A Practical 73/50 Approximation for Contiguous Monotone Moldable Job Scheduling

Klaus Jansen
Kiel University, Germany

Felix Ohnesorge
Kiel University, Germany

Abstract

In moldable job scheduling, we are provided m identical machines and n jobs that can be executed on a variable number of machines. The execution time of each job depends on the number of machines assigned to execute that job. For the specific problem of *monotone* moldable job scheduling, jobs are assumed to have a processing time that is non-increasing in the number of machines.

The previous best-known algorithms are: (1) a Polynomial Time Approximation Scheme (PTAS) with time complexity $\Omega(n^{g(1/\varepsilon)})$, where $g(\cdot)$ is a super-exponential function [Jansen and Thöle '08; Jansen and Land '18], (2) a Fully Polynomial Time Approximation Scheme (FPTAS) for the case of $m \geq 8\frac{n}{\varepsilon}$ [Jansen and Land '18], and (3) a $\frac{3}{2}$ approximation with time complexity $O(nm \log(mn))$ [Wu, Zhang, and Chen '23].

We present a new practically efficient algorithm with an approximation ratio of $\approx (1.4593 + \varepsilon)$ and a time complexity of $O(nm \log \frac{1}{\varepsilon})$. Our result also applies to the *contiguous* variant of the problem. In addition to our theoretical results, we implement the presented algorithm and show that the practical performance is significantly better than the theoretical worst-case approximation ratio.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Parallel computing models; Theory of computation → Scheduling algorithms

Keywords and phrases computing, machine scheduling, moldable, polynomial approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.56

Related Version *Full Version*: `https://arxiv.org/abs/2601.02836`

Supplementary Material *Software*: `https://github.com/Felioh/MoldableJobScheduling`

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project number 453769249.

1 Introduction

In the past many variants of scheduling problems have been studied. The classical task scheduling problem consists of scheduling n jobs to m identical machines, where each job j occupies exactly 1 machine. Motivated by many practical applications, such as high performance computing [8], where parallelism is exploited to speed up the execution of jobs, many extensions of this problem have been studied. One of which is scheduling *moldable* jobs. This is a natural extension where jobs can be executed on a variable number of machines. There are many practical motivations for this problem formulation, as discussed in [2, 8, 11].

In moldable job scheduling we are given $J = \{1, \ldots, n\}$ jobs and m identical machines. Each job can be assigned to $k \leq m$ machines and the processing time of any job j is dependent on the number of assigned machines. We denote the processing time of any job j as $t(j, k)$. Further, the *work* of any job j is defined as $w(j, k) = t(j, k) \cdot k$ and can be described as its area. For *monotone* moldable job scheduling we assume that for any job, the work function and time function are non-decreasing and non-increasing, respectively.

▶ **Assumption 1** (monotony [29]). *Given any job j, for all $k \leq k' \leq m$, we have:*

(i) $w(j,k) \leq w(j,k')$

(ii) $t(j,k) \geq t(j,k')$

A solution to an instance of this problem is given by a *schedule*, containing a start time s_j for each job $j \in J$, and an allotment $\alpha(j) \in \{1,\dots,m\}$ for each job. A schedule is called *feasible* if the following conditions are met:

- A job starts its execution simultaneously on all its assigned machines
- A job may not be interrupted during its execution time
- Each machine can execute at most one job at a time

In the *contiguous* variant of the problem, we additionally require each job to be processed on an adjacent set of machines. In many practical applications obtaining such a schedule, where jobs are processed on adjacent machines, is beneficial as it allows for better memory locality and reduced communication overhead between machines [2]. There are many other applications of this model, as discussed in [1, 5, 6, 13, 29, 31]. In this paper we consider the problem of minimizing the makespan, i.e., the maximum completion time of any job in a schedule.

1.1 Related Work

Note that in moldable job scheduling the number of machines assigned to a job is fixed during the execution of the job. Some older publications refer to this problem as *malleable* job scheduling [4, 12, 13, 21, 24, 27, 29]. However, the notation has recently changed to moldable job scheduling, whereas malleable job scheduling is now used to refer to the problem, where the number of machines assigned to a job can change during its execution.

Moldable job scheduling as well as the monotone variant are known to be NP-hard [9, 20]. Turek, Wolf, and Yu [30] presented a 2-approximation algorithm for the general case of moldable job scheduling. Later, Ludwig and Tiwari [27] presented an algorithm with the same approximation ratio but improved the running time to be polynomial in $\log m$. It has been proven that for moldable job scheduling (without monotony!) no polynomial-time approximation algorithm with a guarantee below 3/2 exists, unless P = NP [7, 25]. But there exists a pseudo-polynomial $(\frac{5}{4}+\varepsilon)$-approximation algorithm for scheduling contiguous moldable jobs with a time complexity of $O(n \log n) \cdot m^{f(1/\varepsilon)}$, where $f(\cdot)$ is a computable function [22]. For special cases of the problem there exist better approximation algorithms: Jansen and Porkolab [21] presented an approximation scheme with linear running time for a constant number of machines m; Decker, Lücking, and Monien [4] gave a 1.25-approximation under the additional assumption of identical jobs. Jobs are called identical if the execution time on any number of machines is the same for all jobs.

In this work we study the specific problem of *monotone* moldable job scheduling. Mounié, Rapine, and Trystram [28, 29] presented a $(\frac{3}{2}+\varepsilon)$-approximation algorithm with a time complexity of $O(nm \log \frac{1}{\epsilon})$ for the contiguous case. Over the past years, various authors have improved this result. Jansen and Thöle [23, 24] have shown the existence of an algorithm with an approximation ratio arbitrarily close to 1.5 for the contiguous problem and presented a Polynomial Time Approximation Scheme (PTAS) for the same problem without the contiguous restriction when m is bounded by a polynomial in n. Jansen and Land [19] generalized the PTAS in [24] by presenting a Fully Polynomial Time Approximation Scheme (FPTAS) for the contiguous case under the assumption of $m > 8\frac{n}{\varepsilon}$. Additionally, they improved the $(\frac{3}{2}+\varepsilon)$-approximation algorithm in [29] to have a running time with only poly-logarithmic dependence on the number of machines m. These result were further

improved by Grage, Jansen, and Ohnesorge [16]. Just recently, Wu, Zhang, and Chen [31] gave a $\frac{3}{2}$-approximation algorithm with a running time of $O(nm \log nm)$ and stated the open question of whether an approximation algorithm for this problem exists with an approximation ratio better than $\frac{3}{2}$ and a similar running time.

Table 1 Overview of approximation algorithms for monotone moldable job scheduling polynomial in n, m and $1/\varepsilon$.

Ratio	Complexity	Author (Year)
$\frac{3}{2}+\epsilon$	$O(\log(1/\varepsilon)nm)$	Mounié, Rapine, Trystram (2007) [29]
$\frac{3}{2}+\epsilon$	$O(\frac{n}{\varepsilon^2}\log m(\frac{\log m}{\epsilon}+\log^3(\varepsilon m)))$	Jansen, Land (2018) [19]
$\frac{3}{2}+\epsilon$	$O(n\log^2(\frac{1+\log(\varepsilon m)}{\varepsilon})+\frac{n}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\log(\varepsilon m))$	Grage, Jansen, Ohnesorge (2023) [16]
$\frac{3}{2}$	$O(nm\log(nm))$	Wu, Zhang, Chen (2023) [31]
$1.4593+\epsilon$	$O(\log(1/\varepsilon)nm)$	This Work

1.2 Our Contributions

The best known approximation ratio for the problem of monotone moldable job scheduling (non-contiguous) is $1+\varepsilon$, given by the PTAS in [23, 24]. This PTAS uses a dynamic program to place jobs across δ^{-2} many starting points. When setting $\varepsilon := 0.5$, we get the number of starting points by calculating $\delta := \sigma_{36}$, such that $\sigma_1 := \frac{1}{18}$ and $\sigma_k := \frac{\sigma_{k-1}^3}{(4\cdot 7^2)}$. This resolves to roughly $\sigma_{36} \approx 10^{-1.2\cdot 10^{17}}$. The overall dynamic program then has a running time of roughly $nm^{2\cdot 10^{2.4\cdot 10^{17}}}$. When comparing the $(\frac{5}{4}+\varepsilon)$-approximation in [22], to current $\frac{3}{2}$-approximation algorithms by setting $\varepsilon := \frac{1}{4}$, we get a running time of $\Omega((mn)^{16^{4^{13}}})$. While these algorithms are of theoretical interest, they have an impractically large running time. Motivated by this, we focus on *practically efficient* approximation algorithms (Table 1).

