SN Operations Research Forum manuscript No.
(will be inserted by the editor)

Algorithms for Scheduling Deadline-Sensitive Malleable Tasks*

Xiaohu Wu · Patrick Loiseau

Received: date / Accepted: date

Abstract Due to the ubiquity of batch data processing, the related problems of scheduling malleable batch tasks have received significant attention. We consider a fundamental model where a set of tasks is to be processed on multiple identical machines and each task is specified by a value, a workload, a deadline and a parallelism bound. Within the parallelism bound, the number of machines assigned to a task can vary over time without affecting its workload. In this paper, we identify a boundary condition and prove by construction that a set of malleable tasks with deadlines can be finished by their deadlines if and only if it satisfies the boundary condition. This core result plays a key role in the design and analysis of scheduling algorithms: (i) when several typical objectives are considered, such as social welfare maximization, machine minimization, and minimizing the maximum weighted completion time, and, (ii) when the algorithmic design techniques such as greedy and dynamic programming are applied to the social welfare maximization problem. As a result, we give four new or improved algorithms for the above problems.

1 Introduction

Batch data processing arises in many applications. We consider scheduling malleable/map-only tasks in the offline setting, which account for a significant proportion in batch data processing [6–8, 2–4]. Examples of such computations include parallel rendering in computer graphics, BLAST searches and CAP3 in bioinformatics, large scale facial recognition systems that compare thousands of

*This paper is an extended version of [21].

National Engineering Research Center of Mobile Network Technologies, Beijing University of Posts and Telecommunications, China
10 Xitucheng Road, Haidian District, Beijing, 100876, China
E-mail: xiaohu.wu@bupt.edu.cn

Inria, FairPlay team, Palaiseau, France
5 Avenue Henry Le Chatelier, 91120 Palaiseau, France
E-mail: patrick.loiseau@inria.fr

faces, grid and random search for hyperparameter optimization in machine learning, and data cleaning. A resource manager has C identical machines and a set $\mathcal{T}$ of n independent malleable tasks to be executed. Each task T_i has a parallelism bound k_i and a workload D_i, which is the number of required machine time units to be consumed. During the execution, the number of machines allocated to a task can vary over time but should not exceed k_i; meanwhile, its workload keeps constant. Throughout the paper, we assume $k_i \leq C$ without loss of generality.

In scheduling theory, malleable tasks can be viewed as an extension of the classic preemptive tasks whose parallelism bounds are one [12,13]. For the latter, when each task has to be finished by a unique deadline and there is only a single machine, the famous EDF (Earliest Deadline First) rule is optimal in the following sense: if there is any scheduling algorithm that can finish a set of tasks by their deadlines, the EDF rule can also achieve this. The EDF rule is initially designed to find an exact algorithm for scheduling batch tasks to minimize the maximum task lateness [14]. Since then, many applications of this rule have been found, e.g., (i) to design exact algorithms for preemptive tasks with release times [15], (ii) for scheduling tasks with deadlines to minimize the total weighted number of tardy tasks [16], and (iii) as a significant principle in the analysis of scheduling feasibility for real-time systems [17]. We are convinced that, as far as malleable tasks are concerned, an algorithm like EDF is also important for designing and analyzing scheduling algorithms (i) under various objectives, or (ii) when different algorithmic design techniques such as greedy and dynamic programming are applied.

The time horizon is divided into d time slots whose durations are the same and represent a unit of time. All tasks of $\mathcal{T}$ are available before the first slot. Each task T_i may be associated with a deadline d_i and a value that can be gained if the task is completed by the deadline. Before this work, Jain *et al.* proposed a greedy algorithm to maximize the social welfare, i.e., the sum of values of tasks completed by their deadlines [7]. Let

$$k = \max_{T_i \in \mathcal{T}}\{k_i\} \tag{1}$$

denote the maximum parallelism bound of all tasks of $\mathcal{T}$, and

$$d = \max_{T_i \in \mathcal{T}}\{d_i\} \tag{2}$$

denote the maximum deadline of tasks. Let

$$len_i = \left\lceil \frac{D_i}{k_i} \right\rceil \tag{3}$$

denote the *minimum execution time* of T_i, i.e., when T_i is always allocated the maximum number k_i of machines during the execution. The slackness s_i of T_i is defined as $\frac{d_i}{len_i}$. Let

$$s = \min_{T_i \in \mathcal{T}}\{s_i\} \tag{4}$$

be the slackness of the least flexible task ($s > 1$). Intuitively, s characterizes the machine allocation urgency, e.g., when s is close to one, there exists at least one task T_i such that the delay in allocating machines to T_i has to be very small to meet the parallelism and deadline constraints. Due to the batch processing nature, tasks are often delay-tolerant and s is large enough [7,9,10]. Jain *et al.* show that their algorithm achieves a performance guarantee $\frac{C-k}{C} \cdot \frac{s-1}{s}$. The authors consider the case of a large computing cluster with numerous machines such that C is much larger than k. Thus, the algorithm is claimed to be optimal since $\frac{C-k}{C} \cdot \frac{s-1}{s}$ approaches one. However, in other cases, a resource manager may have a limited number of machines and the parallelism bound of a task can be set to a large value; then the value of $\frac{C-k}{C}$ can be much smaller than one.

1.1 Our Results

Core result (Section 3). Let $\mathcal{S}$ be an arbitrary subset of $\mathcal{T}$, i.e., $\mathcal{S} \subseteq \mathcal{T}$. The core result of this paper is that a set $\mathcal{S}$ of malleable tasks (each characterized by a workload, a parallelism bound and a deadline) can be finished by their deadlines on the C machines if and only if $\mathcal{S}$ satisfies a boundary condition. While deriving this result, the first key is to identify the maximum workload of $\mathcal{S}$ that can be processed on C machines in an interval from any slot t to slot d, denoted as $\lambda_t^C(\mathcal{S})$, where $t \in [1, d]$. This leads to the definition of the boundary condition on the set $\mathcal{S}$ of tasks. The second key is to propose a scheduling algorithm $Sched(\mathcal{S})$ that can finish all tasks of $\mathcal{T}$ by their deadlines on the C machines if $\mathcal{S}$ satisfies the boundary condition. It has a time complexity of $\mathcal{O}(nkd \log(kn))$; here, the maximum deadline of tasks is assumed to be finitely bounded by a constant.

Applications (Sections 4 and 5). The core result can be used to design and analyze new or improved algorithms for scheduling malleable tasks under different objectives. The objectives considered in this paper include:

(a) *social welfare maximization*: maximize the sum of values of tasks completed by their deadlines;
(b) *machine minimization:* minimize the number of machines needed to produce a schedule that completes each task by its deadline;
(c) *maximum weighted completion time minimization:* minimize the maximum weighted completion time of tasks.

The first and third objectives have been considered in [6–8]. The second objective has been considered for other types of tasks [19] but we are the first to consider it for malleable tasks.

After applying the core result above, we obtain the following algorithmic results:

(i) an improved greedy algorithm GreedyRLM for social welfare maximization with a performance guarantee $\frac{s-1}{s}$ and a pseudo-polynomial time complexity of $\mathcal{O}(knd^2 \max\{d, n\})$, where k and d are defined in Eq. (1) and (2), respectively;
(ii) the first exact dynamic programming (DP) algorithm for social welfare maximization with a pseudo-polynomial time complexity of $\mathcal{O}(\max\{nC^d, n \log n\})$;
(iii) the first exact algorithm for machine minimization with a pseudo-polynomial time complexity of $\mathcal{O}(nd \log(kn))$;
(iv) a $(1+\epsilon)$-approximation algorithm with a pseudo-polynomial time complexity of $\mathcal{O}(nkd \log(kn) \log(n/\epsilon))$ for minimizing the maximum weighted completion time of all tasks.

Both the greedy algorithm of Jain *et al.* [7] and ours belong to a class of greedy algorithms that consider tasks in the non-decreasing order of their marginal values (i.e., the ratio of a task's value to its workload); a task will be accepted and allocated enough resource if it could be completed by its deadline with the currently remaining resource, and rejected otherwise. In this paper, we also show that

- for social welfare maximization, $(1+\epsilon)(\frac{s-1}{s}+\epsilon)$ is an upper bound of the performance guarantee that this class of greedy algorithms can achieve;
- our proposed greedy algorithm has a performance guarantee very close to this upper bound.

The second dynamic programming algorithm above allows obtaining the optimal social welfare but it works efficiently only when d is small since its time complexity is exponential in d.

Technical novelty. For the above class of greedy algorithms, we give a new algorithm analysis, and figure out what resource allocation features of tasks can benefit and determine the performance of such an algorithm. It is an extended analysis of the greedy algorithm for the standard knapsack problem [1] and does not rely on the dual-fitting technique, on which the algorithm in [7] is built; in the knapsack problem, each item/task has additional constraints in a two-dimensional space: the deadline and parallelism bound represent the maximum length and width of space that a task is allowed to occupy. The two most important algorithms for knapsack problem are based on the DP technique and the greedy approach, that also considers items by their marginal values [1]; we give in this paper their counterparts in the scenario of malleable tasks.

The second algorithm can be viewed as an extension of the pseudo-polynomial time exact algorithm in the single machine case [12], which is also designed via the generic dynamic programming procedure. However, before our work, how to enable this extension to malleable tasks was not clear as indicated in [6,7]. This is mainly due to the lack of a definition of $\lambda_t^C(\mathcal{S})$ and the lack of an algorithm that achieves the corresponding state. In contrast, an optimal machine utilization state in the single machine case can be defined much more easily and achieved by the EDF algorithm. Our core result enables a DP algorithm. The above third and fourth algorithms are obtained by respectively applying the core result to a binary search procedure, and the related results in [8].

1.2 Related works

The linear programming approaches to designing and analyzing algorithms for the task model of this paper [6,7] and its variants [9–11] have been well studied[1]. All these works consider the same objective of maximizing the social welfare. In [6], Jain *et al.* propose an algorithm with an approximation ratio of $(1+\frac{C}{C-k})(1+\epsilon)$ via *deterministic rounding of linear programming.* Subsequently, Jain *et al.* [7] propose a greedy algorithm GreedyRTL and use the *dual-fitting technique* to derive an approximation ratio $\frac{C-k}{C}\cdot\frac{s-1}{s}$. In [11], Bodík *et al.* consider an extension of our task model, i.e., DAG-structured malleable tasks; based on *randomized rounding*

[1] We refer readers to [13,20] for more details on the general techniques to design scheduling algorithms.

of linear programming, they propose an algorithm with an expected approximation ratio of $\alpha(\lambda)$ for every $\lambda > 0$, where $\alpha(\lambda) = \frac{1}{\lambda} \cdot e^{-\frac{1}{\lambda}} \cdot \left[1 - e^{-\frac{(1-1/\lambda)C-k}{2\omega\kappa} \cdot \ln\lambda \cdot (1-\frac{\kappa}{C})}\right]$. The online version of our task model is considered in [9,10]; again based on the *dual-fitting technique*, two weighted greedy algorithms are proposed respectively for non-committed and committed scheduling and achieve the competitive ratios of $cr_{\mathcal{A}} = 2 + \mathcal{O}(\frac{1}{(\sqrt[3]{s}-1)^2})$ where $s > 1$ [7] and $\frac{cr_{\mathcal{A}}(s\cdot\omega(1-\omega))}{\omega(1-\omega)}$ where $\omega \in (0,1)$ and $s > \frac{1}{\omega(1-\omega)}$.

Nagarajan *et al.* [8] also consider DAG-structured malleable tasks and propose two algorithms with approximation ratios of 6 and 2 respectively for the objectives of minimizing the total weighted completion time and the maximum weighted lateness of tasks. In particular, they show that seeking a schedule for DAG tasks can be transformed into seeking a schedule for tasks with simpler chain-precedence constraints. Then, whenever there exists a feasible schedule to complete a set of tasks by their deadlines, they propose a non-optimal algorithm where each task is completed by at most 2 times its deadline. Based on this deadline-based schedule, the authors further give two procedures to obtain near-optimal completion times of tasks in terms of the above two objectives.

Technically, the works [6,7,9–11] formulate their problem as an Integer Program (IP) and relax the IP to a relaxed linear program (LP). The techniques in [6, 11] require to solve the LP to obtain a fractional optimal solution and then manage to round the fractional solution to an integer solution of the IP that corresponds to an approximate solution to their original problem. In [7,9,10], the dual fitting technique first finds the dual of the LP and then construct a feasible algorithmic solution X to the dual in some greedy way. This solution corresponds to a feasible solution Y to their original problems, and, due to the weak duality, the value of the dual under the solution X (expressed in the form of the value under Y multiplied by a parameter $\alpha \geq 1$) will be an upper bound of the optimal value of the IP, i.e., the optimal value that can be achieved in the original problem. Therefore, the approximation ratio of the algorithm involved in the dual becomes clearly $1/\alpha$. Here, the approximation ratio is a lower bound of the ratio of the actual value obtained by the algorithm to the optimal value.

Parts of the results of this paper appear at the Allerton conference 2015 [21], including the core result, the greedy and exact algorithms for social welfare maximization, and the exact algorithm for machine minimization. After the initial publication of our results, a recent work [22] also show that the core result is central in the application of the LP technique to the problems here. Specifically, Guo and Shen [22] first use the LP technique to give a new proof of the boundary condition in the core result. Based on this condition, they give a new formulation of the original problems as IP programs, different from the ones in [6,7]. This new formulation enables from a different perspective to reveal almost the same two algorithmic results as ours, i.e., the two exact algorithms for social welfare maximization and machine minimization respectively with a time complexity $\mathcal{O}(n \cdot (C \cdot d)^d)$ and $\mathcal{O}((n+d)^{3.5} L_s(\log n + \log k))$, where L_s is the length of the LP's input.

Note that our proof approach is completely different from that of [22] based on linear programming. Furthermore, we show in this paper that an upper bound of the performance guarantee that the class of greedy algorithms of this paper can achieve is $(1+\epsilon)(\frac{s-1}{s} + \epsilon)$, and propose an algorithm whose performance guaran-

Table 1: Main Notation

Notation	Explanation
C	the total number of machines
$\mathcal{T}$	a set of tasks to be scheduled on C machines
T_i	a task in $\mathcal{T}$
D_i, d_i, v_i	the demand/workload, deadline, and value of a task T_i
k_i	the parallelism bound of T_i, i.e., the maximum number of machines that can be allocated to T_i at a slot
k, d, D	the maximum parallelism bound, deadline and demand of all tasks of $\mathcal{T}$
$y_i(t)$	the number of machines allocated to T_i at a slot t, subject to Eq. (5)
$W(t)$	the total number of machines that are allocated out to the tasks at t, i.e., $W(t) = \sum_{T_i \in \mathcal{T}} y_i(t)$
$\overline{W}(t)$	the total number of machines idle at t, i,e., $\overline{W}(t) = C - W(t)$
len_i	the minimum execution time of T_i where T_i is allocated k_i machines throughout the execution, i.e., $len_i = \lceil \frac{D_i}{k_i} \rceil$
s_i	the slackness of a task, i.e., $\frac{d_i}{len_i}$, measuring the urgency of machine allocation to complete T_i by the deadline
s	the minimum slackness of all tasks of $\mathcal{T}$, i.e., $\min_{T_i \in \mathcal{T}}\{s_i\}$
v_i'	the marginal value of T_i, i.e., $v_i' = \frac{v_i}{D_i}$
$[l]$	the set $\{0, 1, \cdots, l\}$
$[l]^+$	the set $\{1, 2, \cdots, l\}$
$\mathcal{S}$	a subset of $\mathcal{T}$
$\lambda_t(\mathcal{S})$	the maximum workload of $\mathcal{S}$ that could be executed in $[t, d]$ when $C = \infty$, $t \in [d]^+$
$\lambda_t^C(\mathcal{S})$	the maximum workload of $\mathcal{S}$ that could be executed in $[t, d]$ on the C machines when C is finite, $t \in [d]^+$
$\mu_t^C(\mathcal{S})$	the remaining workload of $\mathcal{S}$ that needs to be executed in $[1, t]$ after the maximum workload of $\mathcal{S}$ has be executed on the C machines in $[t+1, d]$, i.e., $\mu_t^C(\mathcal{S}) = \sum_{T_i \in \mathcal{S}} D_j - \lambda_{t+1}^C(\mathcal{S})$, $t \in [d]$
$\mathcal{A}_1, \mathcal{R}_1, \mathcal{A}_2, \cdots, \mathcal{R}_K$	the sets of consecutive accepted (i.e., fully allocated) and rejected tasks by Greedy where $\bigcup_{m=1}^{K} \mathcal{A}_m \cup \mathcal{R}_m = \mathcal{T}$
c_m	the maximum deadline of all rejected tasks of $\cup_{l=1}^{m} \mathcal{R}_l$
c_m'	the maximum deadline of $\cup_{l=1}^{m} \mathcal{A}_l$

tee is close to this upper bound. On the other hand, Guo and Shen [22] consider another standard to determine the order of considering tasks, and propose a greedy algorithm with a performance guarantee $\frac{C-k}{C}$ and a time complexity $\mathcal{O}(n^2 + nd)$. This algorithm may perform poorly when k is not negligible in comparison with C. Finally, we also propose the first algorithm based DP to exactly maximize the social welfare, and consider additionally a new objective of minimizing the maximum weighted completion time of tasks.

