline τ, we assume that every task $T_j \in \mathcal{T}$ can be finished by time τ, i.e., $\gamma(j, \tau) \in [1, k_j]$. As a by-product, *another algorithmic result of this paper* is a $\theta(\delta)$-approximation algorithm with a complexity $\mathcal{O}(n^2 \log m)$ to maximize the throughput with a deadline τ. To the best of our knowledge, we are the first to address this scheduling objective for moldable tasks, while this objective has been addressed for other types of parallel tasks in the literature of scheduling theory (Jansen & Zhang, 2007; Fishkin et al., 2005).

The rest of this paper is organized as follows. In Section 2, we give more related works. In Section 3, we give an overview of the ideas developed in this paper. The following two sections are used to elaborate these ideas. In particular, in Section 4, we propose a scheduling algorithm $Sched$ that produces a schedule with several features described in Section 3. In Section 5, we show the application of $Sched$ to the objectives of makespan minimization and throughput maximization with a deadline respectively. Finally, we conclude this paper in Section 6.

2. Related Work

2.1. Makespan Minimization

The problem of scheduling moldable tasks to minimize the makespan is strongly NP-hard when $m \geq 5$ (Drozdowski, 2004). There is a long history of study with continuous improvements to the approximation ratio or time complexity. Turek et al. (1992) consider moldable tasks without monotonicity and propose a two-phases approach: (i) determine the number of processors assigned to each task and (ii) solve the resulting strip packing problem; the latter has well been studied, *e.g.,* we can directly use the 2-approximation algorithm of Steinberg (Steinberg, 1997). Further, the authors show that any λ-approximation algorithm of a complexity $\mathcal{O}(f(m,n))$ for strip packing can be transformed into a λ-approximation algorithm of a complexity $\mathcal{O}(mnf(m,n))$ for our problem. In the special case of monotonic tasks, Ludwig & Tiwari (1994) improve the transformation complexity to $\mathcal{O}(n\log^2 m + f(m,n))$. Jansen & Porkolab (2002) formulate the original problem as a linear program. They propose a polynomial time approximation scheme (PTAS) when the number m of processors is constant; here, the complexity is exponential in m. Further, Jansen & Thöle (2010) propose a PTAS when m is polynomially bounded in the number n of tasks. In the case of an arbitrary number of processors, Jansen (2012) also propose a polynomial time $(\frac{3}{2}+\epsilon)$-approximation algorithm for any fixed ϵ. Barketau et al. (2014) give an optimal enumerative algorithm whose time complexity is $\mathcal{O}(n^3 2^{(2n+m-2)n})$. In the special case of n identical tasks, Decker et al. (2006) give a $\frac{5}{4}$-approximation algorithm.

As introduced in Section 1, of the great relevance to our work are (Mounié et al., 1999, 2007; Jansen & Land, 2018) that use similar techniques for monotonic tasks. For example, Mounié et al. (2007) apply the dual approximation technique (Hochbaum & Shmoys, 1987): it takes a real number d as an input, and either outputs a schedule of a makespan $\leq \frac{3}{2}d$ or answers correctly that d is a lower bound of the optimal makespan. To realize this, tasks are mainly classified into two subsets $\mathcal{T}_1$ and $\mathcal{T}_2$ whose tasks are respectively assigned $\gamma(j,d)$ and $\gamma(j,\frac{d}{2})$ processors; the classification aims at minimizing the total workload W of $\mathcal{T}_1$ and $\mathcal{T}_2$ while guaranteeing that the total number of processors assigned to $\mathcal{T}_1$ is $\leq m$,

which is formulated as a knapsack problem. If the optimal W exceeds the processing capacity of the m processors, there exists no schedule with a makespan $< d$. Otherwise, the total number of processors assigned to $\mathcal{T}_2$ may exceed m and a series of reductions to the numbers of processors assigned to the tasks of $\mathcal{T}_1$ and $\mathcal{T}_2$ is taken to get a feasible schedule: the tasks are assigned to different parts of processors respectively in the time intervals $[0, \frac{3}{2}d]$, $[0, d]$ and $[d, \frac{3}{2}d]$.

Finally, our problem has also been studied well when the speedup $\eta_{j,p}$ is a concave or convex function of p (Blazewicz et al., 2004, 2006; Barketau et al., 2014; Ebrahimi et al., 2018), which is less relevant to the speedup model of this paper. We don't introduce them in this paper any more.

2.2. Throughput Maximization

Several works have considered scheduling other types of parallel tasks to maximize the throughput. Jansen & Zhang (2007) and Fishkin et al. (2005) consider scheduling rigid tasks with a common deadline, *e.g.,* the former apply the theory of knapsack problem and linear programming to propose an $(\frac{1}{2}+\epsilon)$-approximation algorithm. Jain et al. (2012) consider malleable tasks with individual deadlines. Each task has a linear speedup within a parallelism bound, and there is a parameter s used to characterize the minimum delay-tolerance of all tasks: each T_j has to be finished in a time window $[a_j, d_j]$; it has the minimum execution time len_j when assigned δ_j processors; s is the minimum ratio of $d_j - a_j$ to len_j among all tasks. For offline scheduling, Jain et al. (2012) propose a greedy $\frac{m-k}{m}\frac{s-1}{s}$-approximation algorithm where k is the maximum parallelism bound of all tasks. Wu & Loiseau (2015) prove that the best approximation ratio that the type of greedy algorithms of (Jain et al., 2012) can achieve is $\frac{s-1}{s}$ and propose such an algorithm with a time complexity of $\mathcal{O}(n^2)$; they also show a sufficient and necessary condition under which a set of malleable tasks with deadlines can feasibly be scheduled on a fixed number of processors and propose an exact algorithm by dynamic programming that has a time complexity of $\mathcal{O}\left(\max\{n^2, n(mT)^T\}\right)$, where T is the maximum deadline of tasks. Guo & Shen (2017) give a $\frac{m-k}{m}$-approximation algorithm with a time complexity of $\mathcal{O}(n^2 + nT)$ and also an exact algorithm with a time complexity of $\mathcal{O}\left(n(mT)^T\right)$. For online scheduling, Lucier et al. (2013) propose a $2+\mathcal{O}\left(1/(\sqrt[3]{s}-1)^2\right)$-approximation algorithm. In cloud computing clusters, many applications are delay-tolerant where $s \gg 1$ and $m \gg k$. Thus, their algorithms achieve good approximation ratios in practical settings.

3. Overview of the Approaches

Central to our algorithm design is an algorithm $Sched$ that aims to schedule a set $\mathcal{T}$ of tasks on the m processors in a time interval $[0, d]$ and achieves a processor

utilization $\geq \theta(\delta)$ on the conditions that (i) each scheduled task T_j has a workload $D_{j,\gamma(j,d)}$, which is the minimum workload to finish T_j by time d, and (ii) there exists some task of $\mathcal{T}$ rejected to be scheduled due to the insufficiency of idle processors (see Section 4). We establish the connection of $Sched$ with our two problems in the following ways.

For makespan minimization, we need to schedule all tasks of $\mathcal{T}$, while $Sched$ can play a role only when a part of tasks are scheduled. We apply a binary search procedure to find two parameters U and L such that $Sched$ can schedule all tasks by time U but only a part of tasks by time L, with the relation $U \leq L(1+\epsilon)$ (see Section 5.1). Let d^* denote the optimal makespan. We can establish the relation between U and d^* via L and prove $U/d^* \leq \frac{1}{\theta(\delta)}(1+\epsilon)$, thus showing that the resulting algorithm is a $\frac{1}{\theta(\delta)}(1+\epsilon)$-approximation. Specifically, in the case that $d^* \in [L, U]$, we have $U/d^* \leq (1+\epsilon)/\theta(\delta)$ trivially. In the case that $d^* < L$, we have that the total workload of all tasks of $\mathcal{T}$ in an optimal schedule is $\leq md^*$ but $\geq$ the total workload processed when $Sched$ manages to schedule a part of tasks of $\mathcal{T}$ by time L. Thus, we have $md^* \geq m\theta(\delta)L \geq m\theta(\delta)U/(1+\epsilon)$ and $U \leq d^*(1+\epsilon)/\theta(\delta)$.

For throughput maximization, $v_j/D_{j,\gamma(j,d)}$ is the maximum possible value obtained from processing a unit of workload of T_j, called its value density. Let us accept the maximum number of tasks in the non-increasing order of their value densities until $Sched$ cannot produce a feasible schedule by time τ; then, the feature of $Sched$ leads to that the utilization $\theta(\delta)$ will be the approximation ratio of the resulting algorithm (see Section 5.2).

Finally, the design of $Sched$ relies on the properties of the speedup model in Definition 1 to classify the tasks of $\mathcal{T}$. The threshold δ in Equation (3) is a fixed parameter and we have the following property by Definition 1.