Previously it was assumed that there exists no practically efficient $(\frac{3}{2}-\varepsilon)$-approximation for the problem of monotone moldable job scheduling, for any $\varepsilon > 0$ [31]. We break this barrier by presenting an approximation algorithm with a worst-case approximation ratio of $\approx 1.4593 < 1.5$. It is worth noting that the PTAS presented in [24] cannot solve the contiguous variant of the problem, and it is unknown whether there exists a PTAS for this variant of the problem. Surprisingly, we show that our algorithm creates a contiguous schedule with a makespan of at most $1.4593 \cdot \widetilde{OPT}$, where $\widetilde{OPT}$ is a lower bound on the optimum of the non-contiguous problem. This result also bounds the gap between the contiguous and non-contiguous variant of the problem.

Our algorithm has a running time of $O(mn \log \frac{1}{\varepsilon})$. This is technically only pseudo-polynomial in the encoding length of the problem because of the linear dependency on m. Similar to the algorithms in [22, 24], we can use the FPTAS presented in [16, 19] for instances with $m > 8\frac{n}{\varepsilon}$. Thus, we may assume that $m \leq 8\frac{n}{\varepsilon}$. Our running time of $O(mn \log \frac{1}{\varepsilon})$ is fully polynomial in this case.

▶ **Theorem 2.** *Let $\varepsilon > 0$. For contiguous monotone moldable job scheduling, there exists an algorithm with an approximation ratio of $(1.4593+\varepsilon)$ and a running time of $O(nm \log \frac{1}{\varepsilon})$.*

We achieve this result by improving the long-standing algorithm of Mounié, Rapine, and Trystram [29]. Although this algorithm has been improved many times with respect to the running time [19, 16], improving the approximation ratio seemed to be a difficult task (except for eliminating the ε factor [31]). We manage this with the following new techniques: (1) relaxing the standard 0/1-Knapsack used in their algorithm to a Multiple-Choice Knapsack Problem, (2) packing the jobs into more containers, which results in more complex repair algorithms, and (3) analyzing a single critical case carefully with an elegant geometric argument. Our method of analyzing this critical case results in the so-called Lambert W function appearing in the approximation ratio.

We believe that this result is of theoretical interest as it breaks the long-standing barrier of $\frac{3}{2}$ for these practically efficient algorithms and the techniques employed, particularly the geometric argument, are potentially of broader interest. A similar geometric analysis could be beneficial in other scheduling and packing problems [10, 14, 17]. Furthermore, our theoretical results are supported by practical experiments demonstrating that a schedule length of at most $\frac{10}{7}OPT$ can be guaranteed in most cases. This makes this algorithm a promising candidate for practical applications.

1.3 Preliminaries

As many of the previous works, we utilize a dual approximation framework as proposed by Hochbaum and Shmoys [18]. First, we obtain a lower and upper bound on the optimum makespan by a constant approximation algorithm [30, 27] and then perform binary search in this interval to find the optimal makespan up to an accuracy of ε. Therefore, for this work we assume that we are given a makespan guess d and want to find a schedule with makespan at most λd or reject d if $d < OPT$.

Similar to Mounié, Rapine, and Trystram [29], we define the *canonical number of machines* as follows:

▶ **Definition 3** (Canonical Number of Machines). *Given a real number h, we define for each job j its canonical number of machines $\gamma(j,h)$ as the minimal number of machines needed to execute j in time at most h. If j cannot be processed in time at most h on m machines, we set $\gamma(j,h) = \infty$ by convention.*

We do note that, due to monotony, this number can be found in time $O(\log m)$ using binary search.

2 Description of the Algorithm

In order to motivate the core idea of our algorithm, we first give a brief overview of the algorithm presented by Mounié, Rapine, and Trystram [29]: First, find a partition $\mathcal{S}_1 \sqcup \mathcal{S}_2 = J_B = \{j \in J : t(j,1) > \frac{d}{2}\}$ with some specific properties. Then, schedule the first set $\mathcal{S}_1$ with a total width and height of m and d, respectively. The second set $\mathcal{S}_2$ is scheduled with a height of at most $\frac{d}{2}$ on top, which then results in a total height of at most $\frac{3}{2}d$. Afterwards, the remaining small jobs $J_S = \{j \in J : t(j,1) \leq \frac{d}{2}\}$ are scheduled greedily. For ease of notation we will, similar to [29, 21, 16], call the sets $\mathcal{S}_1, \mathcal{S}_2$ *shelf 1* and *shelf 2*, respectively. We describe the maximum allowed execution time of jobs in each shelf as the *height* of the shelf, e.g. in the algorithm of [29] the height of shelf 1 is d and the height of shelf 2 is $\frac{d}{2}$.

Although this algorithm is quite tailored to an approximation ratio of $\frac{3}{2}$, we manage to adapt it to obtain a better approximation ratio. A key insight is that we can partition the jobs into more sets to more closely resemble the allotment in an optimal schedule. We believe that this technique might be of more general interest. Improving the approximation ratio of this algorithm comes with a few major challenges: (1) Greedily adding jobs $j \in J_S$ with $t(j,1) \leq \frac{d}{2}$ will result in an approximation ratio $\geq 3/2$. (2) We need to be more careful about the partitioning of jobs into shelves.

We overcome the first challenge by defining $J_S = \{j \in J : t(j,1) \leq \frac{3}{7}d\}$ as the set of small jobs, and $J_B = J \setminus J_S$ as the set of big jobs. This choice, while it may seem arbitrary, will become clear in the subsequent analysis. The minimal work of the small jobs in any schedule is denoted as $\mathcal{W}_S = \sum_{j \in J_S} w(j,1)$.

For the partition we allow a third shelf, which is allowed to exceed the width of m. By exploiting the work monotony of jobs, we can later reduce the number of machines assigned to jobs in this shelf. This leads us to the central theorem of our work:

▶ **Theorem 4.** *Given a partitioning of jobs $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3 = J_B$ s.t.*

C1 $\sum_{j \in \mathcal{C}_1} w(j, \gamma(j,d)) + \sum_{j \in \mathcal{C}_2} w\left(j, \gamma\left(j, \frac{4}{7}d\right)\right) + \sum_{j \in \mathcal{C}_3} w\left(j, \gamma\left(j, \frac{3}{7}d\right)\right) \leq md - \mathcal{W}_S$

C2 $\sum_{j \in \mathcal{C}_1} \gamma(j,d) + \sum_{j \in \mathcal{C}_2} \gamma\left(j, \frac{4}{7}d\right) \cdot \frac{1}{2} \leq m$

one can compute a contiguous schedule with makespan at most λd in time $O(mn)$, for any $\lambda > -\frac{1}{3}W_{-1}(-\frac{3}{e^4}) \approx 1.4593$.