A parallel task can be executed on multiple machines simultaneously. Beyond malleable tasks, other types include rigid and moldable tasks, according to the way of setting the number of machines executing a task [5]. A task is said to rigid when the number of machines to execute it is fixed a priori [24–26]. A task is said to be moldable when the number is not fixed but determined before the execution; a rich set of algorithms have been proposed for scheduling them [27–35].

2 Model and Problem Description

There are C identical machines. The time horizon is divided into d time slots: $\{1, 2, \cdots, d\}$. Each slot represents a time interval whose length is a unit of time. We study the offline case where a set of n tasks $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ arrives before the first slot and is executed over the d slots. Each task $T_i \in \mathcal{T}$ is specified by several characteristics: (1) *value* v_i, (2) *demand* or *workload*[2] D_i (measured in machine time units), (3) *deadline* d_i, and (4) *parallelism bound* k_i, where $D_i, d_i, k_i \in \mathcal{Z}^+$. Each machine can process one unit of demand in one unit of time. The parallelism bound k_i limits that, at any time slot t, T_i can be executed on at most k_i machines simultaneously. k_i is a system parameter and the maximum parallelism bound k in Eq. (1) is thus assumed to be finite [18]. A task T_i can only utilize the machines at the slots in $[1, d_i]$, where $d = \max_{T_i\in\mathcal{T}}\{d_i\}$. For a task T_i to be executed, an *allocation* of machines to T_i is a function

$$\begin{aligned} &y_i(t) \in \{0, 1, \cdots, k_i\} \quad \text{for any } t \in [1, d_i] \\ &y_i(t) = 0 \quad \text{for any } t \in [d_i + 1, d], \text{ if } d_i < d \end{aligned} \tag{5}$$

where $y_i(t)$ is the number of machines allocated to T_i at a slot t. Eq. (5) represents the *parallelism and deadline constriants.* The allocation finishes task T_i if

$$\sum\nolimits_{t\leq d_i} y_i(t) = D_i. \tag{6}$$

For the system of C machines, we denote by $W(t) = \sum_{T_i\in\mathcal{T}} y_i(t)$ the system's workload at slot t and by $\overline{W}(t)$ the amount of available machines at t where:

$$W(t) = C - \overline{W}(t) \in [0, C]. \tag{7}$$

The relation of $W(t)$ and C in Eq. (7) is referred to as the *capacity constraint.* In addition, we assume that the maximum deadline of tasks is bounded. We denote by $[l]$ and $[l]^+$ the sets $\{0, 1, \cdots, l\}$ and $\{1, 2, \cdots, l\}$ for a positive integer l. Let $D = \max_{T_i\in\mathcal{T}}\{D_i\}$ denote the maximum workload of tasks. Other related parameters such as k, len_i and s have been given in Eq. (1)-(4). The main notation of this paper is summarized in Table 1.

Given the model above, the following three scheduling objectives are considered separately in this paper:

- *The first objective* is social welfare maximization. It aims to choose an a subset $\mathcal{S} \subseteq \mathcal{T}$ and produce a feasible schedule for $\mathcal{S}$ to maximize the social welfare $\sum_{T_i\in\mathcal{S}} v_i$; here, the value v_i of a task T_i is gained if and only if it is fully allocated by the deadline, i.e., $\sum_{t\leq d_i} y_i(t) \geq D_i$, and partial execution of a task yields no value.
- *The second objective* is machine minimization, i.e., seeking the minimum number of machines needed to produce a feasible schedule of $\mathcal{T}$ on C machines such that the task's parallelism bound and deadline constraints are not violated.
- *The third objective* is to minimize the maximum weighted completion time of tasks, i.e., $\max_{T_i\in\mathcal{T}}\{v_i \cdot c_i\}$, where c_i is the completion time of a task T_i.

[2] In this paper, we use the terms "demand" and "workload" interchangeably.

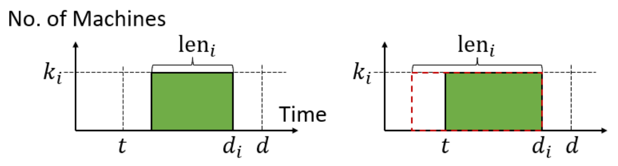


Fig. 1: The green area denotes the maximum workload of T_i that need or could be processed in $[t, d]$ when $t \leq d_i$: if $len_i \leq d_i - t + 1$, $\beta_i = D_i$; otherwise, $\beta_i = k_i \cdot (d_i - t + 1)$.

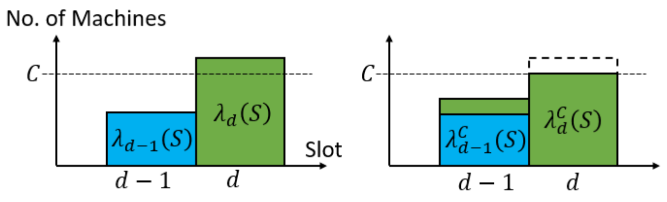


Fig. 2: Derivation from the definition $\lambda_t(\mathcal{S})$ to $\lambda_t^C(\mathcal{S})$ where $t = d, d-1$: in the left subfigure, the green area denotes $\lambda_d(\mathcal{S})$ and the green and blue areas together denote $\lambda_{d-1}(\mathcal{S})$.

3 Scheduling Feasibility

In this section, we give a boundary condition and prove its sufficiency and necessity for finishing a set of tasks by their deadlines on the C machines.

3.1 Feasibility Criterion Definition

All tasks form a set $\mathcal{T}$, and we denote by $\mathcal{S}$ an arbitrary subset of $\mathcal{T}$. In this subsection, we define the maximum workload of $\mathcal{S}$ that could be executed on the C machines in a time slot interval $[t, d]$, for any $t \in [d]^+$, where $d = \max_{T_i \in \mathcal{T}}\{d_i\}$. We denote by $\lambda_t(\mathcal{S})$ *the maximum workload of $\mathcal{S}$ that could be executed in $[t, d]$ in an idealized case* where there is an indefinite number of machines, i.e., $C = \infty$, for any $t \in [d]^+$. In the case where C is finite, we denote by $\lambda_t^C(\mathcal{S})$ *the maximum workload of $\mathcal{S}$ that can be executed on the C machines in* $[t, d]$, where $t \in [d]^+$. We denote by β_i the maximum workload of an individual task $T_i \in \mathcal{S}$ that could be executed in the discrete time slot interval $[t, d]$.

Definition 1 By Eq. (3), (5) and (6), we have

$$\beta_i = \min\{k_i \cdot \max\{d_i - t + 1, 0\}, D_i\} \tag{8}$$

where $d_i - t + 1$ is the number of slots in $[t, d_i]$ if $t \leq d_i$. The definition of β_i is illustrated in Fig. 1. In the case where $C = \infty$, the capacity constraint is ignored and in Eq. (7), $W(t) \in [0, \infty)$ for any $t \in [d]^+$. For any $t \in [d]^+$, $\lambda_t(\mathcal{S})$ is defined as follows:

$$\lambda_t(\mathcal{S}) = \sum\nolimits_{T_i \in \mathcal{S}} \beta_i. \tag{9}$$

Let $\lambda_{d+1}^C(\mathcal{S}) = 0$. For any $t \in [d]^+$, we set:

$$\lambda_t^C(\mathcal{S}) = \lambda_{t+1}^C(\mathcal{S}) + \min\left\{\lambda_t(\mathcal{S}) - \lambda_{t+1}^C(\mathcal{S}), C\right\}. \tag{10}$$

As illustrated in Fig. 2, $\lambda_d^C(\mathcal{S}) = \min\{C, \lambda_d(\mathcal{S})\} = C$ and $\lambda_{d-1}^C(\mathcal{S}) = \lambda_d^C(\mathcal{S}) + (\lambda_{d-1}(\mathcal{S}) - \lambda_d^C(\mathcal{S})) = \lambda_{d-1}(\mathcal{S})$.

By Eq. (10), we have

$$\lambda_t^C(\mathcal{S}) \leq \lambda_t(\mathcal{S}) \text{ for any } t \in [d]^+. \tag{11}$$

An allocation to $\mathcal{S}$ is $\{y_i(\tilde{t}) \mid T_i \in \mathcal{S}, \tilde{t} \in [1, d_i]\}$ that satisfies Eq. (6) and (7). For any $t \in [d]^+$, we have:

$$\sum\nolimits_{T_i \in \mathcal{S}} \sum\nolimits_{\tilde{t}=t}^{d} y_i(\tilde{t}) \leq \lambda_t(\mathcal{S}). \tag{12}$$

where $\sum_{\tilde{t}=t}^{d} y_i(\tilde{t}) \leq \beta_i$. Formally, we have

Proposition 1 *For any $t \in [d]^+$, $\lambda_t^C(\mathcal{S})$ is the maximum workload of $\mathcal{S}$ that can be executed on the C machines in $[t, d]$. In other words, for any feasible allocation to $\mathcal{S}$, we have:*

$$\sum\nolimits_{T_i \in \mathcal{S}} \sum\nolimits_{\tilde{t}=t}^{d} y_i(\tilde{t}) \leq \lambda_t^C(\mathcal{S}). \tag{13}$$

Proof We prove by contradiction. Suppose there exists an allocation to $\mathcal{S}$ that satisfies

$$\sum\nolimits_{T_i \in \mathcal{S}} \sum\nolimits_{\tilde{t}=t}^{d} y_i(\tilde{t}) > \lambda_t^C(\mathcal{S}). \tag{14}$$

Then, there exists a slot $t_0 \in [t, d]$ such that

$$\lambda_{t_0}^C(\mathcal{S}) - \lambda_{t_0+1}^C(\mathcal{S}) < C; \tag{15}$$

otherwise, $\lambda_t^C(\mathcal{S}) = C \cdot (d - t + 1) \overset{(a)}{\geq} \sum_{T_i \in \mathcal{S}} \sum_{\tilde{t}=t}^{d} y_i(\tilde{t})$, which contradicts Eq. (14), where inequality (a) is due to Eq. (7). With abuse of notation, let t_0 be the earliest such slot in $[t, d]$. By Eq. (10) and (15), we have

$$\lambda_{t_0}^C(\mathcal{S}) = \lambda_{t_0}(\mathcal{S}). \tag{16}$$

If $t_0 = t$, Eq. (16) and (14) contradict Eq. (12). If $t_0 > t$, we have

$$\lambda_t^C(\mathcal{S}) = \lambda_{t_0}(\mathcal{S}) + C \cdot (t_0 - t). \tag{17}$$

We have

$$\begin{aligned} &\sum\nolimits_{T_i \in \mathcal{S}} \sum\nolimits_{\tilde{t}=t}^{t_0-1} y_i(\tilde{t}) \overset{(b)}{>} \lambda_t^C(\mathcal{S}) - \sum\nolimits_{T_i \in \mathcal{S}} \sum\nolimits_{\tilde{t}=t_0}^{d} y_i(\tilde{t}) \\ &\overset{(c)}{\geq} \lambda_t^C(\mathcal{S}) - \lambda_{t_0}(\mathcal{S}) \overset{(d)}{=} C \cdot (t_0 - t), \end{aligned}$$

which violates the capacity constraint Eq. (7); herein, inequality (b) is due to Eq. (14), (c) is due to Eq. (12), and (d) is due to Eq. (17). ■

For any $t \in [d]$, we define

$$\mu_t^C(\mathcal{S}) = \sum\nolimits_{T_i \in \mathcal{S}} D_i - \lambda_{t+1}^C(\mathcal{S}). \tag{18}$$

By Proposition 1, for any $t \in [d]^+$, $\mu_t^C(\mathcal{S})$ denotes the remaining (minimum) workload of $\mathcal{S}$ that needs to be processed in $[1, t]$ on the C machines. In this paper, given a set $\mathcal{S}$ of tasks, the inequality below

$$\mu_t^C(\mathcal{S}) \leq C \cdot t \quad \text{for all } t \in [d-1] \tag{19}$$

is referred to as ***boundary condition***.

3.2 Sufficiency and Necessity

In Section 3.2.1 and 3.2.2, we give elements that are needed to prove that there exists a feasible allocation to the tasks of $\mathcal{S}$ that satisfies Eq. (6) and (7), if $\mathcal{S}$ satisfies the boundary condition in Eq. (19). In Section 3.2.3, we give the core result of this paper.

3.2.1 Transformation to Simpler Tasks

Below, we show that the problem of scheduling malleable tasks can be transformed into a problem of scheduling non-malleable preemptive tasks where their parallelism bounds equal one.

Decomposition Procedure. For any task $T_i \in \mathcal{T}$, let $n_i = D_i$ if $D_i < k_i$, and $n_i = k_i$ otherwise, where

$$n_i \leq k_i. \tag{20}$$

T_i can be decomposed into n_i smaller tasks $T_{i,1}, T_{i,2}, \cdots, T_{i,n_i}$. For any $j \in [n_i]^+$, the task $T_{i,j}$ is defined as follows where $len_i = \lceil D_i/k_i \rceil$:

1. The deadline $d_{i,j}$ of $T_{i,j}$ is d_i.
2. The demand $D_{i,j}$ of $T_{i,j}$ is set as follows:
   - If $D_i < k_i$, then $D_{i,j} = 1$.
   - If $D_i/k_i = len_i$, then $D_{i,j} = len_i$.
   - Consider the case where $D_i/k_i < len_i$ and $len_i \geq 2$. If $j \in [D_i - k_i \cdot (len_i - 1)]^+$, then $D_{i,j} = len_i$; if $j \in [D_i - k_i \cdot (len_i - 1) + 1, k_i]$, then $D_{i,j} = len_i - 1$.

   Here, we have:

$$D_i = \sum\nolimits_{j=1}^{n_i} D_{i,j} \text{ where } 0 < D_{i,j} \leq len_i. \tag{21}$$

3. The parallelism bound $k_{i,j}$ of $T_{i,j}$ is one. Let $y_{i,j}(t) \in \{0, 1\}$ denote the number of machines allocated to $T_{i,j}$ at any slot $t \in [d_i]^+$.