Property 3.1. *If a task T_j is (δ_j, k_j)-monotonic, we have that (i) the workload $D_{j,p}$ is non-decreasing and the execution time $t_{j,p}$ is non-increasing in the number p of assigned processors when $p \in [1, k_j]$ and (ii) the speedup is linear when $p \in [1, \delta]$, i.e., $t_{j,p} = \frac{t_{j,1}}{p}$.*

For a task $T_j \in \mathcal{T}$, its execution time on p processors is defined by $t_{j,1}$ and $\eta_{j,p}$. Given the time d, $\gamma(j,d) = \min\{p \in [1, k_j] \,|\, t_{j,p} \leq d\}$ is a fixed parameter and can be found by binary search (Jansen & Land, 2018). The classification of tasks for the scheduling process mainly uses three integer variables ν, H and δ' and is based on the values of $\gamma(j,d)$, $t_{j,\gamma(j,d)}$ and $t_{j,\delta'}$; it attempts to guarantee that the aggregate execution time is in $[rd, d]$ when some tasks in the same class are executed on a group of $\gamma(j,d)$ or δ' processors. Specifically, ν and H are for distinguishing tasks with different $\gamma(j,d)$: a task T_j is said to have a large,

medium, or small $\gamma(j,d)$ if $\gamma(j,d)$ is $\geq H$, in $[\nu, H-1]$, or $\leq \nu - 1$ respectively, where $\nu < H$. Let $r = \frac{H-1}{H}$ and we will use rd and $(1-r)d$ to distinguish tasks with different execution times. The first class of tasks, denoted by $\mathcal{A}'$, includes every task that has a large execution time $\geq rd$ when assigned a group of $\gamma(j,d)$ processors (see Equation (6)), e.g., every task with large $\gamma(j,d)$ has such a feature by Inequality (2).

For the remaining tasks with medium or small $\gamma(j,d)$, we will maintain several relations among ν, H, δ' and δ. For example, by letting $H - 1 \leq \delta' \leq \delta$, the speedup is linear and the workload keeps constant when the number p of assigned processors ranges in $[\gamma(j,d), \delta']$. These relations finally enable the following properties:

- For the tasks with small $\gamma(j,d)$ whose execution times are $< rd$ when assigned $\gamma(j,d)$ processors, they are denoted by $\mathcal{B}_{\nu-1}$ and their execution times will decrease remarkably (by a factor at least $\frac{\delta'}{\nu-1}$) to a small value $< (1-r)d$ when assigned δ' processors (see Equation (7) and Lemma 3). Executing as many such tasks as possible on a group of δ' processors in $[0,d]$ will lead to an aggregate execution time $\geq rd$.

- Let h be an integer in $[\nu, H-1]$. For the tasks with $\gamma(j,d) = h$ whose execution times are $\geq (1-r)d$ when assigned δ' processors and $< rd$ when assigned h processors, they are denoted by $\mathcal{A}_h$ and there exists a positive integer x_h such that the aggregate execution time is in $[rd, d]$ when x_h such tasks are executed one by one on a group of δ' processors (see Equation (11) and Proposition 5).

Finally, each group of $\gamma(j,d)$ or δ' assigned processors described above can achieve a utilization $\geq r$ in $[0,d]$. The overall utilization $\theta(\delta)$ of the m processors is close to r and can be derived when some task is rejected due to the insufficiency of processors, with at most $k-1$ processors idle. The task classification and maintained relations are formally described in Section 4.1, with other related issues solved. The scheduling algorithm $Sched$ is given in Section 4.2.

4. The Algorithm $Sched$

In this section, we consider the case that every task $T_j \in \mathcal{T}$ can be finished by time d, i.e., $\gamma(j,d) \in [1, k_j]$.

Lemma 2. *For every (δ_j, k_j)-monotonic task $T_j \in \mathcal{T}$, Inequality (2) holds.*

Proof. We have $t_{j,\gamma(j,d)} \leq d$ and $t_{j,\gamma(j,d)-1} > d$ by the definition of $\gamma(j,d)$. By Property 3.1, $D_{j,\gamma(j,d)} \geq D_{j,\gamma(j,d)-1}$. Further, we have $\gamma(j,d)t_{j,\gamma(j,d)} \geq (\gamma(j,d)-1)t_{j,\gamma(j,d)-1} > (\gamma(j,d)-1)d$. Hence, Inequality (2) holds. □

4.1. Task Classification

Following the high-level ideas in Section 3, we now begin to elaborate the task classification. For ease of reference, we first summarize the maintained relations between the fixed parameter δ, the integer variables H, ν, δ', x_ν, $\cdots$, x_{H-1}, and the number $r = \frac{H-1}{H}$ where the meanings of these variables and the number r will be clarified later:

$$1 \leq \nu \leq H-1 \leq \delta' \leq \delta \tag{4a}$$

$$\frac{r\nu}{\delta'} \geq 1-r \tag{4b}$$

$$\frac{r(\nu-1)}{\delta'} < 1-r \tag{4c}$$

and for all $h \in [\nu, H-1]$

$$r\frac{h}{\delta'}x_h \leq 1, \tag{5a}$$

$$\max\left\{1-r, \frac{h-1}{\delta'}\right\} x_h \geq r. \tag{5b}$$

As we classify tasks and prove their properties, we can gradually perceive the underlying reasons why these relations are established to get the desired properties. At the end of this subsection, we will give a feasible solution of H, ν, δ', x_ν, $\cdots$, x_{H-1} that satisfy the relations (4a)-(5b).

Fig. 3 summarizes how to classify a task $T_j \in \mathcal{T}$ according to its value of $\gamma(j,d)$ and its execution time on $\gamma(j,d)$ or δ' processors. Specifically, *the first class of tasks* contains all tasks whose execution times $t_{j,\gamma(j,d)}$ are $\geq rd$ when assigned $\gamma(j,d)$ processors and is defined as

$$\begin{aligned}\boldsymbol{\mathcal{A}'} = &\{T_j \in \mathcal{T} | \gamma(j,d) \geq H\} \\ &\cup \left\{T_j \in \mathcal{T} | \gamma(j,d) \in [1, H-1], t_{j,\gamma(j,d)} \geq rd\right\}\end{aligned} \tag{6}$$

$\mathcal{A}'$ also includes a part of tasks with smaller $\gamma(j,d)$ but they have $t_{j,\gamma(j,d)} \geq rd$. Except $\mathcal{A}'$, the remaining tasks have medium or small $\gamma(j,d)$ and each has an execution time $t_{j,\gamma(j,d)} < rd$. Among these tasks, let $\mathcal{B}_{\nu-1}$ denote all tasks with $\gamma(j,d) \leq \nu-1$, i.e.,

$$\mathcal{B}_{\nu-1} = \{T_j \in \mathcal{T} \,|\, \gamma(j,d) \leq \nu-1, t_{j,\gamma(j,d)} < rd\}; \tag{7}$$

let $\mathcal{B}_{H-1}$ denote all tasks that satisfy $\gamma(j,d) \in [\nu, H-1]$ and $t_{j,\delta'} < (1-r)d$, i.e.,

$$\mathcal{B}_{H-1} = \{T_j \in \mathcal{T} | \gamma(j,d) \in [\nu, H-1], t_{j,\delta'} < (1-r)d, t_{j,\gamma(j,d)} < rd\}. \tag{8}$$

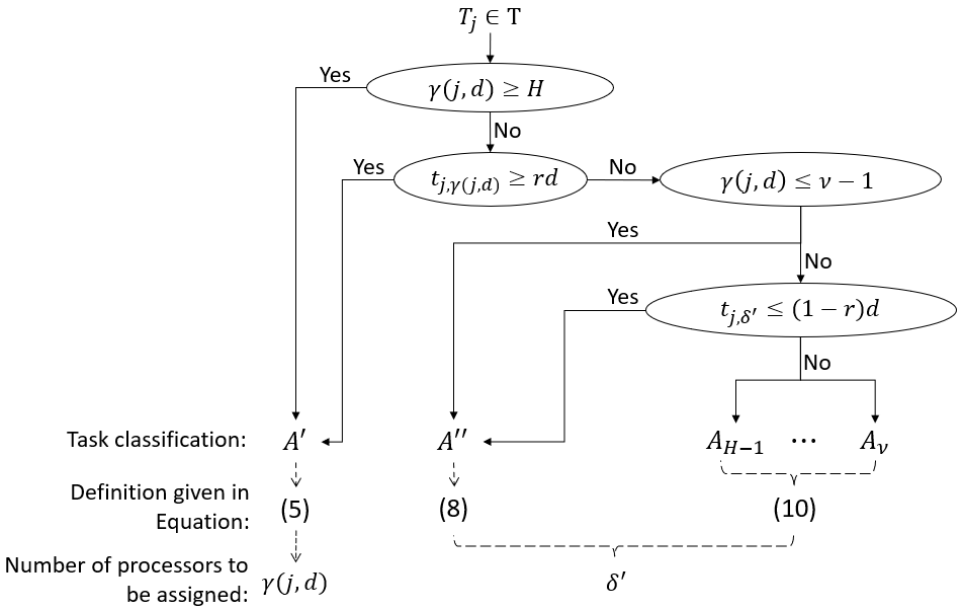

Figure 3: Task Classification.