The Lambert W function, also called omega function or product logarithm is defined such that $ye^y = x$ holds if and only if $y = W(x)$. The W_{-1} branch is used for $-\frac{1}{e} \leq x < 0$ [3]. The fact that the Lambert W function appears is due to the nature of our analysis, where we define integrals to get a bound on the work of jobs in a schedule. This technique, while not new, is not commonly used in the analysis of scheduling problems. Fotakis, Matuschke, and Papadigenopoulos [12] use a similar analysis technique, but the Lambert W function does not appear in their analysis.

2.1 Finding a Partition

We start by showing the existence of a partition of jobs into three classes, as required by Theorem 4, given a schedule with makespan at most d exists.

▶ **Lemma 5.** *Given any schedule with makespan at most d, there exists a partition $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3 = J_B$ that satisfies constraints C1 and C2 of Theorem 4.*

Proof. Consider any schedule for an instance I with makespan $d^* \leq d$. The total work of all small jobs, defined as $J_S = \{j \in J : t(j,1) \leq \frac{3}{7}d\}$, is at least $\mathcal{W}_S = \sum_{j \in J_S} t(j,1)$, due to the assumption of work monotony. Therefore, the work of all remaining jobs is at most $\mathcal{W} = md - \mathcal{W}_S$. For any job $j \in J_B$, we will denote the number of machines assigned to this job in the optimal schedule as $OPT(j)$. We now argue that the partition $\mathcal{C}_1 \coloneqq \{j \in J_B : t(j, OPT(j)) > \frac{4}{7}d\}$, $\mathcal{C}_2 \coloneqq \{j \in J_B : \frac{4}{7}d \geq t(j, OPT(j)) > \frac{3}{7}d\}$, and $\mathcal{C}_3 \coloneqq \{j \in J_B : \frac{3}{7}d \geq t(j, OPT(j))\}$ of jobs in J_B satisfies both constraints C1 and C2.

Due to monotony, we know that the work of any job in $\mathcal{C}_3$ is at least $w(j, \gamma(j, \frac{3}{7}d))$. Therefore:

$$\sum_{j \in \mathcal{C}_3} w\left(j, \gamma\left(j, \frac{3}{7}d\right)\right) \leq \sum_{j \in \mathcal{C}_3} w(j, OPT(j)) \tag{1}$$

Each job in $\mathcal{C}_1$ uses at least $\gamma(j,d)$ machines and has a work of at least $w(j,\gamma(j,d))$, therefore:

$$\sum_{j\in\mathcal{C}_1} w(j,\gamma(j,d)) \leq \sum_{j\in\mathcal{C}_1} w(j,OPT(j)) \tag{2}$$

$$\sum_{j\in\mathcal{C}_1} \gamma(j,d) \leq \sum_{j\in\mathcal{C}_1} OPT(j) \tag{3}$$

For the last partition $\mathcal{C}_2$, first observe that in any optimal schedule of height at most d no job from $\mathcal{C}_2$ can be scheduled on the same machine as a job from $\mathcal{C}_1$. Thus, all jobs in $\mathcal{C}_2$ are scheduled on at most $m - \sum_{j\in\mathcal{C}_1} OPT(j)$ machines. Each job $j \in \mathcal{C}_2$ uses at least $\gamma(j, \frac{4}{7}d)$ machines and, since $t(j,OPT(j)) > \frac{3}{7}d$, at most two such jobs can be scheduled on the same machine. Therefore:

$$\sum_{j\in\mathcal{C}_2} \gamma\left(j,\frac{4}{7}d\right)\cdot\frac{1}{2} \leq m - \sum_{j\in\mathcal{C}_1} OPT(j) \tag{4}$$

For the work area, we give a similar argument as above and get:

$$\sum_{j\in\mathcal{C}_2} w\left(j,\gamma\left(j,\frac{4}{7}d\right)\right) \leq \sum_{j\in\mathcal{C}_2} w(j,OPT(j)) \tag{5}$$

Combining Equations (3) and (4), yields constraint C2 and combining Equations (1), (2), and (5), yields constraint C1. This proves the lemma. ◀

Thus, given a makespan guess $d \geq \mathrm{OPT}$, we know a partition of jobs $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3 = J_B$ exists such that the constraints of Theorem 4 hold. Finding such a partition is equivalent to solving a Multiple-Choice Knapsack Problem (MCKP).

▶ **Definition 6** (Multiple-Choice Knapsack Problem (MCKP))**.** *Given a capacity $m \in \mathbb{N}$, $k \in \mathbb{N}$ classes and $n \in \mathbb{N}$ items, each item $j \in [n]$ with a cost c_{jl} and a size s_{jl} for each $l \in [k]$. Find a partition $\mathcal{C}_1 \sqcup \cdots \sqcup \mathcal{C}_k = [n]$ with:*

$$\min \sum_{l=1}^{k}\sum_{j\in\mathcal{C}_l} c_{jl}$$

$$s.t. \sum_{l=1}^{k}\sum_{j\in\mathcal{C}_l} s_{jl} \leq m$$

Solving the MCKP can be done in time $O(mn)$ using dynamic programming [26]. Given a makespan guess $d \geq OPT$, finding a partition that satisfies the constraints of Theorem 4 is equivalent to solving a MCKP with capacity m and $k = 3$ classes, where each job $j \in J_B$ is an item with costs $c_{j1} = w(j,\gamma(j,d))$, $c_{j2} = w(j,\gamma(j,\frac{4}{7}d))$, and $c_{j3} = w(j,\gamma(j,\frac{3}{7}d))$ and sizes $s_{j1} = \gamma(j,d)$, $s_{j2} = \frac{1}{2}\gamma(j,\frac{4}{7}d)$, and $s_{j3} = 0$. Therefore, we will assume for the remainder of this work that we are given a partition $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3 = J_B$ that satisfies the constraints of Theorem 4.

2.2 Constructing a Schedule

In this section we assume that we are given a makespan guess $d \geq OPT$ and a partition $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3 = J_B$ that satisfies the constraints of Theorem 4. We will prove Theorem 4 by providing an algorithm with the required approximation ratio. An overview of this algorithm is given in Algorithm 1.

Algorithm 1 Description of the Dual Approximation Algorithm.

Require: m identical machines, n moldable jobs, and a makespan guess d
1: Let $J_S = \{j \in J | t(j,1) \leq \frac{3}{7}d\}$ and $J_B = J \setminus J_S$.
2: Construct partition $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3 = J_B$ via MCKP. ▷ Lemma 5
3: **if** $W(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3) > dm - \mathcal{W}_S$ **then**
4: **reject** d.
5: **end if**
6: Distribute jobs in $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ onto shelves according to Lemma 7. ▷ Definition 9
7: Apply Algorithm 2.
8: **if** $q < m/6$ **then**
9: Apply Algorithm 3. ▷ Lemma 11
10: **else**
11: Apply Algorithm 4. ▷ Lemma 13
12: **end if**
13: Greedily add small jobs J_S via Algorithm 5. ▷ Lemma 17

Our goal is now to construct an intermediate schedule, as shown in Figure 1 (left) for all jobs $J_B = J \setminus J_S$. Any job $j \in \mathcal{C}_1$ is scheduled on $\gamma(j,d)$ machines at time 0, any job $j \in \mathcal{C}_2$ is scheduled on $\gamma(j, \frac{4}{7}d)$ machines at time 0, and any job $j \in \mathcal{C}_3$ on $\gamma(j, \frac{3}{7}d)$ machines such that it completes at time $\frac{10}{7}d$. When analyzing the resulting schedule, we notice that (1) the jobs in $\mathcal{C}_1$ and $\mathcal{C}_2$ require more than m machines, and (2) the jobs in $\mathcal{C}_3$ might also require more than m machines.