Like Eq. (6), for any $j \in [n_i]^+$, a task $T_{i,j}$ is finished by the deadline d_i when we have:

$$\sum\nolimits_{t \leq d_i} y_{i,j}(t) = D_{i,j}, \text{ subject to } y_{i,j}(t) \in \{0,1\}. \tag{22}$$

Let $\hat{\mathcal{S}}_i = \{T_{i,1}, T_{i,2}, \cdots, T_{i,n_i}\}$ for any $T_i \in \mathcal{S}$ and

$$\hat{\mathcal{S}} = \bigcup\nolimits_{T_i \in \mathcal{S}} \hat{\mathcal{S}}_i. \tag{23}$$

Like Eq. (7), for any $t \in [d]^+$, the capacity constraint is as follows:

$$\sum\nolimits_{T_{i,j} \in \hat{\mathcal{S}}} y_{i,j}(t) \leq C. \tag{24}$$

Lemma 1 $\lambda_t^C(\hat{\mathcal{S}}) = \lambda_t^C(\mathcal{S})$ *for any* $t \in [d]^+$.

Proof By Definition 1 and Eq. (23), we have

$$\lambda_t(\hat{\mathcal{S}}) = \sum\nolimits_{T_{i,j} \in \hat{\mathcal{S}}} \beta_{i,j} = \sum\nolimits_{T_i \in \mathcal{S}} \sum\nolimits_{j=1}^{n_i} \beta_{i,j} \tag{25}$$

where $\beta_{i,j} = \min\{(d_i - t + 1)^+, D_{i,j}\}$. Each task $T_i \in \mathcal{S}$ corresponds to a set $\hat{\mathcal{S}}_i$ of tasks with the same deadline d_i. There are three cases:

- If $d_i < t$, we have $\beta_i = \beta_{i,j} = 0$ where $j \in [n_i]^+$.
- If $d_i \geq t$ and $d_i - (t-1) < len_i$, we have $\beta_i = k_i \cdot (d - t + 1)$, $\beta_{i,j} = d - t + 1$ and $n_i = k_i$.
- If $d_i \geq t$ and $d_i - (t-1) \geq len_i$, we have $\beta_i = D_i$ and $\beta_{i,j} = D_{i,j}$; herein, Eq. (21) holds.

In each case, we finally have $\beta_i = \sum_{j=1}^{n_i} \beta_{i,j}$. Thus, by Eq. (25), we have $\lambda_t(\hat{\mathcal{S}}) = \sum_{T_i \in \mathcal{S}} \beta_i = \lambda_t(\mathcal{S})$. By Eq. (10), $\lambda_t^C(\hat{\mathcal{S}})$ only relates to $\lambda_{t+1}^C(\hat{\mathcal{S}})$, $\lambda_t(\hat{\mathcal{S}})$ and C where $\lambda_{d+1}^C(\hat{\mathcal{S}}) = \lambda_{d+1}^C(\mathcal{S}) = 0$; thus, $\lambda_t^C(\hat{\mathcal{S}}) = \lambda_t^C(\mathcal{S})$ for any $t \in [d]^+$. ■

Lemma 2 *An allocation to* $\hat{\mathcal{S}}$ *that satisfies Eq. (22) and (24) can be transformed into an allocation to* $\mathcal{S}$ *that satisfies Eq. (6) and (7), with a time complexity of* $\mathcal{O}(nkd)$, *where we have:*

$$y_i(t) = \sum_{j=1}^{n_i} y_{i,j}(t) \quad \text{for any } T_i \in \mathcal{S} \text{ and } t \in [d_i]^+. \tag{26}$$

Proof We have $y_i(t) \in [k_i]$ since $y_{i,j}(t) \in \{0,1\}$ and $n_i \leq k_i$ by Eq. (20). By Eq. (21), we have $\sum_{t \leq d_i} y_i(t) = \sum_{j=1}^{n_i} \sum_{t \leq d_i} y_{i,j}(t) = \sum_{j=1}^{n_i} D_{i,j} = D_i$. For an individual task $T_i \in \mathcal{S}$, by Eq. (26), the time complexity of computing $\{y_i(t)\}_{t=1}^{d_i}$ is $\mathcal{O}(k \cdot d)$ where $n_i \leq k$ and $d_i \leq d$. There are at most n tasks where $|\mathcal{S}| \leq n$. The total time complexity of transformation is $\mathcal{O}(nkd)$. ■

3.2.2 Algorithm

Below, for ease of exposition, we also use $\{\hat{T}_1, \hat{T}_2, \cdots, \hat{T}_{\hat{n}}\}$ to represent the set $\hat{\mathcal{S}}$ in Eq. (23) where $\hat{n} = \sum_{T_i \in \mathcal{S}} n_i \leq k \cdot n$ and consider scheduling $\hat{\mathcal{S}}$ on the C machines. For any $\hat{T}_i \in \hat{\mathcal{S}}$, its deadline is $\hat{d}_i$, its demand is $\hat{D}_i$, its parallelism bound is $\hat{k}_i = 1$, and its allocation at slot $t \in [\hat{d}_i]^+$ is $\hat{y}_i(t) \in \{0, 1\}$. The minimum execution time of $\hat{T}_i \in \hat{\mathcal{S}}$ is $\hat{len}_i = \hat{D}_i$. Initially, we set $\hat{y}_i(t) = 0$ for any $\hat{T}_i \in \hat{\mathcal{S}}$ and $t \in [d]^+$. Let D'_i denote the remaining workload of $\hat{T}_i$ to be processed and $D'_i = \hat{D}_i$. The algorithm LDF($\hat{\mathcal{S}}$) is formally presented in Algorithm 1 where time slots t are considered from the latest slot d towards the earlier ones. At each iteration t, let $\mathcal{K}_t$ denote the set of the tasks $\hat{T}_i$ whose deadlines are no smaller than t and whose remaining workloads D'_i are larger than zero (line 3). Subject to the capacity constraint, we choose $\min\{C, |\mathcal{K}_t|\}$ tasks from $\mathcal{K}_t$ with the largest remaining workloads denoted as $\mathcal{K}'_t$ (line 4) and allocate one machine to each such task $\hat{T}_i$ at slot t (line 5): $D'_i = D'_i - 1$.

Algorithm 1: LDF($\hat{\mathcal{S}}$)

```
1 For all tasks $\hat{T}_i \in \hat{\mathcal{S}}$, set $D'_i = \hat{D}_i$, and $\hat{y}_i(t) = 0$ for all $t \in [d]^+$.// $D'_i$: record the
    remaining workload of $\hat{T}_i$ to be processed.
2 for $t \leftarrow d$ to 1 do
3     Let $\mathcal{K}_t = \{\hat{T}_i : D'_i > 0 \ \& \ \hat{d}_i \geq t\}$.
4     Let $\mathcal{K}'_t$ be the set of the $\min\{C, |\mathcal{K}_t|\}$ tasks in $\mathcal{K}_t$ with largest $D'_i$: sort the tasks
        of $\mathcal{K}_t$ in the non-increasing order of their remaining workloads $D'_i$ and then
        select the first $\min\{C, |\mathcal{K}_t|\}$ tasks.
5     Set $\hat{y}_i(t) = 1$ and $D'_i = D'_i - 1$ for all $\hat{T}_i \in \mathcal{K}'_t$.
```

Lemma 3 *Consider two slots $t < t_0$ and a task $\hat{T}_{i'} \in \mathcal{K}_{t_0}$, where t_0 is the earliest slot in $[t+1, d]$ such that $\hat{y}_{i'}(t_0) = 0$ and $\hat{y}_{i'}(\tilde{t}) = 1$ for any $\tilde{t} \in [t, t_0 - 1]$. If we have $D'_{i'} \geq 2$ at the beginning of iteration t and $|\mathcal{K}_{t_0}| > C$, then we have $\mathcal{K}_{t_0} \subseteq \mathcal{K}_t$.*

Proof At the beginning of iteration t_0, we have

$$D'_{i'} \geq 2 + (t_0 - 1) - t = t_0 - t + 1; \tag{27}$$

by the definition of $\mathcal{K}'_{t_0}$ in line 4, we have that $\hat{T}_{i'} \notin \mathcal{K}'_{t_0}$ and $|\mathcal{K}'_{t_0}| = C$, since $\hat{y}_{i'}(t_0) = 0$; also, we have

$$D'_i \geq D'_{i'} \text{ for any task } \hat{T}_i \in \mathcal{K}'_{t_0}. \tag{28}$$

At each iteration, the remaining workload of a task decreases by at most one (line 5); for any task $\hat{T}_i \in \mathcal{K}'_{t_0}$, by Eq. (27) and (28), we have

$$D'_i \geq (t_0 - t + 1) - \big(t_0 - \tilde{t}\big) \geq \tilde{t} - t + 1 \geq 2$$

at the beginning of each iteration $\tilde{t} \in [t+1, t_0]$; thus, no tasks of $\mathcal{K}_{t_0}$ whose remaining workloads are smaller than 2 are included into $\mathcal{K}'_{\tilde{t}}$. Thus, no tasks of $\mathcal{K}_{t_0}$ have their remaining workload D'_i to be zero at the beginning of iteration t, and we have $\mathcal{K}_{t_0} \subseteq \mathcal{K}_t$. ■

Proposition 2 *If $\hat{\mathcal{S}}$ satisfies the boundary condition, LDF($\hat{\mathcal{S}}$) will produce a feasible allocation to $\hat{\mathcal{S}}$ on the C machines with a time complexity of $\mathcal{O}(nkd\log(kn))$.*

Proof For the feasible allocation, it suffices to show that, upon completion of each iteration $t \in [1, d]$, LDF($\hat{\mathcal{S}}$) will produce a feasible allocation to $\hat{\mathcal{S}}$ on the C machines that satisfies

$$\sum\nolimits_{T_i\in\hat{\mathcal{S}}}\sum\nolimits_{\tilde{t}=t}^{d}\hat{y}_i(\tilde{t}) = \lambda_t^C(\hat{\mathcal{S}}). \tag{29}$$

Then, upon completion of iteration $t = 1$, the remaining workload of $\hat{\mathcal{S}}$ to be executed is $\mu_0^C(\mathcal{S}) \leq 0$ by Eq. (19). Thus, all tasks of $\hat{\mathcal{S}}$ are completed.

Now, we prove Eq. (29) by induction. Let $\hat{\mathcal{K}}_t = \{\hat{T}_i \in \hat{\mathcal{S}} | \hat{d}_i \geq t\}$ denote the tasks whose deadlines are no smaller than t; then, we have

$$\lambda_t(\hat{\mathcal{S}}) \overset{(a)}{=} \lambda_t(\hat{\mathcal{K}}_t) \text{ and } \lambda_t^C(\hat{\mathcal{S}}) \overset{(b)}{=} \lambda_t^C(\hat{\mathcal{K}}_t). \tag{30}$$

where (a) is due to Eq. (8) and (9) and (b) is due to Eq. (10). *First*, we consider the iteration $t = d$ where

$$\hat{\mathcal{K}}_d = \mathcal{K}_d. \tag{31}$$

By Eq. (9), (30) and (31), we have $\lambda_d(\hat{\mathcal{S}}) = \lambda_d(\mathcal{K}_d) = |\mathcal{K}_d|$. Thus, by Eq. (10), we have

$$\lambda_d^C(\hat{\mathcal{S}}) = \min\{\lambda_d(\hat{\mathcal{S}}), C\} = \min\{|\mathcal{K}_d|, C\}.$$

Upon completion of the iteration $t = d$, the allocation in line 5 satisfies

$$\sum_{\hat{T}_i\in\hat{\mathcal{S}}}\hat{y}_i(d) = \sum_{\hat{T}_i\in\mathcal{K}'_d}\hat{y}_i(d) \overset{(a)}{=} \min\{C, |\mathcal{K}_t|\} = \lambda_d^C(\hat{\mathcal{S}}).$$

where equality (a) is due to the choice of $\mathcal{K}'_d$ in line 4.

Second, suppose Eq. (29) holds upon completion of the iteration $t+1$. In the rest of this proof, we analyze the case upon completion of the iteration $t \in [1, d-1]$. If $|\mathcal{K}_t| \geq C$, we have $|\mathcal{K}'_t| = C$ and the allocation upon completion of the iteration t satisfies

$$\sum\nolimits_{\hat{T}_i\in\hat{\mathcal{S}}}\hat{y}_i(t) = \sum\nolimits_{\hat{T}_i\in\mathcal{K}'_t}\hat{y}_i(t) = C \tag{32}$$

Thus, we have

$$\sum\nolimits_{\tilde{t}=t}^{d}\sum\nolimits_{\hat{T}_i\in\hat{\mathcal{S}}}\hat{y}_i(t) \overset{(b)}{=} C + \lambda_{t+1}^C(\hat{\mathcal{S}}) \overset{(c)}{=} \lambda_t^C(\hat{\mathcal{S}}) \tag{33}$$

where equality (b) is due to the induction hypothesis and Eq. (32). By Eq. (10) and (13), we have

$$\sum\nolimits_{\tilde{t}=t}^{d}\sum\nolimits_{\hat{T}_i\in\hat{\mathcal{S}}}\hat{y}_i(t) \leq \lambda_t^C(\hat{\mathcal{S}}) \leq C + \lambda_{t+1}^C(\hat{\mathcal{S}}).$$

Together with the equality (b), we have that the equality (c) holds. By Eq. (33), we have that Eq. (29) holds if $|\mathcal{K}_t| \geq C$.

If $|\mathcal{K}_t| < C$, we prove the following conclusion for every task $\hat{T}_i \in \hat{\mathcal{K}}_t$ upon completion of the iteration t:

Algorithm 2: *Sched*$(\mathcal{S})$

```
1 Apply the decomposition procedure in Section 3.2.1 to decompose the tasks of $\mathcal{S}$ into
    the tasks of $\hat{\mathcal{S}}$;
2 Call LDF($\hat{\mathcal{S}}$), presented in Algorithm 1;
3 Apply Eq. (26) to transform the allocation of $\hat{\mathcal{S}}$ to the allocation of $\mathcal{S}$;
```

(a) If $\hat{d}_i - (t-1) < \hat{len}_i$, we have $\hat{y}_i\left(\tilde{t}\right) = 1$ for all $\tilde{t} \in [t, d_i]$ and $\sum_{\tilde{t}=t}^{d} \hat{y}_i(\tilde{t}) = \hat{d}_i - (t-1)$.
(b) If $\hat{d}_i - (t-1) \geq \hat{len}_i$, we have $\sum_{\tilde{t}=t}^{d} \hat{y}_i(\tilde{t}) = \hat{D}_i$ (i.e., $D_i' = 0$).

Then, by Eq. (8) and (9), we have

$$\sum\nolimits_{\hat{T}_i \in \hat{\mathcal{S}}} \sum\nolimits_{\tilde{t}=t}^{d} \hat{y}_i(\tilde{t}) = \lambda_t(\hat{\mathcal{S}}). \tag{34}$$

By Eq. (11) and (13), we have Eq. (29) holds if $|\mathcal{K}_t| < C$.

Now, we prove the conclusion above. Suppose there exists a task $\hat{T}_{i'} \in \hat{\mathcal{K}}_t$ whose allocation violates the conclusion. Then, in both cases of the conclusion, we have

$$D_{i'}' = \hat{D}_{i'} - \sum\nolimits_{\tilde{t}=t}^{d} \hat{y}_{i'}(\tilde{t}) \geq 1 \tag{35}$$

upon completion of iteration t; further, at iteration t, we have $\hat{T}_{i'} \in \mathcal{K}_t'$ and

$$\hat{y}_{i'}(t) = 1 \tag{36}$$

since $|\mathcal{K}_t| < C$. By Eq. (35) and (36), we have at the beginning of iteration t that

$$D_{i'}' = \hat{D}_{i'} - \sum\nolimits_{\tilde{t}=t+1}^{d} \hat{y}_{i'}(\tilde{t}) \geq 2.$$

Due to Eq. (36), we also have that there exists a slot $t_0 \in [t+1, \hat{d}_{i'}]$ such that $\hat{y}_{i'}(t_0) = 0$ upon completion of iteration t_0; otherwise, the conclusion above holds for $\hat{T}_{i'}$. Let t_0 be the earliest such slot and we have $|\mathcal{K}_{t_0}| > C$ since $\hat{y}_{i'}(t_0) = 0$. By Lemma 3, we have $\mathcal{K}_{t_0} \subseteq \mathcal{K}_t$. Thus, we have $|\mathcal{K}_t| > C$, which contradicts the original fact that $|\mathcal{K}_t| < C$.