The second class of tasks is defined as

$$\boldsymbol{\mathcal{A}''} = \mathcal{B}_{\nu-1} \cup \mathcal{B}_{H-1}. \tag{9}$$

For each task T_j with $\gamma(j,d) \leq H-1$, the relation (4a) ensures by Property 3.1 that the speedup is linear when the number of processors assigned to T_j changes from $\gamma(j,d)$ to δ'; when assigned δ' processors, its execution time $t_{j,\delta'}$ satisfies

$$t_{j,\delta'} = t_{j,\gamma(j,d)} \frac{\gamma(j,d)}{\delta'}. \tag{10}$$

Lemma 3. *For each task $T_j \in \mathcal{B}_{\nu-1}$, we have $t_{j,\delta'} < (1-r)d$.*

Proof. The execution time of T_j satisfies

$$t_{j,\delta'} \overset{(a)}{=} t_{j,\gamma(j,d)} \frac{\gamma(j,d)}{\delta'} \overset{(b)}{<} \frac{\nu-1}{\delta'} rd \overset{(c)}{<} (1-r)d$$

where the above (a), (b) and (c) are due to Equation (10), Equation (7) and the relation (4c) respectively. □

Proposition 4. *For every task $T_j \in \mathcal{A}''$, we have $t_{j,\delta'} < (1-r)d$.*

Proof. It follows from Lemma 3 and the definition of $\mathcal{B}_{H-1}$ in Equation (8). □

Finally, the remaining are all tasks with $\gamma(j,d) \in [\nu, H-1]$ and each has an execution time $t_{j,\gamma(j,d)} < rd$ when assigned $\gamma(j,d)$ processors and $t_{j,\delta'} \geq (1-r)d$ when assigned δ' processors. For each $h \in [\nu, H-1]$, *a single class of tasks* $\mathcal{A}_h$ is defined to contain all such tasks with $\gamma(j,d) = h$, *i.e.,*

$$\boldsymbol{\mathcal{A}_h} = \{T_j \in \mathcal{T} | \gamma(j,d) = h, t_{j,h} < rd, t_{j,\delta'} \geq (1-r)d\}. \tag{11}$$

Proposition 5. *When a task is assigned δ' processors, we have that*

(i) for every task $T_j \in \mathcal{A}_h$, its execution time $t_{j,\delta'}$ is $< l_h d$ where $l_h = \frac{h}{\delta'} r$;

(ii) the aggregate execution time of any x_h tasks of $\mathcal{A}_h$ is in $[rd, d]$.

Proof. The relation (4b) implies that ν is the maximum possible integer such that the relation (4c) can hold. Let us consider every task $T_j \in \mathcal{A}_h$ and by the definition of $\mathcal{A}_h$ in Equation (11), we have

$$t_{j,\gamma(j,d)} < rd \tag{12}$$

$$t_{j,\delta'} \geq (1-r)d. \tag{13}$$

where $\gamma(j,d) = h$. We have by Lemma 2 that the execution time of this task T_j satisfies

$$t_{j,\gamma(j,d)} > \frac{h-1}{h} d. \tag{14}$$

Thus, by Equation (10), we have

$$t_{j,\delta'} = t_{j,h} \frac{h}{\delta'} \overset{(d)}{<} \frac{h}{\delta'} rd \tag{15}$$

$$t_{j,\delta'} = t_{j,h} \frac{h}{\delta'} \overset{(e)}{>} \frac{(h-1)d}{h} \frac{h}{\delta'} = \frac{h-1}{\delta'} d \tag{16}$$

where the above (d) is due to Inequality (12), and (e) is due to Inequality (14). By Inequalities (13), (15) and (16), we have for any task $T_j \in \mathcal{A}_h$ that

$$t_{j,\delta'} \in \left[\max\left\{1-r, \frac{h-1}{\delta'}\right\} d,\ \frac{h}{\delta'} rd \right].$$

While executing any x_h tasks of $\mathcal{A}_h$ one by one on δ' processors, the relations (5a) and (5b) ensure that their aggregate execution time is in $[rd, d]$. Together with Inequality (15), Proposition 5 thus holds. □

Proposition 4 and 5 enable us to design good schedules. Executing as many tasks from $\mathcal{A}''$ as possible on δ' processors by time d can lead to that these processors have a utilization $\geq r$ in $[0, d]$. This also holds for the tasks of $\mathcal{A}_h$ where $h \in [\nu, H-1]$ since at least x_h tasks can be finished by time d.

Proposition 6. *For a given linear-speedup threshold $\delta \geq 5$, let $u = \left\lceil \sqrt[2]{\delta} \right\rceil - 1$ where $u \geq 2$ and $\delta \in [u^2+1, (u+1)^2]$. A feasible solution that satisfies the relations (4a)-(5b) is as follows:*

$$\begin{aligned} H &= u+2 \\ \delta' &= u^2+1 \\ \nu &= u \\ x_h &= 2u+1-h \quad \textit{for all } h \in \{\nu, H-1\} \end{aligned} \tag{17}$$

where $r = \frac{u+1}{u+2}$.

Proof. The proof is about verifying that the setting in Equation (17) can satisfy the relations (4a)-(5b) and its detail can be found in Appendix B. □

In the rest of this paper, we will set the parameter values in the way described in Proposition 6. Since $\nu = u$ and $H = u+2$, the tasks of $\mathcal{T}$ are finally classified as $\mathcal{A}', \mathcal{A}_{u+1}, \mathcal{A}_u, \mathcal{A}''$. Finally, we show the time complexity while classifying the tasks of $\mathcal{T}$. When a task T_j is allocated $p \in [1, m]$ machines, the speedup $\eta_{j,p}$ can be accessed via some oracle in constant time (Jansen & Land, 2018), e.g., the oracle can obtain such information by benchmarking studies (Dutton et al., 2008). Theoretically, the value of k_j or δ_j is a fixed integer in $[1, m]$ and can be obtained by binary search, leading to the proposition below.

Proposition 7. *For each task $T_j \in \mathcal{T}$, the time complexity of finding the value of k_j or δ_j is $\mathcal{O}(\log m)$.*

Proposition 8. *Given the value of d and the values of k_j and δ_j of each task $T_j \in \mathcal{T}$, the time complexity of task classification is $\mathcal{O}(n \log m)$.*

Proof. The time complexity of finding the value of $\delta = \min_{T_j \in \mathcal{T}}\{\delta_j\}$ in Equation (3) is $\mathcal{O}(n)$. We can directly compute the value of δ' by Equation (17). Afterwards, we classify each task T_j where we need to check the value of $\gamma(j,d)$, $t_{j,\gamma(j,d)}$, or $t_{j,\delta'}$ at most four times, as illustrated in Fig. 3; the time complexities of find these values determine the time complexity of classifying a task. Given the execution time $t_{j,1}$ on one processor, $\gamma(j,d) = \min\{p \in [1,k_j] \,|\, t_{j,p} \leq d\}$ can be found by binary search with a time complexity of $\mathcal{O}(\log k_j) \leq \mathcal{O}(\log m)$, where $k_j \leq m$. Given the values of $\gamma(j,d)$ and δ', $t_{j,\gamma(j,d)}$ and $t_{j,\delta'}$ can directly be computed in time $\mathcal{O}(1)$. Thus, the time complexity of classifying the n tasks is $\mathcal{O}(n \log m)$. □

4.2. Algorithm Description

Now, we give the scheduling algorithm $Sched$, which is presented in Algorithm 1. Let m' denote the number of idle processors; initially, $m' = m$. $\mathcal{T}$ is partitioned into $\mathcal{A}'$, $\mathcal{A}_{u+1}, \mathcal{A}_u$, $\mathcal{A}''$, and these sets are also sorted and assigned in this order where the tasks in the same set are chosen in an arbitrary order. Following this order, $Sched$ assigns tasks in the following way until all tasks of $\mathcal{T}$ are assigned or there are not enough idle processors:

Algorithm 1: $Sched(d)$

1 Set the parameters by Proposition 6 and classify the tasks of $\mathcal{T}$
2 $m' \leftarrow m$, $(\mathcal{X}', \mathcal{X}_{u+1}, \mathcal{X}_u, \mathcal{X}_{u-1}) \leftarrow (\mathcal{A}', \mathcal{A}_{u+1}, \mathcal{A}_u, \mathcal{A}'')$
  // $X', \mathcal{X}_{u+1}, \mathcal{X}_u, \mathcal{X}_{u-1}$: the currently unassigned tasks
3 **while** $\mathcal{X}' \neq \emptyset$ *and* $k \leq m'$ **do**
4 Get an arbitrary task T_j off $\mathcal{X}'$: $\mathcal{X}' \leftarrow \mathcal{X}' - \{T_j\}$
5 Assign T_j onto $\gamma(j, d)$ idle processors: $m' \leftarrow m' - \gamma(j, d)$
6 **if** $\mathcal{X}' \neq \emptyset$ and $m' < k$, **then** exit
7 **if** $\bigcup_{l'=u-1}^{u+1} \mathcal{X}_{l'} \neq \emptyset$, **then** let l be the maximum integer in $\{u-1, u, u+1\}$ with $\mathcal{X}_l \neq \emptyset$
8 **while** $\bigcup_{l'=u-1}^{u+1} \mathcal{X}_{l'} \neq \emptyset$ *and* $\delta' \leq m'$ **do**
9 Get δ' idle processors: $m' \leftarrow m' - \delta'$
10 $\mathcal{T}_{\delta'} \leftarrow \emptyset$, $t \leftarrow 0$ // $\mathcal{T}_{\delta'}$: the tasks currently chosen for the δ' processors; t: the aggregate execution time of $\mathcal{T}_{\delta'}$
11 **while** $\mathcal{X}_l \neq \emptyset$ **do**
12 Get an arbitrary task T_j from $\mathcal{X}_l$
13 **if** $t + t_{j,\delta'} \leq d$ **then**
14 $t \leftarrow t + t_{j,\delta'}$, $\mathcal{T}_{\delta'} \leftarrow \mathcal{T}_{\delta'} \cup \{T_j\}$, $\mathcal{X}_l \leftarrow \mathcal{X}_l - \{T_j\}$
15 **else**
16 break // got enough tasks and go to line 22
17 **if** $\mathcal{X}_l = \emptyset$ *and* $l > u - 1$ **then**
  // begin to assign the tasks of $\mathcal{X}_{l-1}$, $\cdots$, $\mathcal{X}_{u-1}$
18 **if** *there is an integer* $\hat{l} \in [u-1, l-1]$ *such that* $\mathcal{X}_{\hat{l}} \neq \emptyset$ **then**
19 Reset l to the maximum such $\hat{l}$ // go to line 11
20 **else**
21 $l \leftarrow u - 1$ // then, $\bigcup_{l'=u-1}^{u+1} \mathcal{X}_{l'}$ becomes empty
22 Assign the tasks of $\mathcal{T}_{\delta'}$ on the δ' idle processors

(i) For each unassigned task $T_j \in \mathcal{A}'$, assign it onto $\gamma(j, d)$ idle processors; then, $m' = m' - \gamma(j, d)$ (lines 3-5).