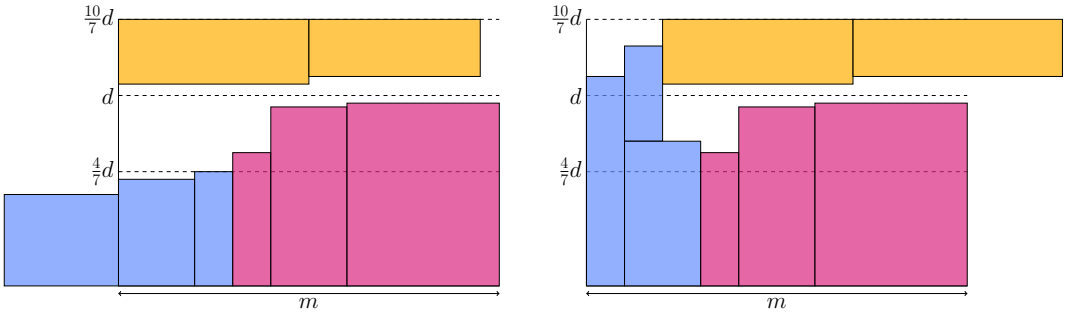

Figure 1 Schedule before Lemma 7 (left) and after (right). Pink jobs are in $\mathcal{C}_1$, blue jobs in $\mathcal{C}_2$, and yellow jobs in $\mathcal{C}_3$.

We fix the first problem by scheduling the jobs $j \in \mathcal{C}_2$ on fewer machines and focus on the second problem in Section 2.3. Note that all jobs are scheduled in a way such that Constraint C1 remains satisfied and the total work of all big jobs is at most $md - \mathcal{W}_S$. This remains true, due to work-monotony, for the remainder of this work by ensuring that no alterations to this schedule increase the number of machines assigned to any job.

▶ **Lemma 7.** *All jobs in $\mathcal{C}_2$ can be scheduled on $m - \sum_{j \in \mathcal{C}_1} \gamma(j,d)$ machines without exceeding the makespan of $\frac{10}{7}d$.*

Proof. First notice that by Constraint C2, we get $\sum_{j \in \mathcal{C}_2} \gamma(j, \frac{4}{7}d) \cdot \frac{1}{2} \leq m - \sum_{j \in \mathcal{C}_1} \gamma(j,d)$. Therefore, we need to effectively schedule each job, such that it requires at most $\gamma(j, \frac{4}{7}d) \cdot \frac{1}{2}$ machines.

We analyze jobs $j \in \mathcal{C}_2$ based on their canonical number of machines $\gamma(j, \frac{4}{7}d)$.

Case 1 $\gamma(j, \frac{4}{7}d) \geq 4$**.** These jobs can be scheduled in such a way that they do not exceed an execution time of $\frac{10}{7}d$ when assigned to $\lfloor \gamma(j, \frac{4}{7}d)/2 \rfloor$ machines. Due to work monotony, we have:

$$
\begin{aligned}
& w\left(j, \gamma\left(j, \frac{4}{7}d\right)\right) \geq w\left(j, \left\lfloor \frac{\gamma\left(j, \frac{4}{7}d\right)}{2} \right\rfloor\right) \\
\Leftrightarrow\ & t\left(j, \gamma\left(j, \frac{4}{7}d\right)\right) \cdot \gamma\left(j, \frac{4}{7}d\right) \geq t\left(j, \left\lfloor \frac{\gamma(j, \frac{4}{7}d)}{2} \right\rfloor\right) \cdot \left\lfloor \frac{\gamma(j, \frac{4}{7}d)}{2} \right\rfloor \\
\Rightarrow\ & \frac{4}{7}d \cdot \frac{\gamma(j, \frac{4}{7}d)}{\left\lfloor \frac{\gamma(j, \frac{4}{7}d)}{2} \right\rfloor} \geq t\left(j, \left\lfloor \frac{\gamma(j, \frac{4}{7}d)}{2} \right\rfloor\right)
\end{aligned}
$$

This implies that $t\left(j, \lfloor \gamma(j, \frac{4}{7}d)/2 \rfloor\right)$ is smaller than or equal to $\frac{10}{7}d$ for any $\gamma(j, \frac{4}{7}d) \geq 4$.

Case 2 $\gamma(j, \frac{4}{7}d) = 2$**.** These jobs can be scheduled on 1 machine without exceeding the makespan of $\frac{10}{7}d$, exactly halving the number of required machines.

Case 3 $\gamma(j, \frac{4}{7}d) = 3$**.** If there are multiple such jobs, they can be scheduled in pairs on top of each other without idle time. In the remainder of this work, we may treat these jobs as a single bigger job. Thus, we may assume there exists at most one of these jobs and denote it as j_3.

Case 4 $\gamma(j, \frac{4}{7}d) = 1$**.** These jobs can also be scheduled in pairs on top of each other. In the remainder of this work, we may treat these jobs as a single bigger job. A remaining single job, denoted j_1, may need to be processed differently.

If neither j_1 nor j_3 exists, we have successfully scheduled all jobs in $\mathcal{C}_2$ such that they require no more than $m - \sum_{j \in \mathcal{C}_1} \gamma(j, d)$ machines.

If only j_1 exists, the schedule still fits within the machine limit since $m - \sum_{j \in \mathcal{C}_1} \gamma(j, d)$ is of integer value.

If only j_3 exists, it can be scheduled on $\gamma(j_3, \frac{10}{7}d) \leq 2$ machines, and again the schedule fits within the machine limit.

If both j_1 and j_3 exist, j_3 is scheduled on 2 machines, with a height of at most $\frac{4}{7}d \cdot \frac{3}{2} = \frac{6}{7}d$, and j_1 is placed on top of one of these machines. We consider these two jobs as one job of height $t(j_3, 2) + t(j_1, 1)$ scheduled on 1 machine, and one job of height $t(j_3, 2)$ scheduled on 1 machine. ◀

▶ Remark 8. Note that splitting the job j_3 into two parts might break the contiguous property of the schedule. However, we can simply reorder the machines such that this job is scheduled on adjacent machines. We will revisit this problem in each of the following proofs individually to ensure at any point that both parts of j_3 can be scheduled on adjacent machines.

For the following sections, we will refer to the schedule constructed in Lemma 7 as a 3-shelf schedule (Definition 9 with $\lambda = \frac{10}{7}$). To that end, we distribute jobs in $\mathcal{C}_1 \sqcup \mathcal{C}_2 \sqcup \mathcal{C}_3$ onto three shelves, as shown in Figure 2. Jobs in $\mathcal{C}_1 \sqcup \mathcal{C}_2$ are distributed onto the first two shelves (S_0 and S_1) such that any job with an execution time greater than d is in S_0 and the remaining jobs are in S_1. The jobs in $\mathcal{C}_3$ are put into S_2 with an execution time of at most $\frac{3}{7}d$.

▶ **Definition 9** (3-Shelf Schedule)**.** *Given a makespan guess d and $\lambda \in [\frac{10}{7}, \frac{3}{2})$, a 3-shelf schedule is a partition of jobs $S_0 \sqcup S_1 \sqcup S_2 = J_B = J \setminus J_S$ such that:*

- $\sum_{j \in S_0} w(j, \gamma(j, \lambda d)) + \sum_{j \in S_1} w(j, \gamma(j, d)) + \sum_{j \in S_2} w(j, \gamma(j, (\lambda - 1)d)) \leq md - \mathcal{W}_S$
- $\sum_{j \in S_0} \gamma(j, \lambda d) + \sum_{j \in S_1} \gamma(j, d) \leq m$

Note that by Lemma 7, $\sum_{j\in S_0}\gamma(j,\frac{10}{7}d)+\sum_{j\in S_1}\gamma(j,d)\leq m$. The total work of the jobs is still bounded by $md-\mathcal{W}_S$. As illustrated in Figure 2, we now have to consider the remaining problem that shelf 2 might require too many machines.