Finally, at each iteration of LDF($\hat{\mathcal{S}}$) from d to 1, a sorting operation is done in line 4, which leads to a time complexity of $\mathcal{O}(\hat{n} \cdot \log \hat{n})$. Thus the total time complexity of LDF($\hat{\mathcal{S}}$) is $\mathcal{O}(d \cdot \hat{n} \cdot \log \hat{n}) \leq \mathcal{O}(nkd \log(kn))$ where $\hat{n} \leq k \cdot n$. ■

3.2.3 Core Result

Theorem 1 *There exists a feasible allocation to $\mathcal{S}$ on the C machines* if and only if *$\mathcal{S}$ satisfies the boundary condition in Eq. (19). The time complexity of giving a feasible allocation is $\mathcal{O}(nkd \log(kn))$.*

Algorithm 3: Greedy

```
1 $\mathcal{A} = \emptyset$;                                  // Record the set of tasks accepted
2 $y_i(t) = 0$ for all $T_i \in \mathcal{T}$ and $t \in [d]^+$;
3 $v_1/D_1 \geq v_n/D_n \geq \cdots \geq v_n/D_n$;
4 for $i \leftarrow 1$ to $n$ do
5     if $\sum_{t=1}^{d_i} \min\{k_i, \overline{W}(t)\} \geq D_i$ then
6         Accept $T_i$: $\mathcal{A} = \mathcal{A} \cup \{T_i\}$;
7         Allocate machines to $T_i$: Allocate($i$);
```

Proof We first prove the "if" direction, i.e., Algorithm 2 can produce a feasible allocation if $\mathcal{S}$ satisfies the boundary condition. By Eq. (21) and (23), we have $\sum_{T_i\in\mathcal{S}} D_i = \sum_{\hat{T}_i\in\hat{\mathcal{S}}} \hat{D}_i$. By Lemma 1, we have $\mu_t^C(\mathcal{S}) = \mu_t^C(\hat{\mathcal{S}})$. Thus, if $\mathcal{S}$ satisfies Eq. (19), $\hat{\mathcal{S}}$ satisfies Eq. (19). By Proposition 2, LDF($\hat{\mathcal{S}}$) can give a feasible allocation to $\hat{\mathcal{S}}$ with a time complexity of $\mathcal{O}(nkd\log(kn))$ and by Lemma 2 there exists a feasible allocation to $\mathcal{S}$ with a time complexity of $\mathcal{O}(nkd)$. Thus, if $\mathcal{S}$ satisfies the boundary condition, Algorithm 2 can produce a feasible allocation to $\mathcal{S}$ on the C machines with a time complexity of $\mathcal{O}(nkd\log(kn))$.

Next, we prove the "only if" direction. For any $t \in [d]^+$, the workload of $\mathcal{S}$ processed in $[1, t]$ does not exceed the capacity constraint Eq. (7):

$$\sum_{T_i\in\mathcal{S}} \sum_{\tilde{t}=1}^{t} y_i(\tilde{t}) \leq C \cdot t. \tag{37}$$

By Proposition 1, we also have

$$\sum_{T_i\in\mathcal{S}} \sum_{\tilde{t}=1}^{t} y_i(\tilde{t}) \geq \sum_{T_i\in\mathcal{S}} D_i - \lambda_{t+1}^C(\mathcal{S}) = \mu_t^C(\mathcal{S}). \tag{38}$$

By Eq. (37) and (38), the "only if" direction holds. ■

4 Applications: Part I

In this section, we consider the objective of social welfare maximization and illustrate the application of the results in Section 3 to the greedy strategy [1,23].

4.1 Generic Greedy Algorithm

A generic greedy algorithm framework is presented in Algorithm 3. We use $\mathcal{A} \subseteq \mathcal{T}$ to denote the set of tasks chosen and included into the solution. Initially, $\mathcal{A} = \emptyset$ and $y_i(t) = 0$ for all $T_i \in \mathcal{T}$ and $t \in [d]^+$. The set below

$$\mathcal{Y} = \{y_i(t) | T_i \in \mathcal{T}, t \in [d]^+\}$$

defines the current machine allocation state. Algorithm 3 have three components and works as follows:

Algorithm 4: Fully-Utilize(i)

```
1 for $t \leftarrow d_i$ to $1$ do
2     $y_i(t) \leftarrow \min\left\{k_i, \overline{W}(t), D_i - \sum_{\bar{t}=t+1}^{d_i} y_i(\bar{t})\right\}$.
```

(i) *Order*: Consider tasks in non-increasing order of their marginal values (lines 3-4). The marginal value of a task T_i is defined as $v'_i = v_i/D_i$ and is the criterion used to judge the task efficiency, i.e., the value obtained when it utilizes one machine per unit of time. We assume without loss of generality that

$$v'_1 \geq v'_2 \geq \cdots \geq v'_n; \tag{39}$$

(ii) *Accepting condition*: At each iteration i, the task T_i is the most efficient among the remaining tasks $\{T_i, \cdots, T_n\}$; then, consider whether to accept T_i or not according to some condition (lines 5-6).

(iii) *Allocation algorithm*: If the task T_i being considered is accepted, use some allocation algorithm, denoted as Allocate(i), to define the allocation of machines to T_i: $\{y_i(t) | t \in [d_i]^+\}$ (line 7).

In Algorithm 3, we don't specify the allocation algorithm Allocate(i). With different specification of Allocate(i), Algorithm 3 can have many instances and represents a class of greedy algorithms, which is referred to as *Greedy*. For example, in [7], two greedy algorithms are proposed and are in fact two instances of Algorithm 3. The preliminary and final algorithms in [7] are called SimpleGreedy and GreedyRTL respectively. In SimpleGreedy, the allocation algorithm can be simply specified as Algorithm 4. As shown in [7], the performance guarantee of SimpleGreedy is $1/(1 + \frac{C}{C-k} \cdot \frac{s}{s-1})$. Further, the authors propose another specific allocation algorithm to improve SimpleGreedy; the resulting greedy algorithm GreedyRTL achieves a performance guarantee $\frac{C-k}{C} \cdot \frac{s-1}{s}$.

In this paper, we hope to understand the best possible performance guarantee that Algorithm 3 can achieve, no matter what allocation algorithm Allocate(i) are applied. Afterwards, we manage to instantiate Algorithm 3 that can achieve the best possible performance guarantee.

Proposition 3 *For any algorithm in the class* Greedy *and an arbitrarily small $\epsilon > 0$, there exists an instance with minimum slackness s such that the social welfare achieved by the schedule constructed by the algorithm is at most $(1+\epsilon)(\frac{s-1}{s} + \epsilon)$ times the social welfare achieved by an optimal schedule.*

Proof Let us consider a special instance:

(i) The number of machines is one, i.e., $C = 1$. There are $n = 5$ tasks. Each task T_i has a deadline d_i, a workload D_i, a parallelism bound k_i, and a marginal value $v'_i = \frac{v_i}{D_i}$.

(ii) The first $n-1$ tasks have the same features. Let $d'_1 = n - 1 = 4$, and for any task T_i with $i \in [1, n-1]$, we set $d_i = d'_1$, $D_i = 1$, $k_i = 1$ and $v'_i = 1 + \epsilon$.

(iii) Let

$$d'_2 = \lceil 1/\epsilon \rceil \gg 2d'_1 = 8. \tag{40}$$

For the last task T_n, we set $d_n = d'_2$, $D_n = d'_2 - d'_1 + 1$, $k_n = 1$ and $v'_n = 1$.

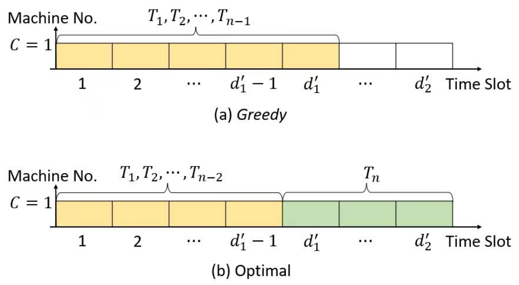


Fig. 3: The allocation to the chosen tasks where $d_1' = n - 1 = 5$ and $d_2' = \lceil 1/\epsilon \rceil \gg 2d_1'$.

In this instance, by Eq. (3) and (4), we have

$$s = \min\left\{d_1', \frac{d_2'}{d_2' - d_1' + 1}\right\} \overset{(a)}{=} \frac{d_2'}{d_2' - d_1' + 1} \tag{41}$$

where (a) is due to that $\frac{d_2'}{d_2'-d_1'+1} = \frac{1}{1-(d_1'-1)/d_2'} \leq 2 \leq d_1'$ where $(d_1'-1)/d_2' \leq \frac{1}{2}$ by Eq. (40). The first n tasks have a smaller deadline than T_n, and a larger marginal value than T_n. By Algorithm 3, Greedy will always fully allocate resource to the first $d_1' = n-1$ tasks as illustrated in Fig. 3(a); afterwards, since each task occupies one time slot, $\sum_{t=1}^{d_1'} W(t) = d_1'$ and $\overline{W}(t) = 0$ for all $t \in [d_1']^+$; finally, the task T_n is rejected since $\sum_{t=1}^{d_2'} \overline{W}(t) = d_2' - d_1' < D_n$. The social welfare achieved by Greedy is $(1+\epsilon)d_1'$. On the other hand, the optimal schedule will execute the task T_n in the interval $[d_1', d_2']$ and execute the first $n-2$ tasks in $[1, d_1'-1]$, as illustrated in Fig. 3(b). The optimal social welfare is $(1+\epsilon)(d_1'-1) + (d_2' - d_1' + 1)$. Let $\rho' = \frac{(1+\epsilon)d_1'}{(1+\epsilon)(d_1'-1)+(d_2'-d_1'+1)}$. Then, $\rho' = \frac{(1+\epsilon)d_1'}{\epsilon(d_1'-1)+d_2'} \leq (1+\epsilon)\frac{d_1'}{d_2'}$. By Eq. (41), we have $\frac{s-1}{s} = \frac{d_1'-1}{d_2'}$. Thus, $\rho' \leq (1+\epsilon)(\frac{s-1}{s} + \frac{1}{d_2'}) \leq (1+\epsilon)(\frac{s-1}{s} + \epsilon)$ by Eq. (40), and the proposition holds. ■

A ρ-approximation algorithm G is defined to be an algorithm for which the social welfare $f(x)$ of the approximate solution $G(x)$ to any instance x will not be no less than a factor ρ times the value OPT of an optimal solution. By Proposition 3, for any algorithm in the class *Greedy*, an upper bound of its approximation ratio ρ is $(1+\epsilon)(\frac{s-1}{s} + \epsilon)$ whose value is close to $\frac{s-1}{s}$.

4.2 Notation

Greedy considers tasks sequentially. The first considered task will be accepted definitely and then we will use the feasibility condition to determine whether to accept or reject the next task according to the current available resource and the characteristics of this task. To describe the process under which Greedy accepts or rejects tasks, we define the sets of consecutive accepted (i.e., fully allocated) and rejected tasks $\mathcal{A}_1, \mathcal{R}_1, \mathcal{A}_2, \cdots$. Specifically, let $\mathcal{A}_m = \{T_{i_m}, T_{i_m+1}, \cdots, T_{j_m}\}$ be the m-th set of the adjacent tasks that are accepted by Greedy where $i_1 = 1$ while

$\mathcal{R}_m = \{T_{j_m+1}, \cdots, T_{i_{m+1}-1}\}$ is the m-th set of the adjacent tasks that are rejected following the set $\mathcal{A}_m$, where $m \in [K]^+$ for some integer K. Integer K represents the last step: in the K-th step, $\mathcal{A}_K \neq \emptyset$ and $\mathcal{R}_K$ can be empty or non-empty; herein, we have

$$\mathcal{T} = \mathcal{A}_1 \cup \mathcal{R}_1 \cup \mathcal{A}_2 \cup \cdots \cup \mathcal{R}_K \tag{42}$$

$$\mathcal{A} = \bigcup\nolimits_{m=1}^{K} \mathcal{A}_m \tag{43}$$

$$\sum\nolimits_{t \leq d_i} y_i(t) = \begin{cases} D_i & \text{if } T_i \in \mathcal{A} \\ 0 & \text{if } T_i \in \cup_{m=1}^{K} \mathcal{R}_m \end{cases} \tag{44}$$

We also denote by c_m the maximum deadline of all rejected tasks of $\cup_{l=1}^{m} \mathcal{R}_l$, i.e.,

$$c_m = \max\nolimits_{T_i \in \cup_{l=1}^{m} \mathcal{R}_l} \{d_i\},$$

and by c'_m the maximum deadline of $\cup_{l=1}^{m} \mathcal{A}_l$, i.e.,

$$c'_m = \max\nolimits_{T_i \in \cup_{l=1}^{m} \mathcal{A}_l} \{d_i\}.$$

While the tasks in $\mathcal{A}_m \cup \mathcal{R}_m$ are being considered, we refer to Greedy as being in the m-th phase. Before the execution of Greedy, we refer to it as being in the 0-th phase. Upon completion of the m-th phase of Greedy, we define a *threshold* parameter t_m^{th} such that

(i) if $c_m \geq c'_m$, set $t_m^{\text{th}} = c_m$, and
(ii) if $c_m < c'_m$, set t_m^{th} to any time slot in $[c_m, c'_m]$.

Here, $d_i \leq t_m^{th}$ for all $T_i \in \cup_{j=1}^{m} \mathcal{R}_j$. For ease of exposition, we let $t_0^{th} = 0$.

4.3 A New Algorithmic Analysis

The main result of this subsection is Theorem 2. We will show that once the resource allocation done by Greedy satisfies some features, its performance guarantee can be deduced immediately. For all $m \in [K]^+$, upon completion of Greedy, we define the following two features that we want the allocation to $\cup_{j=1}^{m} \mathcal{A}_j$ to satisfy:

Feature 1 *The total allocation to $\bigcup_{j=1}^{m} \mathcal{A}_j$ in $[1, t_m^{th}]$ is at least $r \cdot C \cdot t_m^{th}$, where $r \in [0, 1]$, i.e.,*

$$\sum\nolimits_{T_i \in \cup_{j=1}^{m} \mathcal{A}_j} \sum\nolimits_{\bar{t}=1}^{t_m^{th}} y_i(\bar{t}) \geq r \cdot C \cdot t_m^{th}.$$

Feature 2 *For any task $T_i \in \bigcup_{j=1}^{m} \mathcal{A}_j$ and all $l \in [m, K]$, the maximum workload of T_i is processed in $\left[t_l^{th} + 1, d\right]$, i.e.,*

$$\sum\nolimits_{\tilde{t}=t_l^{th}+1}^{d_i} y_i(\tilde{t}) = \min\left\{D_i, k_i \cdot (d_i - t_l^{th})^+\right\}.$$

Theorem 2 *If Greedy achieves a resource allocation structure that satisfies Feature 1 and Feature 2 for all $m \in [K]^+$, it gives an r-approximation to the optimal social welfare.*

In the rest, we prove Theorem 2; we will first provide an upper bound of the optimal social welfare.

Proof Overview. We refer to the original problem of scheduling $\mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_K, \mathcal{R}_K$ on C machines to maximize the social welfare as **the MSW-I problem**. The total value generated by Greedy is as follows:

$$V_1 = \sum\nolimits_{T_i \in \mathcal{A}} v_i = \sum\nolimits_{T_i \in \mathcal{A}} D_i \cdot v_i' \tag{45}$$

where $\mathcal{A}$ is given in Eq. (43).