(ii) If $m' \geq \delta'$, divide the idle processors into $\lfloor \frac{m'}{\delta'} \rfloor$ groups, each with δ' processors. For each group, get unassigned tasks of $\mathcal{A}_{u+1} \cup \mathcal{A}_u \cup \mathcal{A}''$ such that their aggregate execution time on δ' processors is $\leq d$ (lines 10-21); assign these tasks onto the group of processors (line 22).

k and δ' are given in Equations (3) and (17). Algorithm 1 ends (i) if there are unassigned tasks but the idle processors are not enough ($m' < k$ in line 6 or $m' < \delta'$ in line 8), or (ii) if all tasks of $\mathcal{T}$ have been assigned.

4.2.1. Example

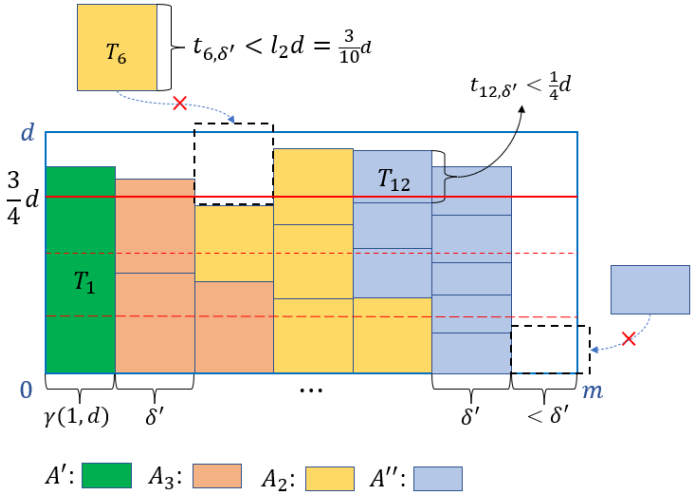

Figure 4: Task assignment when $\delta = 5$ where each colored rectangle represents a task of some type.

We give a toy example where $\delta = 5$ to illustrate the execution of Algorithm 1. By Proposition 6, we have $u = \nu = 2$, $H = 4$, $\delta' = 5$, $x_3 = 2$, $x_2 = 3$, and $r = \frac{3}{4}$; then, $\mathcal{T}$ is divided into 4 subsets $\mathcal{A}'$, $\mathcal{A}_3$, $\mathcal{A}_2$, and $\mathcal{A}''$ (line 1). Suppose that we are given $m = 33$, $\mathcal{A}' = \{T_1\}$, $\mathcal{A}_3 = \{T_2, T_3, T_4\}$, $\mathcal{A}_2 = \{T_5, T_6, \cdots, T_9\}$, $\mathcal{A}'' = \{T_{10}, T_{11}, \cdots, T_{18}\}$ and $\gamma(1, d) = H$ for T_1. Retrospectively, we get six groups from the m processors. The first group has $\gamma(1, d)$ processors, each of the remaining groups has δ' processors, and there are also 4 ungrouped processors. As illustrated in Fig. 4, Algorithm 1 assigns tasks in the following way:

(1) Assign the only task T_1 of $\mathcal{A}'$ onto the 1st group (lines 3-5).

(2) Assign $x_3 = 2$ tasks of $\mathcal{A}_3$ onto the 2nd group (lines 7-16, 22 where $l = 3$).

(3) Assign the last unassigned task of $\mathcal{A}_3$ onto the 3rd group (lines 8-14, where $l = 3$); then, $\mathcal{X}_3 = \emptyset$ and l becomes 2 (lines 17-19). Next, assign one task of $\mathcal{A}_2$ onto the 3rd group (lines 11-14, where $l = 2$). The second task of $\mathcal{A}_2$ cannot be added and completed by time d (lines 11-12, 15-16, 22).
(4) Assign $x_2 = 3$ tasks of $\mathcal{A}_2$ onto the 4th group (lines 8-16, 22 where $l = 2$).
(5) Similarly to the execution of Step 3, assign the last unassigned task of $\mathcal{A}_2$ and three tasks of $\mathcal{A}''$ onto the 5th group (lines 8-19, 22 where $l = 2, 1$).
(6) Assign five tasks of $\mathcal{A}''$ onto the 6th group (lines 8-16, 22 where $l = 1$).
(7) The algorithm ends when $m' = 4 < \delta'$ (line 8), although there is one unassigned task of $\mathcal{A}''$.

By the definition of $\mathcal{A}'$ in Equation (6) and Propositions 4 and 5, the 1st-2nd and 4th-6th groups have an execution time in $[rd, d]$. The 3rd group of δ' processors executes a mix of the tasks of $\mathcal{A}_3$ and $\mathcal{A}_2$; the aggregate execution time of tasks is $< rd$ but $\geq (1 - l_2)d$ since the rejected task of $\mathcal{A}_2$ has an execution time $\leq l_2 d = 3d/10$ by Proposition 5. Finally, there is one unassigned task of $\mathcal{A}''$ and the number of idle processors is at most $\delta' - 1$. The total number of processors whose execution time is $< rd$ is $\delta' + (\delta' - 1) = 2\delta' - 1$. The total workload processed by the m processors in $[0, d]$ is at least

$$w' = (m - 2\delta' + 1)rd + \delta'(1 - l_2)d,$$

and the overall processor utilization in $[0, d]$ is at least

$$\frac{w'}{md} = r - \frac{r(2\delta' - 1)}{m} + \frac{(1 - l_2)\delta'}{m} = \frac{3}{4} - \frac{3.25}{m}. \tag{18}$$

4.2.2. Algorithm Analysis

Now, we prove the features of $Sched$. The following conclusion is a generalization of Equation (18) in the example above.

Proposition 9. *If $Sched$ cannot schedule all tasks of $\mathcal{T}$ on the m processors by time d, then $Sched$ achieves a processor utilization of at least*

$$\theta(\delta) = r - \frac{rk}{m}$$

where $r = \frac{u+1}{u+2} \in (0, 1)$.

Proof. The proof is a generalization of the analysis process to derive Equation (18). Please see the detailed proof in Appendix C. □

Proposition 10. *Given the value of d and the values of k_j and δ_j of each task $T_j \in \mathcal{T}$, the time complexity of Algorithm 1 is $\mathcal{O}(n \log m)$.*

Proof. The time complexity of task classification is $\mathcal{O}(n \log m)$ by Proposition 8 (line 1). Afterwards, the n tasks are assigned to processors one by one (lines 4, 12) and $Sched$ stops when all tasks are assigned or there are not enough processors to assign the remaining tasks, which has a time complexity of $\mathcal{O}(n)$. Hence, Algorithm 1 has a time complexity of $\mathcal{O}(n \log m)$. □

Let $\mathcal{S}$ denote the tasks accepted and scheduled by Algorithm 1 where $\mathcal{S} \subseteq \mathcal{T}$. $\gamma(j, d)$ denotes the minimum number of processors needed to complete T_j by time d. As illustrated in Fig. 3, in Algorithm 1, the number of processors allocated to a task is either $\gamma(j, d)$ or δ' that is no larger than δ by Inequality (4a). By Definition 1 and Property 3.1, we have the following lemma.

Lemma 11. *In Algorithm 1, we have for every task $T_j \in \mathcal{S}$ that its workload is $D_{j,\gamma(j,d)}$, which is the minimum workload needed to be processed to complete T_j by time d.*

Proof. Please see the detailed proof in Appendix D. □

With Proposition 9 and Lemma 11, we have completed the design of the scheduling algorithm $Sched$ described in Section 3.

5. Application to Two Objectives

In this section, we apply $Sched$ to respectively minimize the makespan and maximize the throughput with a common deadline τ.

5.1. Makespan Minimization

Now, we give the algorithm for makespan minimization, which is formally presented in Algorithm 2 and also referred to as the OMS algorithm (Optimized MakeSpan). Its high-level idea is as follows. Initially, let U and L be such that $Sched$ can produce a feasible schedule of all tasks of $\mathcal{T}$ by time U but fails to do so by time L, *e.g.,* $U = n(\delta+2)\max_{T_j \in \mathcal{T}}\{t_{j,1}\}$ and $L = 0$ (line 1); we explain the reason why such U is feasible in Appendix E. The OMS algorithm will repeatedly operate as follows and stops when $U \leq (1+\epsilon)L$ (line 2):

1. $M \leftarrow \frac{U+L}{2}$ (line 3).
2. judge whether there exists a task $T_j \in \mathcal{T}$ that cannot be completed by time M with the parallelism bound k_j (lines 4-8).
3. if $\gamma(j, M) \in [1, k_j]$ for every task $T_j \in \mathcal{T}$ and $Sched$ can produce a feasible schedule of all tasks of $\mathcal{T}$ by time M, set $U \leftarrow M$ (lines 9-11); otherwise, set $L \leftarrow M$ (lines 9, 12-15).