2.3 Repairing the 3-Shelf Schedule

This section is devoted to repairing the 3-shelf schedule obtained in the previous section. Although the above knapsack formulation suggests a makespan guarantee of $\frac{10}{7}d$ is achievable, we will have to relax this to obtain a feasible schedule. To this end, we parameterize the approximation ratio of our algorithm by a parameter $\lambda\in[\frac{10}{7},\frac{3}{2})$, to be chosen later. Thus, we aim to construct a schedule of height at most λd, where we allow jobs in shelf 2 to have an execution time of at most $(\lambda-1)d$.

We will denote the number of machines used by S_0 and S_2 as $m_0\coloneqq\sum_{j\in S_0}\gamma(j,\lambda d)$ and $m_2\coloneqq\sum_{j\in S_2}\gamma(j,(\lambda-1)d)$, respectively. By construction $m-m_0$ machines are used to schedule jobs in S_1 and S_2. Lemma 7 guarantees that the jobs in S_1 require at most $m-m_0$ machines with height of at most d. We denote the number of idle machines in shelf 1, i.e. machines that do not execute any job $j\in S_1$, as q. Note that it is not yet guaranteed that the jobs in S_2 can be scheduled on the remaining $m-m_0$ machines with a maximal height of $(\lambda-1)d$, for any $\lambda\in[\frac{10}{7},\frac{3}{2})$. For the remainder of this section, we assume that we are given a 3-shelf schedule with $m_2>m-m_0$, as the schedule is feasible otherwise (Figure 2).

The following proofs use a lower bound on the total work of jobs to show the desired approximation ratio. Remember that, by Constraint C1 of Theorem 4, the given schedule satisfies $\mathcal{W}_0+\mathcal{W}_1+\mathcal{W}_2\leq md$, where $\mathcal{W}_k$ denotes the total work area of jobs in shelf k. Since, by construction, the total work of jobs in shelf 0 is greater than dm_0, we get $\mathcal{W}_1+\mathcal{W}_2\leq d(m-m_0)$. For ease of notation, we will w.l.o.g ignore shelf 0 and assume that $m_0=0$ as illustrated in Figure 2 (right).

We start by applying transformations (Algorithm 2) to the 3-shelf schedule, which will help reduce the number of machines used by S_2. These transformations are based on ideas from Mounié, Rapine, and Trystram [29] but are parameterized with λ.

Algorithm 2 Repair Shelf Schedule.

Require: 3-shelf schedule; $\lambda\in[\frac{10}{7},\frac{3}{2})$.
1: **while** possible **do**
2: apply Transformation T1, T2, or T3.
3: **end while**

T1 If a job $j\in S_1$ has execution time at most $\frac{\lambda}{2}d$ and is allotted to $p>1$ machines, allocate j to $\gamma(j,\lambda d)$ machines in S_0.

T2 If $j,j'\in S_1$ have execution time less than $\frac{\lambda}{2}d$ and are each allotted to 1 machine, allocate j and j' to the same machine in S_0.

T3 Let q denote the number of idle machines in S_1. If there exists a job $j\in S_2$ with an execution time of less than λd on q machines, allocate j on $\gamma(j,\lambda d)$ machines either to S_1 or S_0, depending on its execution time.

In the following, we will analyze the schedule that results after applying Algorithm 2. Remember, that for ease of notation, we assume $m_0=0$. This does not change the correctness of the proof, since we contradict the total work of the schedule, and shelf 0 has a total work greater than dm_0.

▶ **Lemma 10.** *Given a 3-shelf schedule and any $\lambda \in [\frac{10}{7}, \frac{3}{2})$, we can either find a feasible contiguous schedule with makespan at most λd in time $O(mn)$, or the following properties hold:*

(i) *At most one machine executes a job $j \in S_1$ with height less than $\frac{\lambda}{2}d$.*

(ii) *The work area of any job in S_2 is greater than λdq.*

(iii) *The work area of S_1 is greater than $\frac{\lambda}{2}d\,(m-q)$.*

Proof. We prove the lemma by first showing that Properties (i) and (ii) hold after applying Algorithm 2. Then, we will establish the contiguity of the schedule and analyze the time complexity of the algorithm.

We begin by showing the three properties of the lemma. By the definition of Algorithm 2, any job $j \in S_1$ is scheduled on $\gamma(j,d)$ machines. This implies that any job with an execution time of less than $\frac{\lambda}{2}d$ is scheduled on exactly one machine, by the definition of $\gamma(j,d)$ and monotony. The schedule contains at most one such job, since we could otherwise apply Transformation T2, contradicting the algorithms' termination condition. Property (ii) follows directly, since Transformation T3 cannot be applied.

For the last property we use a proof by contradiction: Suppose Property (iii) does not hold. This implies that there exists a job $j \in S_1$ with an execution time of $t \in (\frac{3}{7}d, \frac{\lambda}{2}d]$. By Property (i), there exists at most one such job, which is scheduled on exactly one machine. We consider three cases:

Case 1 ($S_1 \setminus \{j\} = \emptyset$ and $q = 0$). In this case, $m = 1$. We place all jobs on this one machine and obtain a feasible schedule, if $\mathcal{W}_1 + \mathcal{W}_2 \leq d$.

Case 2 ($S_1 \setminus \{j\} = \emptyset$ and $q > 0$). In this case, $q = m - 1$. Since S_2 contains at least one job $\mathcal{W}_2 > \lambda dq = \lambda d(m-1)$, by Property (ii). With $\mathcal{W}_1 + \mathcal{W}_2 \leq dm$, we get:

$$\begin{aligned}
&\frac{3}{7}d + \lambda d(m-1) < dm \\
&\Leftrightarrow \frac{3}{7} + \lambda m - \lambda < m \\
&\Leftrightarrow (\lambda - 1)m < \lambda - \frac{3}{7} \\
&\Leftrightarrow m < \frac{\lambda - \frac{3}{7}}{\lambda - 1}
\end{aligned}$$

For $\lambda \in [\frac{10}{7}, \frac{3}{2})$, this implies $m \leq 2$. Since $q \geq 1$ and $m = q + 1$, we must have $m = 2$ and $q = 1$. We will first show that S_2 can contain at most one job. To this end, denote the number of jobs in S_2 as k, then the total work area of S_2 is $\mathcal{W}_2 > \lambda dqk = \lambda dk$, by Property (ii) and $q = 1$. With $\mathcal{W}_1 + \mathcal{W}_2 \leq dm$ we get:

$$\begin{aligned}
&\frac{3}{7}d + \lambda dk < 2d \\
&\quad\Leftrightarrow k < \frac{11}{7\lambda} < \frac{11}{10} < 2 \qquad\qquad (\lambda \geq \frac{10}{7})
\end{aligned}$$

This leads to a final contradiction. If we cannot find a feasible schedule, the single job in S_2 has a processing time of at least $\lambda d - t$ on two machines. This implies a total work area of $2(\lambda d - t) + t \geq \frac{3}{2}\lambda d > 2d$, where the first inequality follows from $t \leq \frac{\lambda}{2}d$ and the second from $\lambda > \frac{10}{7}$. Since $\mathcal{W}_1 + \mathcal{W}_2 \leq 2d$, this is a contradiction.

Case 3 ($S_1 \setminus \{j\} \neq \emptyset$). If Property (iii) does not hold, we may assume that there exists another job $j' \in S_1 \setminus \{j\}$ with an execution time less than $\lambda d - t$. Then, we can schedule j on top of j'. Observe that $t(j,d) + t(j',d) > d$, therefore we place j (and j' partially) in S_0. The new schedule satisfies (iii).