In the following, we define a relaxed version of the MSW-I problem. For any $m \in [K]^+$, assume that $\mathcal{R}_m'$ consists of a single task T_m'. The deadline d_m' of T_m' is t_m^{th} where $t_m^{th} \geq c_m$, its workload is infinite, its parallelism bound k_m' is C, and its marginal value $\overline{v}_m'$ is the largest one of the tasks in $\mathcal{R}_m$:

$$v_{i_m}' \geq \cdots \geq v_{j_m}' \geq \overline{v}_m' = v_{j_m+1}'. \tag{46}$$

Let $y_m'(t) \in \{0, 1, \cdots, C\}$ denote the number of machines allocated to a task T_m' at slot $t \in [d_m']^+$. Partial execution of a task of $\mathcal{A}_1, \mathcal{R}_1', \mathcal{A}_2, \cdots, \mathcal{R}_K'$ by its deadline on the C machines is allowed here and can yield linearly proportional value. In particular, for any $m \in [K]^+$, due to the characteristics of T_m' defined above, its allocation only need to satisfy:

$$0 \leq y_m'(t) \leq C \text{ for any } t \in [d_m']^+. \tag{47}$$

The value generated by the allocation of T_m' is $\sum_{t=1}^{d_i'} y_m'(t) \cdot \overline{v}_m'$.

To differentiate the allocation of this relaxed problem from the allocation of the MSW-I problem, we use $\hat{y}_i(t)$ to denote the number of machines allocated to a task $T_i \in \mathcal{A}$ at a slot t. Like Eq. (5), we have $\hat{y}_i(t) \in \{0, 1, \cdots, k_i\}$ for any $t \in [d_i]^+$ and $\hat{y}_i(t) = 0$ for any $t \in [d_i + 1, d]$ if $d_i < d$. However, if we decide to execute T_i, the allocation to T_i in $[1, d_i]$ only needs to satisfy the following condition:

$$0 < \sum\nolimits_{t \leq d_i} \hat{y}_i(t) \leq D_i \tag{48}$$

which is different from Eq. (6). The generated value is $\sum_{t=1}^{d_i} \hat{y}_i(t) \cdot v_i'$. Also, there is a capacity constraint:

$$\sum_{T_i \in \mathcal{A}} \hat{y}_i(t) + \sum_{m=1}^{K} y_m'(t) \leq C \text{ for any } t \in [d]^+. \tag{49}$$

We consider scheduling $\mathcal{A}$ and $\mathcal{R}_1', \cdots, \mathcal{R}_K'$ on the C machines with the objective of maximizing the social welfare, which is referred to as **the MSW-II problem**.

Lemma 4 *The optimal social welfare OPT_2 of the MSW-II problem is an upper bound of the optimal social welfare OPT_1 of the MSW-I problem, i.e., $OPT_1 \leq OPT_2$.*

Proof The proof can be found in Section 6.1. ■

Lemma 5 *An upper bound of the optimal social welfare of the MSW-II problem is as follows:*

$$OPT_2 \leq V_1 + \sum\nolimits_{m=1}^{K} V'_{2,m} \tag{50}$$

where $V'_{2,m} = (1-r) \cdot \left(t_m^{th} - t_{m-1}^{th}\right) \cdot C \cdot \overline{v}'_m$ *and* V_1 *is given in Eq. (45).*

Proof The proof can be found in Section 6.2. ■

The following lemma shows how to use the allocation of the MSW-I problem to bound $\sum_{i=1}^{m} V'_{2,i}$.

Lemma 6 *For any* $m \in [K]^+$*, we have*

$$\sum\nolimits_{i=1}^{m} V'_{2,i} \leq \frac{1-r}{r} \cdot \sum\nolimits_{T_i \in \cup_{j=1}^{m} \mathcal{A}_j} \sum\nolimits_{t=1}^{t_m^{th}} y_i(t) \cdot v'_i.$$

Proof The proof can be found in Section 6.3. ■

By Eq. (44) and (45), we have $\sum\limits_{T_i \in \cup_{j=1}^{K} \mathcal{A}_j} \sum\limits_{t=1}^{t_K^{th}} y_i(t) \cdot v'_i \leq V_1$ where $t_K^{th} \leq d$. By Lemma 6, we have

$$V_1 + \sum\nolimits_{m=1}^{K} V'_{2,m} \leq V_1 + \frac{1-r}{r} \cdot V_1 = \frac{1}{r} \cdot V_1.$$

Finally, by Lemmas 4 and 5, we have $OPT_1 \leq OPT_2 \leq V_1 + \sum_{m=1}^{K} V'_{2,m} \leq \frac{1}{r} \cdot V_1$; thus, $V_1 \geq r \cdot OPT_1$ and Theorem 2 holds.

4.4 Optimal Algorithm Design

Initially, for all $T_i \in \mathcal{T}$, $y_i(t)$ are set to zero at all slots. Recall in Section 4.2 the generic description of the executing process of a greedy algorithm. Now, we introduce the proposed algorithm GreedyRLM, presented as Algorithm 5. It considers the tasks in the non-increasing order of their marginal values (lines 2-4, 20). In the m-th phase, when T_i is considered, GreedyRLM operates as follows:

(i) If $\sum_{t \leq d_i} \min\{\overline{W}(t), k_i\} \geq D_i$, call Allocate-A($i$) (lines 5-6); herein, $T_i \in \mathcal{A}_m$.
(ii) Otherwise, T_i is rejected. If the last task T_{i-1} was accepted, go to find the last task of $\mathcal{R}_m$ (lines 8-10); then set the threshold parameter t_m^{th} of the m-th phase (lines 11-18).

In the m-th phase, when T_i is accepted, Fully-Utilize(i) is first called in Allocate-A(i). It considers slot t one by one from the deadline d_i towards earlier ones. At each iteration t, we have

$$y_i(t) = \min\{k_i, \overline{W}(t), D_i - \sum\nolimits_{\tilde{t}=t+1}^{d} y_i(\tilde{t})\}. \tag{51}$$

The allocation $y_i(t)$ of T_i at slot t is set according to the parallelism and capacity constraints and the remaining workload to be processed after the allocation of the previous iterations. Fully-Utilize(i) does not change the allocations of the previous

Algorithm 5: GreedyRLM

```
Input  : $n$ tasks with $type_i = \{v_i, d_i, D_i, k_i\}$
Output: A feasible allocation of resource to tasks
1  initialize: $y_i(t) \leftarrow 0$ for all $T_i \in \mathcal{T}$ and $1 \leq t \leq d$, $m = 1$, $t_0^{th} = 0$;
2  without loss of generality, assume $v_1' \geq v_2' \geq \cdots \geq v_n'$;
3  $i \leftarrow 1$;
4  while $i \leq n$ do
5      if $\sum_{t \leq d_i} \min\{\overline{W}(t), k_i\} \geq D_i$ then
           // in the $m$-th phase, $T_i$ is accepted
6          Allocate-A($i$), presented as Algorithm 6;
7      else
           // $T_i$ is rejected
8          if the last task $T_{i-1}$ was accepted then
               // the allocation to $\mathcal{A}_m$ was completed where $j_m = i - 1$
9              while $\sum_{t \leq d_{i+1}} \min\{\overline{W}(t), k_{i+1}\} < D_{i+1}$ do
10                 $i \leftarrow i + 1$;
               // the last task of $\mathcal{R}_m$ is $T_{i_{m+1}-1} = T_i$
11             if $c_m \geq c_m'$ then
12                 $t_m^{\mathrm{th}} \leftarrow c_m$;
13             else
14                 let $t_{min}'$ denote the earliest slot in $[c_m + 1, c_m']$ with $\overline{W}(t_{min}') > 0$;
15                 if such $t_{min}'$ exists then
16                     set the threshold $t_m^{\mathrm{th}}$ to $t_{min}' - 1$;
17                 else
18                     set the threshold $t_m^{\mathrm{th}}$ to $c_m'$;
19             $m \leftarrow m + 1$;              // GreedyRLM will enter the next phase
20     $i \leftarrow i + 1$;              // afterwards, the first accepted task of $\mathcal{A}_m$ is $T_i$
```

Algorithm 6: Allocate-A(i)

```
1  Fully-Utilize($i$), presented as Algorithm 4;
2  if $d_i \geq t_{m-1}^{th} + 2$ then
3      AllocateRLM($i$, $t_{m-1}^{th} + 2$), presented as Algorithm 7;
```

tasks. Due to the condition in line 5, we have $\sum_{\bar{t}=1}^{d_i} y_i(\bar{t}) = D_i$ upon completion of Fully-Utilize($\cdot$).

Second, if $d_i \geq t_{m-1}^{th} + 2$, AllocateRLM($i$, $t_{m-1}^{th} + 2$) is called after Fully-Utilize(i). It operates at each slot t from d_i to $t_{m-1}^{th} + 2$ until the total allocation of T_i in $[1, t-1]$ equals zero (lines 1, 2, 11). At each iteration t, Routine($\varDelta$, t) is called to increase the number $\overline{W}(t)$ of available machines at t to $\varDelta$ (line 5); this is realized by transferring the allocations of previously allocated tasks at t to an earlier slot t' with $\overline{W}(t') > 0$. Then, let $\theta = \overline{W}(t)$ and the $\overline{W}(t)$ idle machines at t will be additionally allocated to T_i (line 7); afterwards $\overline{W}(t) = 0$. Correspondingly, the allocations of T_i at the earliest slots will be also reduced by θ (line 8).

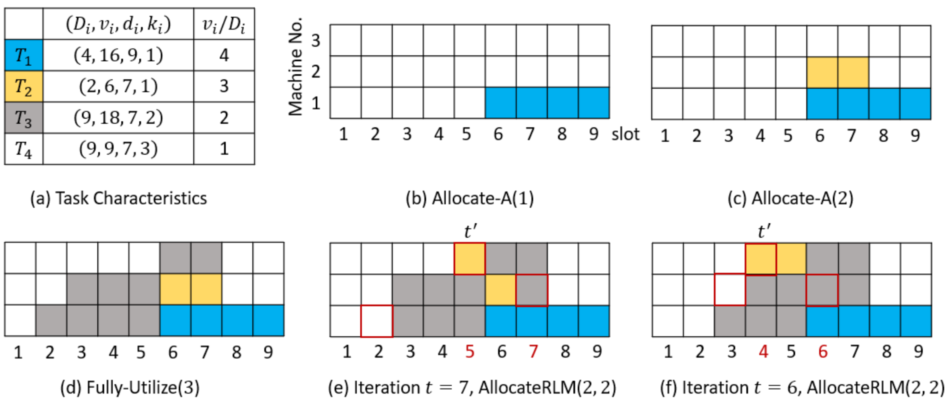
Example. Now, we illustrate the operation of GreedyRLM. Suppose there are $C = 3$ machines and a set $\mathcal{T}$ of 4 tasks $\{T_1, T_2, T_3, T_4\}$. The task characteristics are illustrated in Figure 4(a). Initially, $t_0^{\mathrm{th}} = 0$. In the first phase, first, T_1 is

Algorithm 7: AllocateRLM(i, x)

```
$t \leftarrow d_i$;
while $t \geq x$ and $\sum_{\tilde{t}=1}^{t-1} y_i(\tilde{t}) > 0$ do
    $\Delta \leftarrow \min\left\{k_i - y_i(t), \sum_{\tilde{t}=1}^{t-1} y_i(\tilde{t})\right\}$;
    if $\Delta > 0$ then
        $flag \leftarrow 0$, and call Routine($\Delta$, $t$);
        if $\overline{W}(t) > 0$ then
            $\theta \leftarrow \overline{W}(t)$, and $y_i(t) \leftarrow y_i(t) + \overline{W}(t)$;
            let $t''$ be such a slot that $\sum_{\tilde{t}=1}^{t''-1} y_i(\tilde{t}) < \theta$ and $\sum_{\tilde{t}=1}^{t''} y_i(\tilde{t}) \geq \theta$, and
             execute the following operations:
             1. $\theta \leftarrow \theta - \sum_{\tilde{t}=1}^{t''-1} y_i(\tilde{t})$,
             2. for every $\tilde{t} \in [1, t''-1]$, $y_i(\tilde{t}) \leftarrow 0$,
             3. $y_i(t'') \leftarrow y_i(t'') - \theta$;
        if $flag = 1$ then
            exit AllocateRLM($i$, $x$);
    $t \leftarrow t - 1$;
```

Algorithm 8: Routine(Δ, t)

```
while $\overline{W}(t) < \Delta$ do
    $t' \leftarrow$ the current time slot earlier than and closest to $t$ so that $\overline{W}(t') > 0$;
    if $t' \leq t_{m-1}^{th}$, or there exists no such $t'$ then
        $flag \leftarrow 1$, break;
    if $\sum_{\tilde{t}=1}^{t'-1} y_i(\tilde{t}) \leq \overline{W}(t)$ then
        $flag \leftarrow 1$, break;
    let $i'$ be a task such that $y_{i'}(t) > y_{i'}(t')$;
    $y_{i'}(t) \leftarrow y_{i'}(t) - 1$, $y_{i'}(t') \leftarrow y_{i'}(t') + 1$;
```



	(D_i, v_i, d_i, k_i)	v_i/D_i
🟦 T_1	$(4, 16, 9, 1)$	4
🟨 T_2	$(2, 6, 7, 1)$	3
▨ T_3	$(9, 18, 7, 2)$	2
T_4	$(9, 9, 7, 3)$	1

Fig. 4: An Example of GreedyRLM.

accepted and Fully-Utilize($i = 1$) is called: $y_1(t) = k_1 = 1$ for all $t \in [6, 9]$, which is illustrated in Figure 4(b). Second, T_2 is accepted and Fully-Utilize($i = 2$) is called: $y_2(t) = k_2 = 1$ for all $t \in [6, 7]$, which is illustrated in Figure 4(c). For T_1 and T_2, AllocateRLM(i, 2) is called but has no effect on the previous allocation of Fully-Utilize(i) since $\Delta = 0$ in AllocateRLM(i, 2).

Third, T_3 is accepted. When Fully-Utilize($i = 3$) is called, the allocation state is illustrated in Figure 4(d). Next, AllocateRLM($i = 3$, 2) is called. At iteration $t = d_i = 7$, $\Delta = 1$ (line 3) and then Routine(Δ, $t = 7$) is called (line 5): $t' = 5$; here, we have $i' = 2$; the allocation of T_2 at slot t is reduced by 1 and its allocation at t' is increased by 1. Afterwards, $\overline{W}(t) = 1$. Returning to AllocateRLM(3, 2), the allocation of T_3 at t is increased by 1 to $k_3 = 2$, and its allocation at the earliest slot is reduced by 1 to maintain $\sum_{\bar{t}=1}^{d_3} y_3(\bar{t}) = D_3$. Upon completion of the iteration $t = 7$ of AllocateRLM(3, 2), the allocation state is illustrated in Figure 4(e). At iteration $t = 6$, $\Delta = 1$ (line 3) and then Routine(Δ, $t = 6$) is called (line 5): $t' = 4$; the allocation of T_2 is reduced by one at t and increased by one at t'. The allocation of T_3 is increased by one at t, and its allocation at the earliest slot is reduced by one. Upon completion of the iteration $t = 6$ of AllocateRLM(3, 2), the allocation state is illustrated in Figure 4(f). Fourth, T_4 is rejected. Upon completion of GreedyRLM, we have $K = 1$, $\mathcal{A}_1 = \{T_1, T_2, T_3\}$ and $\mathcal{R}_1 = \{T_4\}$; since $c_1' > c_1$, we have $t'_{min} = 8$ and $t_1^{\text{th}} = 7$.

On the whole, upon finishing each call to Allocate-A($\cdot$), we have that

– the amount $W(t)$ of allocated machines at each slot $t \in [1, d]$ does not decrease,

in contrast to the amount just before executing Allocate-A($\cdot$). On the other hand, we have for all $j \in [m, K]$ that $\overline{W}\left(t_j^{\text{th}} + 1\right) > 0$ upon completion of the j-th phase (i.e., the allocation to $\mathcal{A}_j$) by the definition of t_j^{th} (see lines 11-18 of GreedyRLM). Thus, we also have for every task $T_i \in \bigcup_{l=1}^{m} \mathcal{A}_l$ that

$$\overline{W}\left(t_j^{\text{th}} + 1\right) > 0 \tag{52}$$

before executing and after completing Allocate-A(i). In AllocateRLM($\cdot$), the allocation of previous tasks needs to be transferred from t to an earlier slot. This requires the existence of $T_{i'}$ in line 7 of Routine($\cdot$), which is shown below.