Algorithm 2: The $OMS(\epsilon)$ algorithm

```
1  L ← 0, U ← n(δ + 2) max_{T_j ∈ T}{t_{j,1}}
2  while U > (1 + ε)L do
3      M ← (L+U)/2
4      Flag ← 1
5      for j ← 1 to n do
6          if γ(j, M) = +∞ then
7              Flag ← 0
8              break
9      if Flag = 1 then
           // every task T_j ∈ T can be completed by time M,
           i.e., γ(j, d) ∈ [1, k_j]
10         if Sched produces a feasible schedule of all tasks of T by time M
           then
11             U ← M
12         if Sched can only schedule a part of tasks of T by time M then
13             L ← M
14     else
15         L ← M
```

In the rest of this subsection, we analyze the approximation ratio and complexity of the algorithm. As shown below, for a task T_j, the larger the value of d, the smaller the value of $\gamma(j, d)$.

Lemma 12. *If $d' < d''$ and $\gamma(j, d'), \gamma(j, d'') \in [1, k_j]$, then we have $\gamma(j, d') \geq \gamma(j, d'')$.*

Proof. We prove this by contradiction. Suppose $\gamma(j, d') < \gamma(j, d'')$; then we have by Property 3.1 that $t_{j,\gamma(j,d')} \geq t_{j,\gamma(j,d'')}$. Since $t_{j,\gamma(j,d')} \leq d' < d''$, the minimum number of processors needed to complete T_j by time d'' is no greater than $\gamma(j, d')$, which contradicts the assumption that $\gamma(j, d') < \gamma(j, d'')$. □

Let d^* denote the optimal makespan. In an optimal schedule, let D_j^* denote the workload of a task T_j and D^* denote the total workload of all tasks of $\mathcal{T}$ to be processed on the m processors in $[0, d^*]$ where we have

$$md^* \geq D^*. \tag{19}$$

When the OMS algorithm ends, if $\gamma(j, L) \in [1, k_j]$ for every task $T_j \in \mathcal{T}$, only a part of tasks are scheduled by $Sched$ by time L and we have by Proposition 9 that $\theta(\delta)$ is a lower bound of the processor utilization in $[0, L]$; we denote by D_j^L the workload of a scheduled task T_j and by D^L the total workload of all the scheduled tasks; here, we have

$$mL \geq D^L \geq \theta(\delta)mL. \tag{20}$$

Lemma 13. *When the OMS algorithm ends, if $d^* < L$, then we have that (i) $D^* \geq D^L$ and (ii) $\gamma(j, L) \in [1, k_j]$ for every task $T_j \in \mathcal{T}$.*

Proof. For every $T_j \in \mathcal{T}$, if $d^* < L$, we have $\gamma(j, L) \in [1, k_j]$ since T_j can be finished by d^*, with the parallelism bound k_j. By Lemma 12, if $d' < d''$, we have $\gamma(j, d') \geq \gamma(j, d'')$. Since $d^* < L$, we have in an optimal schedule that the number of processors assigned to a task T_j is $\geq \gamma(j, d^*)$, which is $\geq \gamma(j, L)$. By Property 3.1, we have $D_j^* \geq D_{j,\gamma(j,d^*)} \geq D_{j,\gamma(j,L)}$. By Lemma 11, we have $D_{j,\gamma(j,L)} = D_j^L$. Finally, we have $D_j^* \geq D_j^L$ and $D^* \geq D^L$. □

Proposition 14. *The OMS algorithm gives a $\frac{1}{\theta(\delta)}(1+\epsilon)$-approximation to the makespan minimization problem with a complexity of $\mathcal{O}(n \log m \log(nmt_m/\epsilon))$ where $t_m = \max_{T_j \in \mathcal{T}}\{t_{j,1}\}$.*

Proof. For the approximation ratio, it suffices to show $U/d^* \leq \frac{1}{\theta(\delta)}(1+\epsilon)$ where $\theta(\delta) \in (0, 1)$. When the OMS algorithm ends, we have

$$U \leq (1+\epsilon)L. \tag{21}$$

Obviously, $d^* \leq U$. In the case that $d^* \in [L, U]$, we have $\frac{U}{d^*} \leq 1+\epsilon \leq \frac{1}{\theta(\delta)}(1+\epsilon)$. In the other case that $d^* < L$, we have by Inequalities (19), (20) and (21) and Lemma 13 that

$$md^* \geq D^* \geq D^L \geq m\theta(\delta)L \geq m\theta(\delta)\frac{U}{1+\epsilon}.$$

Further, we have $U/d^* \leq (1+\epsilon)/\theta(\delta)$.

Executing the OMS algorithm needs prior knowledge of the values of k_j and δ_j of all the n tasks of $\mathcal{T}$, which will be used for computing the upper bound U and in calling $Sched$ (lines 1, 10, 12); the time complexity of obtaining these values is $\mathcal{O}(n \log m)$ by Proposition 7. While executing the OMS algorithm, the initial values of U and L are $n(\delta+2)t_m$ and 0. The binary search stops when $U \leq L(1+\epsilon)$ and the number of iterations is $\mathcal{O}(\log(n\delta t_m/\epsilon)) \leq \mathcal{O}(\log(nmt_m/\epsilon))$, where $\delta \leq m$. At each iteration, the time complexity of computing $\gamma(j, d)$ is

Algorithm 3: GreedyAlgo(τ)

```
1  initialize S_i = {T_1, T_2, ··· , T_i} for all i ∈ [1, n];
2  for i ← 1 to n do
3      if Sched produces a feasible schedule of all tasks of S_i by time τ then
4          i' ← i;
5      else
6          exit;
```

$\mathcal{O}(\log k_j) \leq \mathcal{O}(\log m)$ as we show in the proof of Proposition 8; while judging whether there exists a task $T_j \in \mathcal{T}$ that cannot be completed by time M (lines 4-8), the time complexity is $\mathcal{O}(n \log m)$; then, $Sched$ is run (line 10 or 12) and has a time complexity $\mathcal{O}(n \log m)$ by Proposition 10. The entire execution process has a time complexity $\mathcal{O}(n \log m \log (nmt_m/\epsilon))$, which is also the complexity of the OMS algorithm. □

5.2. Throughput Maximization with a Common Deadline

Let $v'_j = v_j / D_{j,\gamma(j,\tau)}$, and it is the maximum possible value obtained from processing a unit of workload of T_j, referred to as the (maximum) value density of T_j. We assume without loss of generality that

$$v'_1 \geq v'_2 \geq \cdots \geq v'_n.$$

We propose a greedy algorithm called GreedyAlgo, presented in Algorithm 3: it considers tasks in the non-increasing order of their value densities v'_j and finally finds the maximum i' such that $Sched$ can output a feasible schedule by time τ for the first i' tasks, denoted by $\mathcal{S}_{i'}$, but fails to do so for the first $i' + 1$ tasks. The throughput of GreedyAlgo is $\sum_{j=1}^{i'} v_j$.

Proposition 15. *GreedyAlgo gives a $\theta(\delta)$-approximation to the throughput maximization problem with a common deadline and it has a complexity of $\mathcal{O}(n^2 \log m)$.*

In the rest of this subsection, we give an overview of the proof of Proposition 15. By Proposition 9, $\theta(\delta)$ is a lower bound of the processor utilization when $Sched$ schedules $\mathcal{S}_{i'}$ in $[0, \tau]$. Let $\mathcal{OPT}$ denote the optimal throughput of our problem. The proof of Proposition 15 has two parts:

(i) We give an upper bound of $\mathcal{OPT}$, denoted by $\overline{\mathcal{OPT}}$, *i.e.,*

$$\overline{\mathcal{OPT}} \geq \mathcal{OPT} \tag{22}$$

where $\overline{\mathcal{OPT}}$ will be specified in Equation (24).

(ii) We show that $\theta(\delta)$ is a lower bound of the ratio of the throughput of GreedyAlgo to the upper bound, *i.e.,*

$$\frac{\sum_{j=1}^{i'} v_j}{\overline{\mathcal{OPT}}} \geq \theta(\delta). \tag{23}$$

Then, we have by Inequalities (22) and (23) that

$$\frac{\sum_{j=1}^{i'} v_j}{\mathcal{OPT}} \geq \frac{\sum_{j=1}^{i'} v_j}{\overline{\mathcal{OPT}}} \geq \theta(\delta).$$

Thus, the throughput $\sum_{j=1}^{i'} v_j$ of GreedyAlgo is at least $\theta(\delta)$ times the optimal throughput $\mathcal{OPT}$ and GreedyAlgo is a $\theta(\delta)$-approximation algorithm.

For the first part, let us consider a fractional knapsack problem (Korte & Vygen, 2018) and there are a knapsack of size τm and n divisible items. With abuse of notation, each item is still denoted by T_j, with a fixed size $D_{j,\gamma(j,\tau)}$ and a value v_j. Its optimal solution is packing into the knapsack the first σ items, denoted by $\mathcal{S}'$, with the highest value densities such that their total size equals τm: $\sum_{j=1}^{\sigma-1} D_{j,\gamma(j,\tau)} + \alpha D_{\sigma,\gamma(\sigma,\tau)} = \tau m$ where $\alpha \in (0, 1]$ and the σ-th item may be partially packed. The following lemma completes the description of the first part.

Lemma 16. *An upper bound of $\mathcal{OPT}$ is*

$$\overline{\mathcal{OPT}} = \sum_{j=1}^{\sigma-1} v_j + \alpha v_\sigma, \tag{24}$$

which is the optimal value of the knapsack problem.