It remains to show that the resulting schedule is contiguous. We must ensure that no job is split across non-adjacent machines. Note that this can only happen for jobs that are split between two shelves. If such a job j_3 exists (see Remark 8), we can choose j (the job violating Property (iii)) to be $j := j_3$. We consider the three cases from above: In Cases 1 and 2, job j $(= j_3)$ is the only job in S_1. We can schedule it at the boundary between shelf 1 and shelf 0, ensuring it occupies a contiguous block of machines. In case 3, j $(= j)$ is moved entirely into shelf 0, so it is no longer split. The other job j' can be chosen as the smallest job in S_1 and placed at the boundary, preserving contiguity. This guarantees that all jobs are scheduled on adjacent machines.

The time complexity results from the following argument: Denote the number of big jobs as n_B. Since the transformations move jobs from shelf 2 to shelf 1 or 0, and from shelf 1 to shelf 0, the while-loop in Algorithm 2 can be executed at most $O(n_B)$ times. Transformations T1 and T2 can be implemented in $O(n_B)$. Transformation T3 can be implemented in $O(n_B)$ for scanning the jobs in S_2 and $O(\log m)$ to determine $\gamma(j, \lambda d)$ for the chosen job $j \in S_2$. This results in a time complexity of $O(n_B^2 + n_B \log m)$. Since n_B is bounded by both n and m, we can conclude that the time complexity of Algorithm 2 is $O(nm)$. ◀

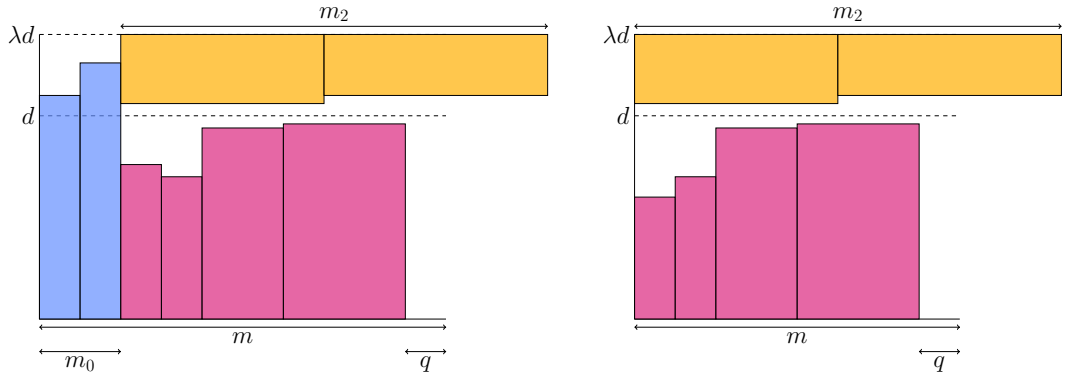

Figure 2 Infeasible 3-shelf schedule. Blue jobs are in S_0, pink jobs in S_1, and yellow jobs in S_2.

Our goal for the remainder of this section is now to show that S_2 does not require more than m machines. The new idea is to compress jobs in S_2 to a height greater than the height of their shelf ($(\lambda - 1)d$), since the jobs in S_1 cannot be utilizing the entire height of their shelf. We formulate multiple new repair algorithms that utilize this observation, and show that they can construct a feasible schedule in several ways.

▶ **Lemma 11.** *Given a 3-shelf schedule with $q \leq m/6$, we can find a feasible contiguous schedule with makespan at most $\frac{10}{7}d$ (if $q = 0$), and $\frac{13}{9}d$ (if $0 < q \leq m/6$) in time $O(mn)$.*

Proof. In this case a feasible schedule can be found by applying Algorithm 3. The proof of this lemma uses fairly simple techniques, therefore we defer the proof of this lemma to the full version. ◀

Algorithm 3 RepairS2-1.

while $m_2 > m$ **do**
  Let j be the job with the smallest height among jobs in S_2.
  Let p be the number of machines j is allotted to.
  Allot j to $p-1$ machines.
end while
Sort jobs in S_1 in descending order of execution time.
Sort jobs in S_2 in ascending order of execution time.

It remains to show that we can also find a feasible schedule for the case $q > m/6$. We prepare this proof with the following observation:

▶ **Observation 12.** *Given a 3-shelf schedule with $q > m/6$ and $\lambda \geq \frac{4}{3}$, the number of jobs in S_2 is at most $|S_2| \leq 1$.*

Proof. Suppose $|S_2| > 1$. By Lemma 10.(ii), we have ($\mathcal{W}_2 > \lambda dq \cdot |S_2| \geq 2\lambda dq$). With Lemma 10.(iii), we get:

$$\begin{aligned} & \mathcal{W}_1 + \mathcal{W}_2 \leq dm \\ \Rightarrow\ & \frac{\lambda}{2} d(m-q) + 2\lambda dq < dm \\ \Leftrightarrow\ & \frac{\lambda}{2} m + \frac{3}{2}\lambda q < m \\ \Leftrightarrow\ & \frac{3}{4}\lambda m < m \qquad (q > m/6) \\ \Leftrightarrow\ & \lambda < \frac{4}{3} \end{aligned}$$

This is a contradiction to $\lambda \geq \frac{4}{3}$. ◀

Algorithm 4 RepairS2-2.

Denote the single job in S_2 as j_0
Sort all jobs in S_1 in descending order of execution time
Set $i = 0$
for each $i \in \{0, \ldots, m-q-1\}$ **do**
  Place j_0 on the $m-i$ least loaded machines
  if $C_{\max} \leq \lambda d$ **then**
    return
  end if
end for

With this, we may assume that $|S_2| = 1$ and denote the single job in S_2 as j_0. This observation can be exploited by enumerating all possible allotments for this job (Algorithm 4).

The following lemma analyzes the correctness of the last (and most complex) case.

▶ **Lemma 13.** *Given a 3-shelf schedule with $q > m/6$, we can find a feasible schedule with makespan at most λd, for any $\lambda \geq -\frac{1}{3} W_{-1}(-\frac{3}{e^4}) \approx 1.4593$.*

Proof. We intend to prove the lemma by showing that the height of any job in S_1 scheduled on machine $i \in \{0, \ldots, m-1\}$ is greater than $\max\left(\lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}, \frac{\lambda}{2}d\right)$, provided we cannot find a feasible schedule with Algorithm 4 (as illustrated in Figure 3). With this observation, we can analyze the total work area of the schedule and contradict $\mathcal{W}_1 + \mathcal{W}_2 \leq dm$.

Consider a schedule as illustrated in Figure 3 (left), where Algorithm 4 cannot place $j_0 \in S_2$. We start by showing that when placing all jobs in shelf 1 in non-increasing order of processing time, the processing time of jobs (load L_i) on each machine $i \in \{0, \ldots, m-1\}$ is greater than $g(i) = \lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}$.