Lemma 7 *The precondition of calling Routine(Δ, t) is $\sum_{\tilde{t}=1}^{t-1} y_i(\tilde{t}) > 0$ and $\Delta > 0$. In Routine(Δ, t), the task $T_{i'}$ in line 7 always exists if the conditions in lines 3 and 5 are false.*

Proof Since $\Delta > 0$, we have

$$y_i(t) < k_i \tag{53}$$

and $\sum_{\tilde{t}=1}^{t-1} y_i(\tilde{t}) > 0$ before calling Routine($\cdot$), which further indicates

$$\overline{W}(t) = 0 \tag{54}$$

after Fully-Utilize(i) since at its iteration t we have $y_i(t) = \min\left\{k_i, \overline{W}(t), D_i - \sum_{\bar{t}=t+1}^{d} y_i(\bar{t})\right\}$. Since the conditions in the lines 3 and 5 of Routine(Δ, t) are false, we have

$$y_i(t') = k_i \tag{55}$$

after Fully-Utilize(i) and

$$\overline{W}(t') > 0. \tag{56}$$

By Eq. (7) and Eq. (53)-(56), we have $W(t) - y_i(t) > W(t') - y_i(t')$ at the first iteration of Routine($\cdot$); then, there exists a $T_{i'}$ such that $y_{i'}(t') < y_{i'}(t)$. In the subsequent iteration of Routine($\cdot$), $\overline{W}(t)$ becomes > 0 since partial allocation of $T_{i'}$ is transferred from t to t'; however, it still holds that $\overline{W}(t) < \varDelta \leq k_i - y_i(t)$. So, we have

$$\begin{aligned} W(t) - y_i(t) &= C - \overline{W}(t) - y_i(t) \\ &> W(t') - k_i = W(t') - y_i(t') \end{aligned}$$

and such $T_{i'}$ can still be found like the first iteration. ■

Lemma 8 *For any $j \in [m, K]$, upon completion of Allocate-A(i), we have either $y_i(t) = k_i$ for all $t \in [t_j^{th} + 1, d_i]$ or $\sum_{\bar{t}=t_j^{th}+1}^{d_i} y_i(t) = D_i$, if $d_i \geq t_j^{th} + 1$.*

Proof We prove by contradiction. Suppose upon completion of Allocate-A(i) that (i) $\sum_{\bar{t}=t_j^{\mathrm{th}}+1}^{d_i} y_i(\bar{t}) < k_i \cdot (d_i - t_j^{\mathrm{th}})$ if $len_i > d_i - t_j^{th}$, and (ii) $\sum_{\bar{t}=t_j^{\mathrm{th}}+1}^{d_i} y_i(\bar{t}) < D_i$ if $len_i \leq d_i - t_j^{th}$; herein, $\sum_{\bar{t}=1}^{d_i} y_i(\bar{t}) = D_i$. In both cases, we have that

$$\sum\nolimits_{\bar{t}=1}^{t_j^{th}} y_i(\bar{t}) > 0. \tag{57}$$

and there exists a slot $\hat{t} \in [t_j^{th} + 1, d_i]$ such that

$$y_i(\hat{t}) < k_i. \tag{58}$$

Eq. (57) also holds at the iteration $\hat{t}$ of AllocateRLM(i, $t_{m-1}^{\mathrm{th}} + 2$). However, it is impossible that the iteration $\hat{t}$ would end with the states in Eq. (57) and (58): herein, $\varDelta > 0$, the slot t' in line 2 of Algorithm 8 exists where $t' \geq t_j^{\mathrm{th}} + 1 > t_{m-1}^{\mathrm{th}}$, and the condition in line 5 is false since Eq. (57) holds. ■

Proposition 4 *GreedyRLM gives an $\frac{s-1}{s}$-approximation to the optimal social welfare with a time complexity of $\mathcal{O}(knd^2 \max\{d, n\})$.*

In the rest, we will prove Proposition 4.

Lemma 9 *The time complexity of GreedyRLM is $\mathcal{O}(knd^2 \max\{d, n\})$.*

Proof GreedyRLM considers the n tasks one by one. For the task T_i, the time complexity comes from Allocate-A(i), which depends on AllocateRLM($\cdot$). In AllocateRLM($\cdot$), each loop iteration at $t \in [1, d_i]$ needs to seek the time slot t' and the task $T_{i'}$ at most D_i times. The time complexities of respectively seeking t' and $T_{i'}$ are $\mathcal{O}(d)$ and $\mathcal{O}(n)$; the maximum of these two complexities is $\max\{d, n\}$. Since $d_i \leq d$ and $D_i \leq D$, we have that the time complexity of Allocate-A(i) is $\mathcal{O}(dD \max\{d, n\})$. Since $D \leq dk$, we have that $\mathcal{O}(dD \max\{d, n\}) = \mathcal{O}(kd^2 \max\{d, n\})$. Finally, the time complexity of GreedyRLM is $\mathcal{O}(knd^2 \max\{d, n\})$. ■

Due to Theorem 2, in the following, we only need to prove that Features 1 and 2 holds in GreedyRLM where $r = \frac{s-1}{s}$, which is given in Proposition 5 and 6. The worst-case utilization of GreedyRLM is derived mainly by analyzing the resource allocation state when a task T_i in the first m phases is rejected (the condition in line 5 of GreedyRLM is not satisfied), and we have that

Proposition 5 *Upon completion of GreedyRLM, Feature 1 holds in which* $r = \frac{s-1}{s}$.

Proof The omitted proof can be found in Section 6. ■

Proposition 6 *For any* $j \in [m, K]$*, upon completion of GreedyRLM, we have for any* $T_i \in \bigcup_{l=1}^{m} \mathcal{A}_l$ *that either* $y_i(t) = k_i$ *for all* $t \in [t_j^{th}+1, d_i]$ *or* $\sum_{t=t_j^{th}+1}^{d_i} y_i(t) = D_i$*, if* $d_i \geq t_j^{th}+1$.

Proof By Lemma 8, for any $m' \in [m]^+$ and $T_i \in \mathcal{A}_{m'}$, we have either $y_i(t) = k_i$ for all $t \in [t_{m'}^{th}+1, d_i]$ or $\sum_{t=t_{m'}^{th}+1}^{d_i} y_i(t) = D_i$ upon completion of Allocate-A(i). For any $l \in [m', K]$, we observe the subsequent execution of Allocate-A($\cdot$) after T_i whose input is a task in $\mathcal{A}_l$ and could conclude that,

(a) upon completion of Allocate-A($\cdot$), the allocations to T_i in $[1, t_{l-1}^{\text{th}}]$ are still the ones before executing it;
(b) Allocate-A($\cdot$) can only change the allocations of T_i in the range $[t_{l'}^{\text{th}}+1, t_{l'+1}^{\text{th}}]$ where $l' \in [l-1, K]$ but does not change the total amount of the allocations in $[t_{l'}^{\text{th}}+1, t_{l'+1}^{\text{th}}]$.

As a result, every subsequent execution of Allocate-A($\cdot$) never changes the total amount of allocations of T_i in $[t_j^{\text{th}}+1, t_{j+1}^{\text{th}}]$ for all $j \in [m, K]$. Further, upon completion of GreedyRLM, the total amount of the allocations to T_i in $[t_j^{\text{th}}+1, d]$ equals its counterpart upon completion of Allocate-A(i).

In the following, we prove the two points (a) and (b) above. In the execution of Allocate-A($\cdot$), Fully-Utilize($\cdot$) is first called and it does not change the allocation to the previous tasks; then, AllocateRLM($\cdot$, $t_l^{\text{th}}+2$) is called in which only Routine($\cdot$) (i.e., its lines 7-8) can change the allocation to the previous tasks including T_i. In lines 7-8, a previous task $T_{i'}$ is found to change its allocations at t and t'; here, t' is defined in line 2 and $t_{l-1}^{\text{th}} < t' < t$. As a result, Allocate-A($\cdot$) cannot change the allocations of the previous tasks in $[1, t_{l-1}^{\text{th}}]$; for all $t \in [t_{l'}^{\text{th}}+1, t_{l'+1}^{\text{th}}]$ where $l' \in [l, K]$, during the execution of the iteration of AllocateRLM($\cdot$) at t, we have $t' \geq t_{l'}^{\text{th}}+1$. Hence, the change to the allocations of the previous tasks can only happen in the interval $[t_{l'}^{\text{th}}+1, t_{l'+1}^{\text{th}}]$. ■

5 Applications: Part II

In this section, we illustrate the applications of the results in Section 3 to (i) the dynamic programming technique for social welfare maximization, (ii) the machine minimization objective, and (iii) the objective of minimizing the maximum weighted completion time.

Algorithm 9: DP($\mathcal{T}$)

```
1  $\mathcal{F} \leftarrow \{T_1\}$;
2  $A(1) \leftarrow \{(\emptyset, 0), (\mathcal{F}, v(\mathcal{F}))\}$;
3  for $i \leftarrow 2$ to $n$ do
4      $A(j) \leftarrow A(i-1)$;
5      for each $(\mathcal{F}, v(\mathcal{F})) \in A(i-1)$ do
6          if $\{T_i\} \cup \mathcal{F}$ satisfies the boundary condition then
7              if there exist a pair $(\mathcal{F}', v(\mathcal{F}')) \in A(i)$ so that (1) $H(\mathcal{F}') = H(\mathcal{F} \cup \{T_i\})$,
               and (2) $v(\mathcal{F}') \geq v(\mathcal{F} \cup \{T_i\})$ then
8                  Add $(\{T_i\} \cup \mathcal{F}, v(\{T_i\} \cup \mathcal{F}))$ to $A(i)$;
9                  Remove the dominated pair $(\mathcal{F}', v(\mathcal{F}'))$ from $A(i)$;
10             else
11                 Add $(\{T_i\} \cup \mathcal{F}, v(\{T_i\} \cup \mathcal{F}))$ to $A(i)$;
12 return $\arg\max_{(\mathcal{F}, v(\mathcal{F})) \in A(n)} \{v(\mathcal{F})\}$;
```

5.1 Dynamic Programming

For the problem with social welfare, a subset of tasks is chosen to maximize their total value, with the constraint that these tasks need to be completed by their deadlines. For any solution, there must exist a feasible schedule for the chosen tasks, and the set of chosen tasks in an optimal solution also needs to satisfy the boundary condition by Theorem 1. Then, to find the optimal solution, we only need address the following problem: if we are given C machines, how can we choose a subset $\mathcal{S}$ of tasks in $\mathcal{T}$ such that (i) this subset satisfies the boundary condition, and (ii) no other subset of selected tasks achieves a better social welfare? This problem can be solved via the generic dynamic programming (DP) procedure.

To enable a DP algorithm, we need to identify a dominant condition for the model of this paper [20]. Let $\mathcal{F} \subseteq \mathcal{T}$ and recall the notation $\lambda_t^C(\mathcal{F})$ in Eq. (10). Now, we define a d-dimensional vector

$$H(\mathcal{F}) = \left(\lambda_1^C(\mathcal{F}) - \lambda_2^C(\mathcal{F}), \cdots, \lambda_d^C(\mathcal{F}) - \lambda_{d+1}^C(\mathcal{F})\right),$$

where we have for all $t \in [d]^+$ that $\lambda_t^C(\mathcal{F}) - \lambda_{t+1}^C(\mathcal{F})$ denotes the maximum resource that $\mathcal{F}$ can utilize on the C machines in the slot t after $\mathcal{F}$ has maximally utilized $\lambda_{t+1}^C(\mathcal{F})$ resource in $[t+1, d]$. Let $v(\mathcal{F})$ denote the total value of the tasks in $\mathcal{F}$ and then we introduce the notion of one pair $(\mathcal{F}, v(\mathcal{F}))$ *dominating* another $(\mathcal{F}', v(\mathcal{F}'))$ if $H(\mathcal{F}) = H(\mathcal{F}')$ and $v(\mathcal{F}) \geq v(\mathcal{F}')$, that is, the solution to our problem indicated by $(\mathcal{F}, v(\mathcal{F}))$ uses the same amount of resource as $(\mathcal{F}', v(\mathcal{F}'))$, but obtains at least the same value.

We now give the general DP procedure DP($\mathcal{T}$), also presented as Algorithm 9 [20]. Here, we iteratively construct the lists $A(i)$ for all $i \in [n]^+$. Each $A(i)$ is a list of pairs $(\mathcal{F}, v(\mathcal{F}))$, in which $\mathcal{F}$ is a subset of $\{T_1, T_2, \cdots, T_i\}$ satisfying the boundary condition. Each list only maintains all the dominant pairs. Specifically, we start with $A(1) = \{(\emptyset, 0), (\{T_1\}, v_1)\}$. For each $i = 2, \cdots, n$, we first set $A(i) \leftarrow A(i-1)$, and for each $(\mathcal{F}, v(\mathcal{F})) \in A(i-1)$, we add $(\mathcal{F} \cup \{T_i\}, v(\mathcal{F} \cup \{T_i\}))$ to the list $A(i)$ if $\mathcal{F} \cup \{T_i\}$ satisfies the boundary condition. We finally remove from $A(i)$ all the dominated pairs. DP($\mathcal{T}$) will select a subset $\mathcal{S}$ of $\mathcal{T}$ from all pairs $(\mathcal{F}, v(\mathcal{F})) \in A(n)$ so that $v(\mathcal{F})$ is the maximum.

Algorithm 10: Machine Minimization

```
1  $L = 1$, $U = k \cdot n$;
2  Compute the $\lambda_t(\mathcal{T})$ in Eq. (9) for all $t \in [d]^+$;
3  while $U - L > 1$ do
4      $M = \lfloor (U + L)/2 \rfloor$;
5      Compute the $\lambda_t^M(\mathcal{T})$ in Eq. (10) for all $t \in [d]^+$;
6      Compute the $\mu_t^M(\mathcal{T})$ in Eq. (18) for all $t \in [d]$;
7      if the boundary condition in Eq. (19) holds then
8          $U = M$;
9      else
10         $L = M$;
```

Proposition 7 *DP($\mathcal{T}$) outputs a subset $\mathcal{S}$ of $\mathcal{T} = \{T_1, \cdots, T_n\}$ such that $v(\mathcal{S})$ is the maximum value subject to the condition that $\mathcal{S}$ satisfies the boundary condition; the time complexity of DP($\mathcal{T}$) is $\mathcal{O}(nC^d)$.*

Proof The proof is similar to the one in the knapsack problem [20]. By induction, we need to prove that $A(i)$ contains all the non-dominated pairs corresponding to feasible sets $\mathcal{F} \in \{T_1, \cdots, T_i\}$. When $i = 1$, the proposition holds obviously. Now suppose it hold for $A(i-1)$. Let $\mathcal{F}' \subseteq \{T_1, \cdots, T_i\}$ and $H(\mathcal{F}')$ satisfies the boundary condition. We claim that there is some pair $(\mathcal{F}, v(\mathcal{F})) \in A(i)$ such that $H(\mathcal{F}) = H(\mathcal{F}')$ and $v(\mathcal{F}) \geq v(\mathcal{F}')$. First, suppose that $T_i \notin \mathcal{F}'$. Then, the claim follows by the induction hypothesis and by the fact that we initially set $A(i)$ to $A(i-1)$ and removed dominated pairs. Now suppose that $T_i \in \mathcal{F}'$ and let $\mathcal{F}_1' = \mathcal{F}' - \{T_i\}$. By the induction hypothesis there is some $(\mathcal{F}_1, v(\mathcal{F}_1)) \in A(i-1)$ that dominates $(\mathcal{F}_1', v(\mathcal{F}_1'))$. Then, the algorithm will add the pair $(\mathcal{F}_1 \cup \{T_i\}, v(\mathcal{F}_1 \cup \{T_i\}))$ to $A(i)$. Thus, there will be some pair $(\mathcal{F}, v(\mathcal{F})) \in A(i)$ that dominates $(\mathcal{F}', v(\mathcal{F}'))$. Since the space size of $H(\mathcal{F})$ is no more than C^d, the time complexity of DP($\mathcal{T}$) is $\mathcal{O}(nC^d)$. ■

Proposition 8 *Given the subset of tasks $\mathcal{S}$ chosen by DP($\mathcal{T}$), Sched($\mathcal{S}$) gives a feasible schedule of $\mathcal{S}$, which is an optimal solution to the welfare maximization problem with a time complexity $\mathcal{O}(\max\{nC^d, n\log n\})$.*

Proof It follows from Proposition 7 and Theorem 1. ■

Remark. As in the knapsack problem [20], to construct the algorithm DP($\mathcal{T}$), the pairs of the possible state of resource utilization and the corresponding best social welfare have to be maintained and a d-dimensional vector has to be defined to indicate the resource utilization state. This seems to imply that we cannot make the time complexity of a DP algorithm polynomial in L.