Proof. GreedyAlgo chooses a subset of tasks $\mathcal{S}_{i'} = \{T_1, T_2, \cdots, T_{i'}\}$ and uses $Sched$ to schedule $\mathcal{S}_{i'}$ on the m processors in $[0, \tau]$. We will show that any solution to the problem of this paper corresponds to a feasible solution to the above knapsack problem, where the same tasks/items are chosen and the two solutions have the same total value of tasks/items; the lemma thus holds. Specifically, when a task $T_j \in \mathcal{S}_{i'}$ is chosen in our problem and assigned p_j processors, we can correspondingly pack an item T_j with a size $D_{j,\gamma(j,\tau)}$ into the above knapsack. By Lemma 11, $D_{j,p_j} = D_{j,\gamma(j,\tau)}$ and $\sum_{T_j \in \mathcal{S}_{i'}} D_{j,\gamma(j,\tau)} \leq \tau m$; thus, the items T_1, T_2, $\cdots$, $T_{i'}$ can successfully be packed into the knapsack. □

For the second part, the detailed proof of (23) will be provided in Appendix F. Below, we provide the underlying intuition while proving (23). The workload of each task $T_j \in \mathcal{S}_{i'}$ accepted by GreedyAlgo is also $D_{j,\gamma(j,d)}$ by Lemma 11. $\mathcal{S}_{i'}$ and $\mathcal{S}'$ contain the first i' and σ tasks with the highest value densities respectively.

We have $i' \leq \sigma$ since in GreedyAlgo the utilization of the m processors in $[0, \tau]$ is ≤ 1. Thus, the average value density of $\mathcal{S}_{i'}$ is no smaller than the average value density of $\mathcal{S}'$, *i.e.,*

$$\frac{\sum_{j=1}^{i'} v_j}{\sum_{j=1}^{i'} D_{j,\gamma(j,d)}} \geq \frac{\overline{\mathcal{OPT}}}{\tau m}.$$

Further, we can prove (23):

$$\frac{\sum_{j=1}^{i'} v_j}{\overline{\mathcal{OPT}}} \geq \frac{\sum_{j=1}^{i'} D_{j,\gamma(j,d)}}{\tau m} \geq \theta(\delta).$$

Executing GreedyAlgo needs prior knowledge of the values of k_j and δ_j of all the n tasks of $\mathcal{T}$, which will be used in calling $Sched$ (line 3); the time complexity of obtaining these values is $\mathcal{O}(n \log m)$ by Proposition 7. During its execution, it considers $\mathcal{S}_1$, $\mathcal{S}_2$, $\cdots$, $\mathcal{S}_n$ one by one (line 2). Whenever $Sched$ attempts to schedule the tasks of $\mathcal{S}_i$ on m processor by time τ (line 3), it has a time complexity $\mathcal{O}(n \log m)$ by Proposition 10. Thus, the entire execution process has a time complexity $\mathcal{O}(n^2 \log m)$, which is also the complexity of GreedyAlgo.

6. Conclusions

In this paper, we study the problem of scheduling n independent moldable tasks on m processors that arises in large-scale parallel computations. For makespan minimization, the best known result is a $(\frac{3}{2} + \epsilon)$-approximation algorithm with a complexity linear in n and polynomial in $\log m$ and $\frac{1}{\epsilon}$, where ϵ is arbitrarily small; it is achieved under a monotonic assumption: the execution time of a task T_j is non-increasing and its workload is non-decreasing in the number p of assigned processors. We propose a new perspective of the existing speedup models: the speedup of a task T_j is linear when p is small (up to a threshold δ_j); afterwards, there may be a larger threshold k_j such that the task is strictly monotonic when p ranges in $[\delta_j, k_j]$; the bound k_j indicates an unacceptable overhead when parallelizing the task on too many processors. Let δ be the minimum linear-speedup threshold of all tasks and k be the maximum parallelism bound of all tasks. For any $\delta \geq 5$, let $u = \lceil \sqrt[2]{\delta} \rceil - 1$. A main algorithmic result of this paper is a $\frac{1}{\theta(\delta)}(1 + \epsilon)$-approximation algorithm for makespan minimization with a complexity $\mathcal{O}(n \log m \log (nmt_m/\epsilon))$ where $\theta(\delta) = \frac{u+1}{u+2}\left(1 - \frac{k}{m}\right)$ $(m \gg k)$; typically, δ can range in $[25, 150]$. As a by-product, we also propose a $\theta(\delta)$-approximation algorithm for throughput maximization with a common deadline with a complexity $\mathcal{O}(n^2 \log m)$.

Acknowledgements

The work of Xiaohu Wu has been partially supported by the National Key R&D Program of China (2022YFB2902900). The work of Patrick Loiseau has been partially supported by MIAI@Grenoble Alpes (ANR-19-P3IA-0003), by the French National Research Agency (ANR) through grant ANR-20-CE23-0007 and through the "Investissements d'avenir" program (ANR-15-IDEX-02); and by the Alexander von Humboldt Foundation.

References

Aridor, Y., Domany, T., Goldshmidt, O., Kliteynik, Y., Moreira, J., & Shmueli, E. (2005). Open job management architecture for the blue gene/l supercomputer. In *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 91–107). Springer.

Barketau, M., Kovalyov, M., Weglarz, J., & Machowiak, M. (2014). Scheduling arbitrary number of malleable tasks on multiprocessor systems. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, *62*, 255–261.

Belkhale, K. P., & Banerjee, P. (1990). An approximate algorithm for the partitionable independent task scheduling problem. In *Proceedings of the 1990 International Conference on Parallel Processing* (pp. 72–75). Pennsylvania State University Press.

Benoit, A., Le Fèvre, V., Perotin, L., Raghavan, P., Robert, Y., & Sun, H. (2022a). Resilient scheduling of moldable parallel jobs to cope with silent errors. *IEEE Transactions on Computers*, *71*, 1696–1710.

Benoit, A., Perotin, L., Robert, Y., & Sun, H. (2022b). Online scheduling of moldable task graphs under common speedup models. In *Proceedings of the 51st International Conference on Parallel Processing* (pp. 1–12). ACM.

Blazewicz, J., Kovalyov, M., Machowiak, M., Trystram, D., & Weglarz, J. (2006). Preemptable malleable task scheduling problem. *IEEE Transactions on Computers*, *55*, 486–490.

Blazewicz, J., Machowiak, M., Weglarz, J., Kovalyov, M. Y., & Trystram, D. (2004). Scheduling malleable tasks on parallel processors to minimize the makespan. *Annals of Operations Research*, *129*, 65–80.

Crescenzi, P. (1997). A short guide to approximation preserving reductions. In *Proceedings of the Twelfth Annual IEEE Conference Computational Complexity* (pp. 262–273). IEEE.

Crescenzi, P., Fraigniaud, P., Halldorsson, M., Harutyunyan, H. A., Pierucci, C., Pietracaprina, A., & Pucci, G. (2016). On the complexity of the shortest-path broadcast problem. *Discrete Applied Mathematics*, *199*, 101–109.

Decker, T., Lücking, T., & Monien, B. (2006). A $\frac{5}{4}$-approximation algorithm for scheduling identical malleable tasks. *Theoretical Computer Science*, *361*, 226–240.

Drozdowski, M. (1996). Real-time scheduling of linear speedup parallel tasks. *Information processing letters*, *57*, 35–40.

Drozdowski, M. (2004). Scheduling parallel tasks – algorithms and complexity. In *Handbook of scheduling: algorithms, models, and performance analysis*. CRC Press.

Dutton, R. A., & Mao, W. (2007). Online scheduling of malleable parallel jobs. In *Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems* (pp. 136–141). ACTA Press.

Dutton, R. A., Mao, W., Chen, J., & Watson III, W. (2008). Parallel job scheduling with overhead: A benchmark study. In *Proceedings of the IEEE International Conference on Networking, Architecture, and Storage* (pp. 326–333). IEEE.

Ebrahimi, R., McCauley, S., & Moseley, B. (2018). Scheduling parallel jobs online with convex and concave parallelizability. *Theory of Computing Systems*, *62*, 304–318.

Fishkin, A. V., Gerber, O., Jansen, K., & Solis-Oba, R. (2005). Packing weighted rectangles into a square. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science* (pp. 352–363). Springer.

Guo, L., & Shen, H. (2017). Efficient approximation algorithms for the bounded flexible scheduling problem in clouds. *IEEE Transactions on Parallel and Distributed Systems*, *28*, 3511–3520.

Guo, S., & Kang, L. (2010). Online scheduling of malleable parallel jobs with setup times on two identical machines. *European Journal of Operational Research*, *206*, 555–561.

Havill, J. T., & Mao, W. (2008). Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research*, *187*, 1126–1142.

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

Jain, N., Menache, I., Naor, J., & Yaniv, J. (2012). Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures* SPAA'12 (pp. 255–266). ACM.

Jansen, K. (2012). A $(\frac{3}{2}+\epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *Proceedings of the 24th annual ACM symposium on Parallelism in algorithms and architectures* (pp. 224–235). ACM.

Jansen, K., & Land, F. (2018). Scheduling monotone moldable jobs in linear time. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium* (pp. 172–181). IEEE.

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

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

Jansen, K., & Zhang, G. (2007). Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, *47*, 323–342.

John, L. K., & Eeckhout, L. (2018). *Performance evaluation and benchmarking*. CRC Press.

Kell, N., & Havill, J. (2015). Improved upper bounds for online malleable job scheduling. *Journal of Scheduling*, *18*, 393–410.

Korte, B., & Vygen, J. (2018). The knapsack problem. In *Combinatorial Optimization: Theory and Algorithms* (pp. 471–487). Berlin, Heidelberg: Springer.

Lucier, B., Menache, I., Naor, J. S., & Yaniv, J. (2013). Efficient online scheduling for deadline-sensitive jobs. In *Proceedings of the 25th ACM symposium on Parallelism in Algorithms and Architectures* (pp. 305–314). ACM.