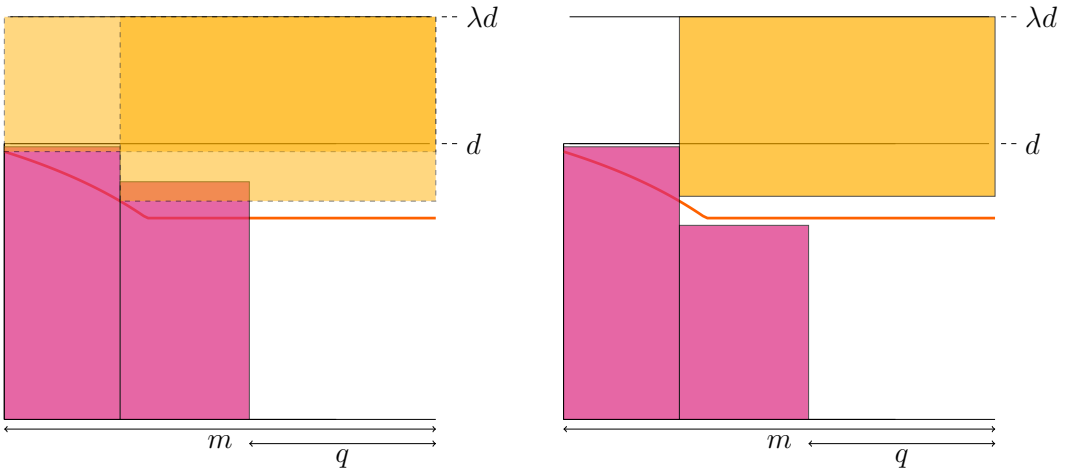

Figure 3 Infeasible (left) and feasible (right) schedule after applying Algorithm 4. The red line indicates $\max\left(\frac{\lambda d}{2}, \lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}\right)$. Pink jobs are in S_1 and yellow jobs in S_2.

▶ **Assumption 14.** *If Algorithm 4 does not find a feasible schedule, the jobs in S_1 can be placed such that the load L_i on each machine $i \in \{0, \ldots, m-1\}$ satisfies:*

$$L_i > g(i) = \lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}$$

This assumption can be proven by a simple contradiction argument. We formally show this in the full version of this paper.

It remains for us to analyze the total work area of the schedule under Assumption 14 to reach a contradiction to $\mathcal{W}_1 + \mathcal{W}_2 \leq dm$. We will split this analysis into two cases.

Case 1 ($m < 6$): In this case, we explicitly calculate a bound on $\mathcal{W}_1 + \mathcal{W}_2$. For more details please refer to the full version of this paper.

Case 2 ($m \geq 6$): The most important observation in this case is that we can correctly bound the work on shelf 1 by the following integral:

$$\mathcal{W}_1 > \int_0^{m-q} \max\left\{\lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}, \frac{\lambda}{2}d\right\} di \tag{6}$$

This bound is not entirely trivial, since shelf 1 might contain one job with an execution time of less than $\frac{\lambda}{2}d$, by Lemma 10.(i). We formally show the correctness of this bound, under the assumption of $m \geq 6$, in the full version of this paper.

In order to reach a contradiction, we need to show that

$$\begin{aligned}
&\mathcal{W}_1 + \mathcal{W}_2 \\
&> \int_0^{m-q} \max\left\{\lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}, \frac{\lambda}{2}d\right\} di + \max\left\{(\lambda-1)dm, \lambda dq\right\} \\
&> dm
\end{aligned}$$

The first inequation follows directly from Equation (6), Lemma 10(ii), and $\gamma(j_0, (\lambda-1)d) > m \Rightarrow \mathcal{W}_2 > (\lambda-1)dm$. In order to show the second inequation, we will interpret this expression as a function in q and show that its minimum, at $q = \frac{\lambda-1}{\lambda}m$, is greater than dm.

▶ **Observation 15.** *The function*

$$\mathcal{W}(q) = \int_0^{m-q} \max\left\{\lambda d - \frac{dm - \frac{\lambda}{2}d(m-q)}{m-i}, \frac{\lambda}{2}d\right\} di + \max\left\{(\lambda-1)dm, \lambda dq\right\}$$

has a minimum at $q = \frac{\lambda-1}{\lambda}m$, for any $\lambda \in [\frac{10}{7}, \frac{3}{2})$.

We give a full proof of this observation in the full version of this paper.

Since $\mathcal{W}(q)$ has the minimal value (in the interval $[0, m]$) at $q = \frac{\lambda-1}{\lambda}m$, we need to show that:

$$\begin{aligned}
&\mathcal{W}\left(\frac{\lambda-1}{\lambda}m\right) > dm \\
\Leftrightarrow &\int_0^{m-\frac{\lambda-1}{\lambda}m} \max\left\{\lambda d - \frac{\frac{1}{2}dm}{m-i}, \frac{\lambda}{2}d\right\} di + (\lambda-1)dm > dm \\
\Leftrightarrow &\int_0^{m-\frac{\lambda-1}{\lambda}m} \max\left\{\lambda d - \frac{\frac{1}{2}dm}{m-i}, \frac{\lambda}{2}d\right\} di > (2-\lambda)dm \qquad (7)
\end{aligned}$$

In order to further analyze this expression, we need to take a closer look at the integral term. First, let's find the switching point i' of the max-function, where $\lambda d - \frac{\frac{1}{2}dm}{m-i'} = \frac{\lambda}{2}d$.

$$\begin{aligned}
&\lambda d - \frac{\frac{1}{2}dm}{m-i'} = \frac{\lambda}{2}d \\
\Leftrightarrow\ &\frac{\frac{1}{2}dm}{m-i'} = \frac{\lambda}{2}d \\
\Leftrightarrow\ &dm = \lambda d(m-i') \\
\Leftrightarrow\ &\frac{\lambda-1}{\lambda}m = i'
\end{aligned}$$

Then, we can split the integral from Equation (7) at i' and analyze both parts separately.

$$\begin{aligned}
&\int_0^{m-\frac{\lambda-1}{\lambda}m} \max\left\{\lambda d - \frac{\frac{1}{2}dm}{m-i}, \frac{\lambda}{2}d\right\} di \\
&= \int_0^{\frac{\lambda-1}{\lambda}m} \left(\lambda d - \frac{\frac{1}{2}dm}{m-i}\right) di + \int_{\frac{\lambda-1}{\lambda}m}^{m-\frac{\lambda-1}{\lambda}m} \frac{\lambda}{2}d\, di \\
&= \lambda d \int_0^{\frac{\lambda-1}{\lambda}m} 1\, di - \int_0^{\frac{\lambda-1}{\lambda}m} \frac{\frac{1}{2}dm}{m-i}\, di + \left(\left(m - \frac{\lambda-1}{\lambda}m\right) - \frac{\lambda-1}{\lambda}m\right)\frac{\lambda}{2}d \\
&= (\lambda-1)\, dm - \frac{1}{2}dm \int_0^{\frac{\lambda-1}{\lambda}m} \frac{1}{m-i}\, di + \frac{2-\lambda}{2}dm
\end{aligned}$$

$$= (\lambda - 1)\, dm - \frac{1}{2} dm \left(\ln(m) - \ln\left(m - \frac{\lambda - 1}{\lambda} m \right) \right) + \frac{2 - \lambda}{2} dm$$
$$= (\lambda - 1)\, dm - \frac{1}{2} dm \ln(\lambda) + \frac{2 - \lambda}{2} dm$$

Together with Equation (7), we get:

$$\begin{aligned}
(\lambda - 1)\, dm - \frac{1}{2} dm \ln(\lambda) + \frac{2 - \lambda}{2} dm &> (2 - \lambda) dm \\
\Leftrightarrow -dm \ln(\lambda) + (2 - \lambda)\, dm &> (6 - 4\lambda) dm \\
\Leftrightarrow \ln(\lambda) &< 3\lambda - 4 \\
\Leftrightarrow \lambda &< e^{3\lambda - 4} \\
\Leftrightarrow \lambda e^{-3\lambda} &< e^{-4} \\
\Leftrightarrow -3\lambda e^{-3\lambda} &> -3e^{-4}
\end{aligned}$$

The Lambert W function is defined such that $ye^y = x$, iff $W(x) = y$. We note that we use the W_{-1} branch of the Lambert function and since this branch is monotonically decreasing we get, with $x := -3e^{-4}$, $y := -3\lambda$, the following inequation.

$$-3\lambda < W_{-1}(-3e^{-4})$$
$$\Leftrightarrow \lambda > -\frac{1}{3} W_{-1}(-3e^{-4})$$

Thus, we reach a contradiction to $\mathcal{W}_1 + \mathcal{W}_2 > dm$ for any $\lambda > -\frac{1}{3} W_{-1}(-3e^{-4}) \approx 1.4593$.