5.2 Machine Minimization

In this subsection, we aim to finish all tasks of $\mathcal{T}$ by their given deadlines. The minimal number of machines needed to produce a feasible allocation of $\mathcal{T}$ is exactly the minimum C^* such that the boundary condition is satisfied. An upper bound of the minimum C^* is $k \cdot n$ and C^* can be obtained through a binary search procedure, which is formally presented in Algorithm 9.

Algorithm 11: Minimizing Maximum Completion Time

```
1 $max\_c = n \cdot \max_{T_i \in \mathcal{T}}\{len_i\}$;
2 $U = max\_c \cdot \max_{T_i \in \mathcal{T}}\{v_i\}$, $L = \max_{T_i \in \mathcal{T}}\{v_i\}$;
3 while $U > (1+\epsilon) \cdot L$ do
4     $M = (U+L)/2$;
5     For every task $T_i \in \mathcal{T}$, let $d_i = \lfloor M/v_i \rfloor$;
6     if Algorithm 2 produces a feasible schedule of $\mathcal{T}$ by Theorem 1 then
7         $U = M$;
8     else
9         $L = M$;
```

Proposition 9 *There exists an exact algorithm for the machine minimization objective with a time complexity of $\mathcal{O}(nd\log(kn))$.*

Proof When Algorithm 9 ends, we have that (a) $U = L + 1$ and (b) if we consider scheduling $\mathcal{T}$ on U machines, the resulting boundary condition can be satisfied; however, if we consider scheduling $\mathcal{T}$ on L machines, the boundary condition does not hold. By Theorem 1, the final U is the optimal C^*. The binary search procedure itself in lines 2-9 has a time complexity of $\mathcal{O}(\log(kn))$. The operations of computing $\{\lambda_t(\mathcal{T})\}_{t=1}^{d}$ and $\left\{\lambda_t^M(\mathcal{T})\right\}_{t=1}^{d}$ in lines 2 and 5 have a time complexity of $\mathcal{O}(nd)$. Afterwards, the operations of computing $\left\{\mu_t^M(\mathcal{T})\right\}_{t=0}^{d}$ in line 6 have a time complexity of $\mathcal{O}(n+d)$. Thus, the total time complexity of Algorithm 9 is $\mathcal{O}(nd\log(kn))$. ■

5.3 Maximum Weighted Completion Time Minimization

In this subsection, we are given the parameters D_i, k_i and v_i of each task $T_i \in \mathcal{T}$; here, with abuse of notation, v_i represents the weight of T_i. We will finish all the tasks of $\mathcal{T}$ on the C machines. The weighted completion time of a task T_i is defined below:

$$w_i(\varepsilon_i) = v_i \cdot \varepsilon_i. \tag{59}$$

where ε_i is a positive integer and the time slot at which T_i is finished, i.e., $\varepsilon_i = \max\{t : y_i(t) > 0\}$. For each task $T_i \in \mathcal{T}$, we have $\sum_{t=1}^{\varepsilon_i} y_i(t) = D_i$. Our objective is to minimize the maximum weighted completion time of all tasks of $\mathcal{T}$, i.e.,

$$\text{minimize} \ \ \max_{T_i \in \mathcal{T}}\{w_i(\varepsilon_i)\}. \tag{60}$$

Previously, Nagarajan et al. observed that the problem of minimizing the maximum weighted completion time can be reduced to a feasibility problem by introducing deadlines and applying a simple binary search [8]. This idea can be adapted to solve the problem of this subsection, as we will see in Algorithm 11. Specifically, suppose we have a schedule of $\mathcal{T}$ whose maximum weighted completion time is x where x is a positive real number and $w_i(\varepsilon_i) \le x$ for all $T_i \in \mathcal{T}$. Then, by Eq. (59), we have an upper bound of the completion time ε_i of each task $T_i \in \mathcal{T}$, denoted as $d_i(x)$:

$$d_i(x) = \lfloor x/v_i \rfloor. \tag{61}$$

Let $\mathcal{D}(x) = \{d_i \,|\, T_i \in \mathcal{T}, d_i = d_i(x)\}$. We apply the binary search procedure to the core results in Section 3, and the resulting algorithm is presented in Algorithm 11. For any task $T_i \in \mathcal{T}$, an upper bound of its completion time is $max\_c = n \cdot \max_{T_i \in \mathcal{T}}\{len_i\}$, and a lower bound is 1. By Eq. (59) and (60), we can thus set the corresponding upper bound U and lower bound L of the optimal for our problem (lines 1-2). Algorithm 11 operates until $U \leq (1+\epsilon)\cdot L$ (line 3):

(i) Let $M = (U+L)/2$. We set the deadline of $T_i \in \mathcal{T}$ to $d_i(M)$ such that when every task T_i is finished by slot $d_i(M)$, the maximum weighted completion time of all tasks is no larger than M (lines 4-5).
(ii) If Algorithm 2 can produce a feasible schedule under the deadline setting $\mathcal{D}(M)$, we set $U = M$ (lines 6-7); otherwise, we set $L = M$ (lines 8-9).

Lemma 10 *For any positive real number x, if there exists a schedule of $\mathcal{T}$, denoted by $Sched'$, whose maximum weighted completion time is x, then we can always obtain a feasible schedule of $\mathcal{T}$ under the deadline setting $\mathcal{D}(x)$ whose maximum weighted completion time is no larger than x.*

Proof In $Sched'$, we denote the completion time slot of $T_i \in \mathcal{T}$ by ε_i'. By Theorem 1, we can give a feasible schedule of $\mathcal{T}$ in which each task T_i is finished by the slot ε_i'. Since $\max_{T_i\in\mathcal{T}}\{\varepsilon_i' \cdot v_i\} \leq x$, we have $\varepsilon_i' \leq d_i(x)$ and $d_i(x)\cdot v_i \leq x$ by Eq. (61). The lemma thus holds. ■

Proposition 10 *There is a $(1+\epsilon)$-approximation algorithm with a time complexity of $\mathcal{O}(nkd\log(kn)\log(n/\epsilon))$ for scheduling independent malleable tasks under the objective of minimizing the maximum weighted completion time of tasks.*

Proof When Algorithm 11 ends, we can get a feasible schedule of $\mathcal{T}$ under the deadline setting $\mathcal{D}(U)$ and cannot achieve this under the deadline setting $\mathcal{D}(L)$. Here, U is an upper bound of the optimal maximum weighted completion time achievable. By Lemma 10, we have that there does not exist a schedule whose maximum weighted completion time is L and L is thus a lower bound of the optimal maximum weighted completion time. Since $U \leq (1+\epsilon)\cdot L$, the schedule of $\mathcal{T}$ under the deadline setting $\mathcal{D}(U)$ gives a $(1+\epsilon)$-approximation algorithm. The binary search procedure itself in lines 3-9 has a time complexity of $\mathcal{O}(\log n/\epsilon)$. By Theorem 1, the test in line 6 has a time complexity of $\mathcal{O}(nkd\log(kn))$. Thus, the total complexity of Algorithm 11 is $\mathcal{O}(nkd\log(kn)\log(n/\epsilon))$. ■

6 Related Proofs

6.1 Proof of Lemma 4

Let us consider an optimal allocation to $\mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_K, \mathcal{R}_K$ for the MSW-I problem. If we replace an allocation to a task in $\mathcal{R}_m$ with the same allocation to a task in $\mathcal{R}_m'$ and do not change the allocation to $\mathcal{A}_m$, this generates a feasible allocation for the MSW-II problem, which yields at least the same social welfare by Eq. (46); hence, Lemma 4 holds.

6.2 Proof of Lemma 5

Lemma 11 *For any $m \in [K]^+$, while the allocation of $\cup_{j=1}^{m}\mathcal{A}_j$ of the MSW-I problem satisfies Features 1 and 2, we have:*

$$\mu_{t_m^{th}}^{C}\left(\cup_{j=1}^{m}\mathcal{A}_j\right) \geq r\cdot C\cdot t_m^{th}. \tag{62}$$

Proof Due to Feature 2 and Definition 1, we have

$$\begin{aligned}&\sum\nolimits_{T_i\in\cup_{j=1}^{m}\mathcal{A}_j}\sum\nolimits_{\bar{t}=t_m^{\text{th}}+1}^{d} y_i(\bar{t})\\ =&\sum\nolimits_{T_i\in\cup_{j=1}^{m}\mathcal{A}_j}\min\left\{D_i, k_i\cdot(d_i-t_m^{\text{th}})^+\right\}\\ =&\lambda_{t_m^{\text{th}}+1}\left(\cup_{j=1}^{m}\mathcal{A}_j\right)\end{aligned}$$

By Eq. (11) and (13), we have

$$\lambda_{t_m^{\text{th}}+1}^{C}\left(\cup_{j=1}^{m}\mathcal{A}_j\right)=\sum\nolimits_{T_i\in\cup_{j=1}^{m}\mathcal{A}_j}\sum\nolimits_{\bar{t}=t_m^{\text{th}}+1}^{d} y_i(\bar{t}).$$

By Eq. (18), we have

$$\mu_{t_m^{\text{th}}}^{C}\left(\cup_{j=1}^{m}\mathcal{A}_j\right)=\sum\nolimits_{T_i\in\cup_{j=1}^{m}\mathcal{A}_j}\sum\nolimits_{\bar{t}=1}^{t_m^{\text{th}}} y_i(\bar{t})$$

Finally, by Feature 1, the lemma holds. ■

Lemma 12 *Suppose there are K variables $x_1, x_2, \cdots, x_K$ that satisfy:*

$$\sum\nolimits_{j=1}^{m} x_j \leq (1-r)\cdot t_m^{th}\cdot C \textit{ for any } m\in[K]^+. \tag{63}$$

Then, $\sum_{m=1}^{K} x_m\cdot\overline{v}'_m$ achieves the maximum value when $x_m^* = (1-r)\cdot(t_m^{th}-t_{m-1}^{th})\cdot C$ for all $m\in[K]^+$, where

$$\overline{v}'_1 \geq \overline{v}'_2 \geq \cdots \geq \overline{v}'_K.$$

Proof When the maximum value is achieved, we have

$$\sum\nolimits_{j=1}^{K} x_j = (1-r)\cdot t_K^{\text{th}}\cdot C; \tag{64}$$

otherwise, for the case where $m = K$, we can increase the value of x_K until the equality in Eq. (63) is achieved to maximize $\sum_{m=1}^{K} x_m\cdot\overline{v}'_m$. In the following, we only consider the solutions that satisfy Eq. (64) and these solutions only have two cases: (i) $x_m = (1-r)\cdot(t_m^{th}-t_{m-1}^{th})\cdot C$ for all $m\in[K]^+$ and (ii) there exists a $m'\in[K]^+$ such that $x_{m'} < (1-r)\cdot(t_{m'}^{th}-t_{m'-1}^{th})\cdot C$. We will prove that the value of $\sum_{m=1}^{K} x_m\cdot\overline{v}'_m$ under the solution in the first case is always no smaller than its counterpart under a solution in the second case. Then, Lemma 12 will hold.

Without loss of generality, let m' denote the smallest such m' where

$$x' = (1-r)\cdot(t_{m'}^{th}-t_{m'-1}^{th})\cdot C - x_{m'} > 0. \tag{65}$$

Then, if $m' \geq 2$, we also have

$$x_m = (1-r)\cdot(t_m^{th} - t_{m-1}^{th})\cdot C \text{ for } m \in \left[m'-1\right]^+ . \tag{66}$$

By Eq. (64)-(66), we have $m' < K$ and

$$\sum\nolimits_{j=m'+1}^{K} x_j = (1-r)\cdot(t_K^{th} - t_{m'}^{th})\cdot C + x'.$$

If the solution $x_1, x_2, \cdots, x_K$ satisfies Eq. (64)-(66), then the value of $\sum_{m=1}^{K} x_m \cdot \overline{v}'_m$ is at least the same after we increase the value of $x_{m'}$ in Eq. (65) by x' to $(1-r)\cdot(t_{m'}^{th} - t_{m'-1}^{th})\cdot C$ and simultaneously reducing the value of $\sum_{j=m'+1}^{K} x_j$ by x'. ■

In the following, we use Lemmas 11 and 12 to prove Lemma 5. All tasks are divided into $\mathcal{A}_1, \mathcal{R}'_1, \mathcal{A}_2, \cdots, \mathcal{R}'_K$. For any $m \in [K]^+$, we first show which tasks are not needed to be executed in $[t_{m-1}^{th}+1, t_m^{th}]$ in order to get an optimal allocation that can achieve the maximum value: if $m \geq 2$, the tasks of $\mathcal{R}'_1, \cdots, \mathcal{R}'_{m-1}$ will not be executed in $[t_{m-1}^{th}+1, t_m^{th}]$; if $m \leq K-1$, the tasks of $\mathcal{R}'_{m+1}, \cdots, \mathcal{R}'_K$ will not be executed in $[t_{m-1}^{th}+1, t_m^{th}]$. This says that, there exists an optimal allocation/schedule that only needs to allocate machines to the tasks of $\mathcal{A}_1, \cdots, \mathcal{A}_K$ in $[1, d]$ and to the tasks of $\mathcal{R}'_m$ in $[t_{m-1}^{th}+1, t_m^{th}]$ for any $m \in [K]^+$. In the rest of this proof, we focus on such an optimal allocation.

The proof of the above conclusion is as follows. Given a $m \in [K]^+$, if $m \geq 2$, all tasks of $\mathcal{R}'_1, \cdots, \mathcal{R}'_{m-1}$ could not be processed in $[t_{m-1}^{th}+1, t_m^{th}]$ due to the deadline constraint. If $m \leq K-1$, the marginal values of tasks satisfy $\overline{v}'_m \geq \overline{v}'_{m+1} \geq \cdots \geq \overline{v}'_K$ by Eq. (46); instead of processing $\mathcal{R}'_{m+1}, \cdots, \mathcal{R}'_K$ in $[t_{m-1}^{th}+1, t_m^{th}]$, processing $\mathcal{R}'_m$ could generate at least the same value or even a higher value.