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). ACM.

Mounié, G., Rapine, C., & Trystram, D. (1999). Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of the 11th ACM symposium on Parallel algorithms and architectures* (pp. 23–32). ACM.

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

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

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

Wang, Q., & Cheng, K.-H. (1992). A heuristic of scheduling parallel tasks and its analysis. *SIAM Journal on Computing*, *21*, 281–294.

Wu, F., Zhang, X., & Chen, B. (2023). An improved approximation algorithm for scheduling monotonic moldable tasks. *European Journal of Operational Research*, *306*, 567–578.

Wu, X., & Loiseau, P. (2015). Algorithms for scheduling deadline-sensitive malleable tasks. In *Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing* (pp. 530–537). IEEE.

Appendix A. S-reduction

For a given objective, the problem of offline scheduling of independent moldable tasks on identical machines under the (δ_j, k_j)-monotonic model is referred to as the problem A, while its counterpart under the monotonic model is referred to as the problem B. let $\mathcal{OPT}_A$ and $\mathcal{OPT}_B$ denote the optimal objective function values of the two problems A and B; here, the objective function can be either makespan minimization or throughput maximization. Let c_A and c_B denote the objective function values of the two problems A and B. A S-reduction from A to B is formally defined as follows (Crescenzi, 1997; Crescenzi et al., 2016):

Definition 17. *A pair of functions (f, g) is a S-reduction from A to B if all of the following conditions are met: (1) functions f and g are computable in polynomial time; (2) if x is an instance of problem A, then $f(x)$ is an instance of problem B, and $\mathcal{OPT}_B(f(x)) = \mathcal{OPT}_A(x)$; (3) if y is a solution to $f(x)$, then $g(x, y)$ is a solution to x, and $c_A(x, g(x, y)) = c_B(f(x), y)$.*

Proposition 18. *The problem A is S-reducible to the problem B, where f and g have the same time complexity $\mathcal{O}(n)$.*

Proof. Let x denote a specific set of n independent (δ_j, k_j)-monotonic moldable tasks $\{T_1, T_2, \cdots, T_n\}$ for the problem A. For each task $T_j \in x$, its execution time is non-increasing and its workload is non-increasing in the number p of processors allocated to it when $p \in [1, k_j]$. We also construct another task T'_j as follows: (i) it has the same speedup feature as T_j when $p \in [1, k_j]$, (ii) if T'_j is allocated more than k_j processors (i.e., $p > k_j$), its execution time and workload cease to change, i.e., $t'_{j,p} = t'_{j,k_j}$ and $D'_{j,p} = D'_{j,k_j}$ where $t'_{j,p}$ is the execution time of T'_j and $D'_{j,p}$ is the workload of T'_j when it is allocated p processors, and (iii) all other possible features of T'_j are the same as T_j, such as the execution time on one processor and the task value; here, allocating T'_j more than k_j processors does not bring any benefit although there is no parallelism constraint on T'_j. Each task T'_j in $f(x)$ is a monotonic task. Let $f(x) = \{T'_1, T'_2, \cdots, T'_n\}$, which is an instance of B. The time complexity of constructing $f(x)$ from x is $\mathcal{O}(n)$.

Suppose y is an optimal or approximate solution to $f(x)$; y defines a feasible schedule of $f(x)$ that determines the number p'_j of processors allocated to each task $T'_j \in f(x)$ and the time interval $[a'_j, e'_j]$ in which T'_j is executed. Each T_j in x uniquely corresponds to a task T'_j in $f(x)$, and vice versa. The following function g transforms the solution y into a feasible solution $g(x, y)$ to x: for each scheduled task T'_j in $f(x)$ for the problem B, allocate $\min\{k_j, p'_j\}$ processors to T_j and execute T_j in the same time interval $[a'_j, e'_j]$ when it comes to the problem A; if a task T'_j in $f(x)$ is not scheduled, the corresponding T_j in x is not scheduled either. Obviously, g can be computed with a time complexity $\mathcal{O}(n)$. In the solutions $g(x, y)$ and y, $T_j \in x$ and $T'_j \in f(x)$ have the same workload and are finished at the same time, if they are scheduled. Thus, the two solutions have the same objective function value, e.g., the same makespan or throughput. Thus, we have $c_A(x, g(x, y)) = c_B(f(x), y)$, and

$$\mathcal{OPT}_A(x) \geq \mathcal{OPT}_B(f(x)). \tag{A.1}$$

Conversely, if y' is an optimal solution to x for the problem A in which the number of processors allocated to each task $T_j \in x$ is p_j and the time interval in which T_j is executed is $[a_j, e_j]$. Then, this solution to x is also a solution to $f(x)$

for the problem B. Thus, the two solutions have the same objective function value. we thus have

$$\mathcal{OPT}_A(x) \leq \mathcal{OPT}_B(f(x)). \tag{A.2}$$

By (A.1) and (A.2), we have $\mathcal{OPT}_A(x) = \mathcal{OPT}_B(f(x))$. □

Appendix B. Proof of Proposition 6

We can easily verify that the setting in Equation (17) satisfies the relation (4a). We have

$$\frac{r\nu}{\delta'} = \frac{u+1}{u+2}\frac{u}{u^2+1} \overset{(a)}{\geq} \frac{1}{u+2} = 1 - r$$

where the above (a) is due to $u(u+1) \geq u^2+1$; thus, the relation (4b) is satisfied. We have

$$\frac{r(\nu-1)}{\delta'} = \frac{u+1}{u+2}\frac{u-1}{u^2+1} < \frac{1}{u+2} = 1 - r;$$

thus, the relation (4c) is satisfied.

We have $h \in [\nu, H-1] = [u, u+1]$ by Equation (17). In the following, we first prove that the relations (5a) and (5b) hold when $h = u$. We have

$$r\frac{u}{\delta'}x_u = \frac{u+1}{u+2}\frac{u}{u^2+1}(u+1) \overset{(b)}{\leq} 1 \tag{B.1}$$

where (b) is due to that $(u+1)^2u - (u+2)(u^2+1) = -2 < 0$. Thus, the relation (5a) holds when $h = u$. We have

$$\max\left\{1-r, \frac{u-1}{\delta'}\right\} = \max\left\{\frac{1}{u+2}, \frac{u-1}{u^2+1}\right\} \overset{(c)}{=} \begin{cases} \frac{1}{u+2} & \text{if } 2 \leq u \leq 3 \\ \frac{u-1}{u^2+1} & \text{if } u \geq 4 \end{cases} \tag{B.2}$$

where (c) is due to that $(u^2+1) - (u+2)(u-1) = 3 - u$. Further, if $2 \leq u \leq 3$, we have

$$\frac{1}{u+2}x_u = \frac{u+1}{u+2} \leq 1. \tag{B.3}$$

If $u \geq 4$, we can easily verify that

$$\frac{u-1}{u^2+1}x_u = \frac{(u-1)(u+1)}{u^2+1} \leq 1. \tag{B.4}$$

By Inequalities (B.2), (B.3) and (B.4), the relation (5b) holds when $h = u$.

Next, we prove that the relations (5a) and (5b) hold when $h = u + 1$.

$$r\frac{u+1}{\delta'}x_{u+1} = \frac{u+1}{u+2}\frac{u+1}{u^2+1}u \overset{(d)}{\leq} 1 \tag{B.5}$$

where (d) is again due to that $(u+1)^2u - (u+2)(u^2+1) = -2 < 0$. Thus, the relation (5a) holds when $h = u + 1$. We have

$$\max\left\{1-r,\ \frac{u}{\delta'}\right\} = \max\left\{\frac{1}{u+2},\ \frac{u}{u^2+1}\right\} \overset{(e)}{=} \frac{u}{u^2+1}$$

where (e) is due to that $(u+2)u - (u^2+1) = 2u - 1 > 0$. Further, we can easily verify that

$$\frac{u}{u^2+1}x_{u+1} = \frac{u^2}{u^2+1} \leq 1.$$

Thus, the relation (5b) holds when $h = u + 1$.

Appendix C. Proof of Proposition 9

After executing $Sched$, the m processors may be divided into three parts:

(i) the first part executes the tasks of $\mathcal{A}'$ (lines 3-5), *e.g,* the 1st group of processors in the example above,

(ii) the second part executes the tasks of $\mathcal{A}_{H-1}, \cdots, \mathcal{A}_u, \mathcal{A}''$ (lines 7-22), *e.g.,* the 2nd-6th groups in the example,

(iii) the third part is idle and not assigned any task.

$Sched$ ends with *two cases*: (i) $m' < k$ (line 6), or (ii) $m' < \delta'$ (line 8). Different parts exist in each case. Our analysis proceeds by showing (a) which parts of processors exist in each case and (b) the utilization of each part. $\theta(\delta)$ is a lower bound of the ratio of the total workload processed by different parts to md.

First, we analyze the utilizations of the three parts. The first part of processors has a utilization $\geq r$ in $[0, d]$ by the definition of $\mathcal{A}'$. The utilization of the third part is zero. The second part can be divided into several groups, each with δ' processors. Let $\mathcal{A}_{u-1} = \mathcal{A}''$ for ease of exposition. For each group, we have

(1) it is assigned the tasks purely from a single set $\mathcal{A}_h$ where $h \in [u-1, u+1]$ (see the second, fourth and sixth groups in the example), or

(2) it is a mix of the tasks of multiple sets $\mathcal{A}_h, \mathcal{A}_{h-1}, \cdots, \mathcal{A}_{h'}$ where $u + 1 \geq h > h' \geq u - 1$ and $h' \in \{u, u-1\}$.

In the former case, each group has an execution time $\geq rd$ by Propositions 4 and 5. In the latter, there exists a task T_j of $\mathcal{A}_{h'}$ that cannot be completed by time d:

(2.a) if $h' = u$, the group may have an execution time $< rd$ but $\geq (1 - l_u)d$ since $t_{j,\delta'} < l_u d$ by Proposition 5 (see the third group in the example); by Proposition 6, the processed workload is at least

$$w = \delta'(1 - l_u)d = (u^2 + 1)\left(1 - \frac{u}{u^2+1}\frac{u+1}{u+2}\right)d = \frac{u^3 + u^2 + 2}{u+2}d. \tag{C.1}$$

(2.b) if $h' = u - 1$, the group has an execution time $\geq rd$ since $t_{j,\delta'} < (1 - r)d$ (see the fifth group in the example).

To sum up, in the second part, there are at most δ' processors whose utilization is $< r$ in $[0, d]$ and on which the amount of processed workload is $\geq w$.

Next, we analyze which parts of processors exist. *In the first case*, $Sched$ ends at line 6 and the first and third parts may exist. The third part has at most $k - 1$ idle processors. Thus, the average utilization of the m processors is at least

$$r_1 = \frac{(m - k + 1)rd}{md} = r - r\frac{k-1}{m}.$$

In the second case, $Sched$ ends at line 8. All the three parts may exist and the third part has at most $\delta' - 1$ processors. For the second part, there are at most δ' processors whose utilization is $< r$. Thus, the average utilization of the m processors is at least

$$\begin{aligned} r_2 &= \frac{w + (m - \delta' - (\delta' - 1))rd}{md} \overset{(a)}{=} r - \frac{1}{m}\left(\left(2u^2 + 1\right)\frac{u+1}{u+2} - \frac{u^3 + u^2 + 2}{u+2}\right) \\ &= r - \frac{1}{m}\frac{u^3 + u^2 + u - 1}{u+2} \overset{(b)}{=} r - \frac{1}{m}\left(\delta' r - \frac{2}{u+2}\right) \end{aligned}$$

where the above (a) and (b) are due to Equation (C.1) and Proposition 6. Finally, when $Sched$ ends, a lower bound of the processor utilization is $\min\{r_1, r_2\}$, *i.e.,*

$$\theta(\delta) = r - \max\left\{\frac{r(k-1)}{m}, \frac{1}{m}\left(\delta' r - \frac{2}{u+2}\right)\right\} \overset{(c)}{\geq} r - \frac{rk}{m} \tag{C.2}$$

where (c) is because $\delta' \leq \delta \leq k$ by Inequality (4a).

Appendix D. Proof of Lemma 11

$\gamma(j,d)$ is the minimum number of processors needed to complete T_j by time d. By Property 3.1, $D_{j,\gamma(j,d)}$ is the minimum workload needed to be processed to complete T_j by time d. In Algorithm 1, the number of processors used to simultaneously execute a task is either $\gamma(j,d)$ for $\mathcal{A}'$ or no more than δ for $\mathcal{A}_{H-1}, \cdots, \mathcal{A}_u$, and $\mathcal{A}''$. For the latter, by Inequality (4a), we have for each task T_j that $\gamma(j,d) \leq H-1 \leq \delta' \leq \delta$; by Property 3.1, the workload of T_j keeps constant when the number of assigned processors varies in $[0,\delta]$ and we have in Algorithm 1 that the workload of T_j equals $D_{j,\gamma(j,d)}$. Thus, the lemma holds.

Appendix E. The Initial Value of U

The initial value of U is set as

$$U = n(\delta+2)\max_{T_j \in \mathcal{T}}\{t_{j,1}\},$$

which is at least $\delta+2$ times the total execution time of all tasks when every task is assigned one processor. $\gamma(j,U)$ is the minimum number of processors needed to complete T_j by time U, and we have $\gamma(j,U)=1$ for all $T_j \in \mathcal{T}$. We have by Inequality (4a) that $1 \leq H-1 \leq \delta' \leq \delta$. By Property 3.1, we have for every task $T_j \in \mathcal{T}$ that

$$t_{j,\delta'} = \frac{t_{j,1}}{\delta'} \leq \frac{U}{n(\delta+2)\delta'} \leq \frac{U}{\delta+2} < \frac{U}{H} = (1-r)U$$

where $n \geq 1$ and $r = \frac{H-1}{H}$. Every task of $\mathcal{T}$ has an execution time $< (1-r)U$ when assigned δ' processors. Thus, all tasks of $\mathcal{T}$ are in the class $\mathcal{A}''$, and the other classes $\mathcal{A}'$, $\mathcal{A}_{H-1}$, $\cdots$, $\mathcal{A}_\nu$ are empty. Now, we show that $Sched$ can produce a feasible schedule for all tasks of $\mathcal{T}$ by time U. All tasks of $\mathcal{T}$ constitute $\mathcal{A}''$ and will be executed one by one on δ' processors (see lines 7-22 of Algorithm 1); the total execution time of $\mathcal{T}$ is $\leq n\max_{T_j \in \mathcal{T}}\{t_{j,1}\} \leq U$.

Appendix F. The Detailed Proof of the Second Part

Below, we formally prove Inequality (23). GreedyAlgo accepts the first i' tasks with the highest value densities v'_j, and the achieved throughput is $\sum_{j=1}^{i'} v_j$. $Sched$ is used to schedule the i' tasks, and each accepted task T_j has a workload $D_{j,\gamma(j,\tau)}$ by Lemma 11. We denote by $\omega \in [0,1]$ the actual utilization of the m processors in $[0,\tau]$ achieved by GreedyAlgo, *i.e.,*

$$\sum_{j=1}^{i'} D_{j,\gamma(j,\tau)} = \omega\tau m.$$

By Proposition 9, $\theta(\delta)$ is a lower bound of the processor utilization and we have

$$\omega \geq \theta(\delta) \in (0, 1). \tag{F.1}$$

Since $\omega \leq 1$, we have

$$i' \leq \sigma. \tag{F.2}$$

Lemma 19. *The throughput* $\sum_{j=1}^{i'} v_j$ *achieved by GreedyAlgo is at least* $\theta(\delta)\overline{\mathcal{OPT}}$ *where* $\overline{\mathcal{OPT}}$ *is given in Equation (24).*

Proof. By Inequality (F.2), we will analyze two cases that $i' = \sigma$ and $i' < \sigma$ respectively. *In the case that* $i' = \sigma$, we have

$$\sum_{j=1}^{i'} v_j \leq \overline{\mathcal{OPT}} = \sum_{j=1}^{\sigma-1} v_j + \alpha v_\sigma$$

by Lemma 16. Thus, we have $\alpha = 1$ and the lemma holds.

In the case that $i' < \sigma$, let

$$X_1 = \frac{1}{\omega} \sum_{j=1}^{i'} D_{j,\gamma(j,\tau)} - \sum_{j=1}^{i'} D_{j,\gamma(j,\tau)}$$

$$X_2 = \begin{cases} \sum_{j=i'+1}^{\sigma-1} D_{j,\gamma(j,\tau)} + \alpha D_{\sigma,\gamma(j,\tau)} & \text{if } i' < \sigma - 1 \\ \alpha D_{\sigma,\gamma(j,\tau)} & \text{if } i' = \sigma - 1 \end{cases}$$

$$Y = \begin{cases} \sum_{j=i'+1}^{\sigma-1} v_j + \alpha v_\sigma & \text{if } i' < \sigma - 1 \\ \alpha v_\sigma & \text{if } i' = \sigma - 1 \end{cases}$$

Recall

$$\tau m = \frac{1}{\omega} \sum_{j=1}^{i'} D_{j,\gamma(j,\tau)} = \sum_{j=1}^{\sigma-1} D_{j,\gamma(j,\tau)} + \alpha D_{\sigma,\gamma(\sigma,\tau)}.$$

We thus have $X_1 = X_2$ since

$$\tau m - X_1 = \sum_{j=1}^{i'} D_{j,\gamma(j,\tau)} = \tau m - X_2.$$

The total value obtained by GreedyAlgo is $\sum_{j=1}^{i'} v_j$ and we have

$$\begin{aligned}\frac{\sum_{j=1}^{i'} v_j}{\omega\tau m} &\overset{(a)}{=} \frac{\sum_{j=1}^{i'} v'_j \left(\frac{1}{\omega} D_{j,\gamma(j,\tau)} - D_{j,\gamma(j,\tau)} + D_{j,\gamma(j,\tau)}\right)}{\tau m} \\ &\overset{(b)}{\geq} \frac{\sum_{j=1}^{i'} v_j + v'_{i'} X_1}{\tau m} \overset{(c)}{=} \frac{\sum_{j=1}^{i'} v_j + v'_{i'} X_2}{\tau m} \\ &\overset{(d)}{\geq} \frac{\sum_{j=1}^{i'} v_j + Y}{\tau m} = \frac{\overline{\mathcal{OPT}}}{\tau m}.\end{aligned} \tag{F.3}$$

Here, in Equation (a), $v_j = v'_j D_{j,\gamma(j,\tau)}$; Inequalities (b) and (d) are due to that $v'_1 \geq \cdots \geq v'_{i'} \geq \cdots \geq v'_n$; Equation (c) is due to that $X_1 = X_2$. Due to Inequality (F.3), we have $\sum_{j=1}^{i'} v_j \geq \omega\overline{\mathcal{OPT}}$; further, by Inequality (F.1), the lemma holds. □