Suppose we are considering an optimal schedule described above. Let

$$\mathcal{Y}_A = \left\{\hat{y}_i(t) | T_i \in \bigcup\nolimits_{l=1}^{K} \mathcal{A}_l, t \in [d]^+\right\}$$

denote the allocations of $\bigcup_{l=1}^{K} \mathcal{A}_l$ in $[1, d]$. For any $m \in [K]^+$, we define several parameters below:

$$C_m = C\cdot(t_m^{th} - t_{m-1}^{th}) \tag{67}$$

$$R_m = C_m - \sum\nolimits_{T_i \in \bigcup_{l=1}^{K} \mathcal{A}_l} \sum\nolimits_{t=t_{m-1}^{\mathrm{th}}+1}^{t_m^{\mathrm{th}}} \hat{y}_i(t) \tag{68}$$

$$p_m = \sum\nolimits_{T_i \in \mathcal{A}_m} \left(D_i - \sum\nolimits_{t=1}^{d} \hat{y}_i(t)\right) \tag{69}$$

$$V_{p_m} = \sum\nolimits_{T_i \in \mathcal{A}_m} \left(D_i - \sum\nolimits_{t=1}^{d} \hat{y}_i(t)\right)\cdot v'_i \tag{70}$$

$$V_A = \sum\nolimits_{T_i \in \bigcup_{l=1}^{K} \mathcal{A}_l} \sum\nolimits_{t=1}^{d} \hat{y}_i(t)\cdot v'_i \tag{71}$$

$$R'_m = R_m - p_m \tag{72}$$

The allocation of T'_m in $\left[t_{m-1}^{th}+1, t_m^{th}\right]$ is denoted as

$$\mathcal{Y}_m = \left\{y'_m(t) | t \in \left[t_{m-1}^{th}+1, t_m^{th}\right]\right\}.$$

In an optimal solution, we have that the equality in Eq. (49) holds and thus

$$\sum\nolimits_{t=t^{\mathrm{th}}_{m-1}+1}^{t^{\mathrm{th}}_m} y'_m(t) = R_m \tag{73}$$

where R_m is given in Eq. (68). By Eq. (46), we have

$$V_{p_m} \geq p_m \cdot \overline{v}'_m. \tag{74}$$

The allocation of an optimal schedule is denoted as $\mathcal{Y}_A$ and $\bigcup_{m=1}^{K} \mathcal{Y}_m$. By Eq. (71) and (73), the corresponding optimal social welfare is as follows:

$$\begin{aligned} OPT_2 &= V_A + \sum\nolimits_{m=1}^{K} \overline{v}'_m \cdot R_m \\ &\overset{(a)}{\leq} V_A + \sum\nolimits_{m=1}^{K} V_{p_m} + \sum\nolimits_{m=1}^{K} \overline{v}'_m \cdot R'_m \\ &\overset{(b)}{\leq} V_1 + \sum\nolimits_{m=1}^{K} \overline{v}'_m \cdot R'_m \end{aligned} \tag{75}$$

where inequality (a) is due to Eq. (72) and (74), and (b) holds since $V_A + \sum_{m=1}^{K} V_{p_m} = \sum_{T_i \in \bigcup_{l=1}^{K} \mathcal{A}_l} D_i \cdot v'_i = V_1$ where V_1 is given in Eq. (45). In the rest of this proof, for any $m \in [K]^+$, we prove $\sum_{j=1}^{m} R'_j \leq (1-r) \cdot t^{\mathrm{th}}_m \cdot C$. Then, by Eq. (75) and Lemma 12, we have that Lemma 5 holds.

By Eq. (67), (68), (69) and (72), we have:

$$\begin{aligned} \sum_{j=1}^{m} R'_j &= t^{\mathrm{th}}_m \cdot C - \sum_{T_i \in \bigcup\limits_{l=1}^{K} \mathcal{A}_l} \sum_{t=1}^{t^{\mathrm{th}}_m} \hat{y}_i(t) - \sum_{j=1}^{m} p_j \\ &\leq t^{\mathrm{th}}_m \cdot C - \sum_{T_i \in \bigcup\limits_{l=1}^{m} \mathcal{A}_l} \left(D_i - \sum_{t=1}^{d} \hat{y}_i(t) \right) + \sum_{t=1}^{t^{\mathrm{th}}_m} \hat{y}_i(t) \\ &= t^{\mathrm{th}}_m \cdot C - \left(\sum_{T_i \in \bigcup\limits_{l=1}^{m} \mathcal{A}_l} D_i - \sum_{T_i \in \bigcup\limits_{j=1}^{m} \mathcal{A}_j} \sum_{t=t^{\mathrm{th}}_m+1}^{d} \hat{y}_i(t) \right) \\ &\overset{(c)}{\leq} t^{\mathrm{th}}_m \cdot C - \mu^{C}_{t^{\mathrm{th}}_m} \left(\cup_{j=1}^{m} \mathcal{A}_j \right) \overset{(d)}{\leq} (1-r) \cdot t^{\mathrm{th}}_m \cdot C \end{aligned}$$

Here, inequality (c) is due to Eq. (18) and (13); inequality (d) is due to Lemma 11.

6.3 Proof of Lemma 6

It suffices to prove that, the total allocation to $\cup_{l=1}^{m} \mathcal{A}_l$ in $[1, t_m^{th}]$ could be divided into m parts such that, for all $l \in [1, m]$, (i) the l-th part has a size $r \cdot (t_l^{th} - t_{l-1}^{th}) \cdot C$, and (ii) the allocation of the l-th part is associated with marginal values no smaller than $\overline{v}'_l$. Then, the total value generated by executing the l-th part is no smaller than $\frac{1-r}{r}$ times the total value generated by the allocation to $\mathcal{R}'_l$

in $[t_{l-1}^{th}+1, t_l^{th}]$. As a result, the value generated by the total allocation to $\cup_{l=1}^{m}\mathcal{A}_l$ in $[1, t_m^{th}]$ is no smaller than $\frac{1-r}{r}$ times the value generated by the allocation to $T_1', \cdots, T_m'$.

Due to Feature 1, the allocation to $\mathcal{A}_1$ achieves a utilization r in $[1, t_1^{th}]$ and we could use a part of this allocation as the first part whose size is $r \cdot t_1^{th} \cdot C$. Next, the allocation to $\mathcal{A}_1 \cup \mathcal{A}_2$ achieves a utilization r in $[1, t_2^{th}]$; we could deduct the allocation used for the first part and get a part of the remaining allocation to $\mathcal{A}_1 \cup \mathcal{A}_2$ as the second part, whose size is $r \cdot (t_2^{th} - t_1^{th}) \cdot C$. Similarly, we could get the 3rd, $\cdots$, m-th parts that satisfy the first point mentioned at the beginning of this proof. Since the marginal value of the task of $\mathcal{R}_l'$ is no larger than the ones of the tasks in $\cup_{l'=1}^{l}\mathcal{A}_{l'}$ for all $1 \leq l \leq m$, the second point mentioned above also holds.

6.4 Proof of Proposition 5

We consider a task $T_i \in \cup_{l=1}^{m}\mathcal{R}_l$ such that $d_i = c_m$, and suppose $T_i \in \mathcal{R}_{m'}$ for some $m' \in [m]^+$. *First*, we analyze the resource allocation state at the moment that T_i is rejected. Let μ denote the number of the slots t in $[1, c_m]$ with $\overline{W}(t) \geq k_i$. At this moment, we have $\mu \leq len_i - 1 = \lceil \frac{d_i}{s_i} \rceil - 1$ since $\sum_{t \leq d_i} \min\{k_i, \overline{W}(t)\} < D_i$. In the worst case, all machines are idle at the μ slots, and the total idle resource at the other slots in $[1, c_m]$ is $< D_i - \mu \cdot k_i$. As a result, we have the current utilization of $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_{m'}$ in $[1, c_m]$ is at least

$$\begin{aligned}
&\frac{C \cdot d_i - \mu \cdot C - (D_i - \mu \cdot k_i)}{C \cdot d_i} \\
&\geq \frac{C \cdot d_i - D_i - (len_i - 1) \cdot (C - k_i)}{C \cdot d_i} \\
&\geq \frac{C \cdot (d_i - len_i) + (C - k_i) + (len_i \cdot k_i - D_i)}{C \cdot d_i} \\
&\geq \frac{s-1}{s} = r.
\end{aligned}$$

Second, we show, after T_i is rejected, the resource allocation by Allocate-A(j) to each subsequent task $T_j \in \mathcal{A}_l$, where $l \in [m'+1, L]$, doesn't change the utilization in $[1, c_m]$; here, we have $c_m = c_{m'} \leq t_{m'}^{th} \leq \cdots \leq t_m^{th} \leq \cdots \leq t_L^{th}$. Fully-Utilize($j$) doesn't change the allocation to the previously accepted tasks. In AllocateRLM(j, $t_l^{th}+2$), the operations of changing the allocation to other tasks happen in its call to Routine(Δ, t); due to the function of lines 7-8, the call to Allocate-A(j) will never change the current allocation of $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_{m'}$ in $[1, c_m]$. Hence, if $t_m^{th} = c_m$, upon completion of GreedyRLM, the utilization of $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_m$ is at least r where $m' \leq m$; if $t_m^{th} > c_m$, since each time slot in $[c_m+1, t_m^{th}]$ is fully utilized by the definition of t_m^{th}, the utilization in $[c_m+1, t_m^{th}]$ is 1 and the final resource utilization will also be at least r.

7 Conclusion

In this paper, we study the problem of scheduling n malleable batch tasks on C identical machines. Our core result is to give the first optimal scheduling algorithm so that C machines can be optimally utilized by a set of batch tasks. We further derive four algorithmic results in obvious or non-obvious ways: (i) a greedy algorithm for social welfare maximization with a pseudo-polynomial time complexity of $\mathcal{O}(knd^2\max\{d, n\})$ that achieves an approximation ratio of $\frac{s-1}{s}$, (ii) the first dynamic programming algorithm for social welfare maximization with a pseudo-polynomial time complexity of $\mathcal{O}(\max\{nC^d, n\log n\})$, (iii) the first exact algorithm for machine minimization with a pseudo-polynomial time complexity of $\mathcal{O}(nd\log(kn))$, and (iv) a $(1+\epsilon)$-approximation algorithm with a pseudo-polynomial time complexity of $\mathcal{O}(nkd\log(kn)\log(n/\epsilon))$ for minimizing the maximum weighted completion time of all tasks. Here, d is the maximum deadline of tasks while k is the maximum parallelism bound of tasks.

Acknowledgements 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.

Compliance with Ethical Standards **Conflict of Interest** The authors declare that they have no conflict of interest.

Data Availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

References

1. G. Brassard, and P. Bratley. Fundamentals of Algorithmics. *Prentice-Hall, Inc.*, 1996.
2. X. Wu, P. Loiseau, and E. Hyytiä. "Towards Designing Cost-Optimal Policies to Utilize IaaS Clouds with Online Learning." *IEEE Transactions on Parallel and Distributed Systems* (2019).
3. G. C. Fox. "Data intensive applications on clouds." *In Proceedings of the second international workshop on Data intensive computing in the clouds*, pp. 1-2. ACM, 2011.
4. T. Gunarathne, T.-L. Wu, J. Qiu, & G. Fox. "Cloud computing paradigms for pleasingly parallel biomedical applications." *In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (HPDC'10), pp. 460-469. ACM, 2010.
5. P.-F. Dutot, G. Mounié, & D. Trystram, "Scheduling parallel tasks: approximation algorithms," in Handbook of Scheduling: Algorithms, Models, and Performance Analysis, J. Y.-T. Leung (Eds.), CRC Press, Boca Raton, 2004.
6. N. Jain, I. Menache, J. Naor, & J. Yaniv. "A Truthful Mechanism for Value-Based Scheduling in Cloud Computing." *In the 4th International Symposium on Algorithmic Game Theory*, pp. 178-189, Springer, 2011.
7. N. Jain, I. Menache, J. Naor, & J. Yaniv. "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*, pp. 255-266. ACM, 2012.
8. V. Nagarajan, J. Wolf, A. Balmin, & K. Hildrum. "Malleable scheduling for flows of jobs and applications to MapReduce." *Journal of Scheduling*, 2019, vol. 22, no 4, p. 393-411.
9. B. Lucier, I. Menache, J. Naor, & J. Yaniv. "Efficient online scheduling for deadline-sensitive jobs." *In Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 305-314. ACM, 2013.
10. Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. Naor, & J. Yaniv. "Truthful online scheduling with commitments." *In Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pp. 715-732. ACM, 2015.
11. P. Bodík, I. Menache, J. Naor, & J. Yaniv. "Brief announcement: deadline-aware scheduling of big-data processing jobs." *In Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pp. 211-213. ACM, 2014.

12. Eugene L. Lawler. “A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs.” *Annals of Operations Research* 26, no. 1 (1990): 125-133.
13. D. Karger, C. Stein, & J. Wein. “Scheduling Algorithms.” In CRC Handbook of Computer Science. 1997.
14. James R. Jackson. “Scheduling a Production Line to Minimize Maximum Tardiness.” *Management Science Research Project Research Report* 43, University of California, Los Angeles, 1955.
15. W. A. Horn. “Some Simple Scheduling Algorithms.” *Naval Research Logistics Quarterly*, 21:177-185, 1974.
16. E. L. Lawler, & J. M. Moore. “A Functional Equation and Its Application to Resource Allocation and Sequencing Problems.” *Management Science* 16, no. 1 (1969): 77-84.
17. J. A. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, “Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms.” *Kluwer Academic*, 1998.
18. T. White. “Hadoop: The definitive guide.” *O’Reilly Media, Inc.*, 2012.
19. J., Sudipto Guha, S. Khanna, and J. Naor. “Machine minimization for scheduling jobs with interval constraints.” *In Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science* (FOCS’04), pp. 81-90. IEEE, 2004.
20. D. P. Williamson and D. B. Shmoys. “The Design of Approximation Algorithm.” *Cambridge University Press*, 2011.
21. X. Wu, & P. Loiseau. ”Algorithms for scheduling deadline-sensitive malleable tasks.” *In Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing* (Allerton), pp. 530-537. IEEE, 2015.
22. L. Guo, & H. Shen. “Efficient Approximation Algorithms for the Bounded Flexible Scheduling Problem in Clouds.” *IEEE Transactions on Parallel and Distributed Systems* 28, no. 12 (2017): 3511-3520.
23. G. Even, “Recursive greedy methods,” in *Handbook of Approximation Algorithms and Metaheuristics*, T. F. Gonzalez, ed., CRC, Boca Raton, FL, 2007, ch. 5.
24. K. Jansen, & M. Rau. “Closing the Gap for Pseudo-Polynomial Strip Packing.” *In Proceedings of the 27th Annual European Symposium on Algorithms* (ESA’19), pp. 62:1-62:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
25. K. Jansen, & M. Rau. “Improved approximation for two dimensional strip packing with polynomial bounded width.” Theoretical Computer Science (2019).
26. M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, & D. Trystram. “Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms.” *Discrete Mathematics, Algorithms and Applications* 3, no. 04 (2011): 553-586.
27. K. Jansen, & F. Land. “Scheduling monotone moldable jobs in linear time.” *In 2018 IEEE International Parallel and Distributed Processing Symposium* (IPDPS’18), pp. 172-181. IEEE, 2018.
28. R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mounié, & D. Trystram. “Scheduling independent moldable tasks on multi-cores with GPUs.” *IEEE Transactions on Parallel and Distributed Systems* 28, no. 9 (2017): 2689-2702.
29. X. Wu, & P. Loiseau. “Efficient Algorithms for Scheduling Moldable Tasks.” arXiv:1609.08588v8 (2016).
30. X. Wu, & P. Loiseau. “Efficient approximation algorithms for scheduling moldable tasks.” European Journal of Operational Research 310, no. 1 (2023): 71-83.
31. G. Mounié, C. Rapine, & D. Trystram. “A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks.” *SIAM Journal on Computing* 37, no. 2 (2007): 401-412.
32. P.-F. Dutot, and D. Trystram. “Scheduling on hierarchical clusters using malleable tasks.” *In Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 199-208. ACM, 2001.
33. R. Lepère, D. Trystram, & G. J. Woeginger. “Approximation algorithms for scheduling malleable tasks under precedence constraints.” *International Journal of Foundations of Computer Science* 13, no. 04 (2002): 613-627.
34. K. Jansen, & L. Porkolab. “Linear-time approximation schemes for scheduling malleable parallel tasks.” *Algorithmica* 32, no. 3 (2002): 507-520.
35. W. Ludwig, & P. Tiwari. “Scheduling malleable and nonmalleable parallel tasks.” *In Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 167-176. SIAM, 1994.