This concludes the proof of Lemma 13. ◀

▶ **Observation 16.** *The feasible schedule constructed by Algorithm 4 can be rearranged to a contiguous schedule without altering the makespan.*

Proof. Note that by definition, any job contained entirely in one shelf can be scheduled on a contiguous set of machines. There exists at most one job j_3, mentioned in Remark 8, that might be split between two shelves. A key observation for this proof is that, although the order of jobs in S_1 is important, we can partition the jobs in S_1 into two groups. The first group contains all jobs that fit underneath the singular job in S_2 and the second group contains all jobs that do not fit underneath this job. The order within these two groups does not matter. Thus, we can place the job j_3 on the edge of shelf 1, and if required, the allotment can be mirrored. With this we can guarantee that the job j_3 is scheduled next to its counterpart in S_0 on adjacent machines. ◀

Lemmas 11 and 13, and Observation 16 complete the proof of Theorem 4 and show that we can find a feasible contiguous schedule for all jobs in J_B.

3 Adding the Small Jobs

To complete the dual approximation, we need to add the previously ignored small jobs J_S. To this end, we recall that the total work on the obtained shelf schedule is at most $md - \mathcal{W}_S$.

Algorithm 5 Adding the small jobs.

Require: Shelf-Schedule for J_B
1: **for each** $j \in J_S$ **do**
2: Schedule job j on the least loaded machine.
3: **end for**

▶ **Lemma 17.** *Given a feasible contiguous schedule of length at most λd for J_B with a work area of at most $md - \mathcal{W}_S$, Algorithm 5 delivers a feasible contiguous schedule with makespan at most λd for the entire instance $J_B \cup J_S$, for any $\lambda \geq \frac{10}{7}$.*

Proof. Consider a schedule, as constructed in Section 2. We schedule all jobs in shelf 1 with a starting time of 0 and all jobs in shelf 2 such that they complete at exactly λd. This way the idle times on each machine are uninterrupted.

The load of each machine is defined by the sum of execution times of the jobs scheduled on it. By definition, the idle time of any machine i is equal to λd minus the load on that machine.

Now suppose Algorithm 5 does not deliver a feasible λd schedule. In this case, there must exist a machine i with a load strictly greater than λd after adding a small job. This implies that the load on that machine, before adding the small job, was greater than d, since all small jobs have an execution time of at most $\frac{3}{7}d$ on one machine, by definition. With i being the least loaded machine, this contradicts the assumption that the total work area of this schedule is at most md. ◀

The time complexity of the presented algorithm is $O(nm)$. With a dual approximation framework, we can find a feasible schedule with makespan at most $(1.4593 + \varepsilon)OPT$ in time $O(nm \log 1/\varepsilon)$.

4 Experimental Evaluation

In addition to the theoretical improvement on the current state-of-the-art algorithms, we also implemented the presented algorithm. In the following, we present some experimental results of the algorithm on randomly generated instances and show that the algorithm is capable of solving realistic instances in reasonable time. The implementation is in Java and publicly available on GitHub (`https://github.com/Felioh/MoldableJobScheduling`) as well as the test-instances and results. We tested on randomly generated instances with n jobs and m machines, where the execution time of each job is uniformly distributed in $[1, 100]$ (on one machine) and the remaining values are uniformly distributed, respecting monotony restrictions.

All tests were run on a single core of an Intel i5-8400T CPU with 2GB of RAM available to the Java process.

We split the tests into two groups, one where we fixed the number of jobs to $n = 1000$ and varied the number of machines $m \in [500, 2000]$, and one where we fixed the number of machines to $m = 1000$ and varied the number of jobs $n \in [500, 2000]$. For all tests, we set $\varepsilon = 0.05$.

Figures 4 and 5 show the execution time of our implementation for these two groups of tests. Although we did not take any special care to optimize the implementation, the algorithm is able to solve all instance in less than 30 seconds. The execution time decreases for $n >> m$, since in these instances the optimal makespan is bigger. This results in more jobs being classified as small jobs and an easier to solve instance.

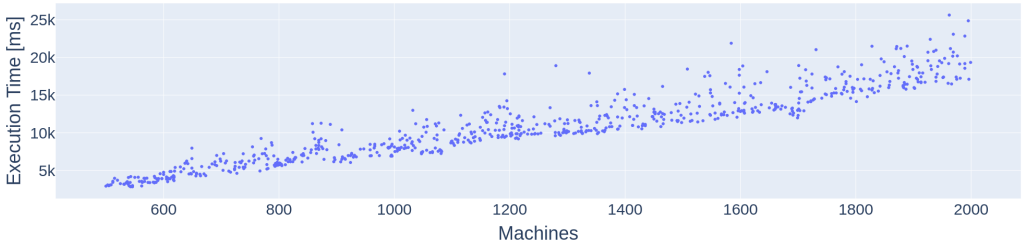

Figure 4 Runtime of the algorithm for varying number of machines.

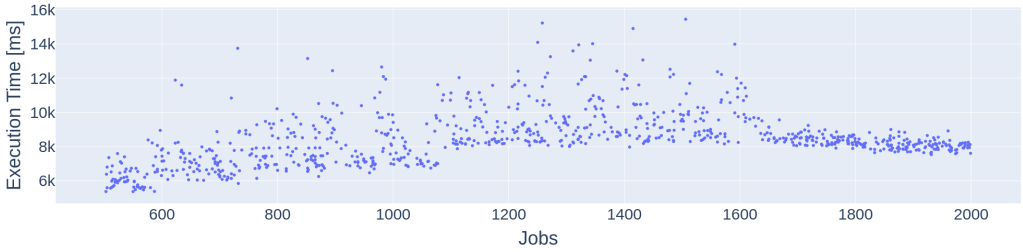

Figure 5 Runtime of the algorithm for varying number of jobs.

We note that while the theoretical worst-case approximation ratio is $1.4593 + \varepsilon$, the algorithm was able to guarantee a makespan below $(\frac{10}{7} + \varepsilon)OPT$ in all randomly generated instances. This is due to the fact that the worst-case approximation ratio only occurs in very specific cases. To be more precise, the presented algorithm can guarantee a makespan of at most $(\frac{13}{9} + \varepsilon)OPT$, unless the number of idle machines in shelf 1 is greater than $m/6$ (see Lemma 13).

However, we can show that our analysis is tight in the worst case. Consider the following instance with $m = 13$ and $n = 10$. All jobs have a constant work-function with the following values:

- j_1: $w(j,k) = 6.01, \quad \forall k \in \{1,\dots,m\}$
- j_2: $w(j,k) = 0.99, \quad \forall k \in \{1,\dots,m\}$
- $j_3 - j_{10}$: $w(j,k) = \frac{3}{4}, \quad \forall k \in \{1,\dots,m\}$

An optimal schedule for this instance has a makespan of $C_{\max} = 1$, since the work of all jobs is $W = 13$ and by placing all jobs on $m = 13$ machines, we can achieve a makespan of $OPT = W/m = 1$. On the other hand, an optimal solution to the MCKP problem in our algorithm is given by $\mathcal{C}_1 = \{j_2,\dots,j_{10}\}$, $\mathcal{C}_2 = \emptyset$, $\mathcal{C}_3 = \{j_1\}$. In this case, the algorithm will schedule j_1 on 13 machines and all remaining jobs on 1 machine each. This results in a makespan of $C_{\max} = 6.01/13 + 0.99 \approx 1.452$. For larger m, we can construct similar instances and achieve an approximation ratio arbitrarily close to our theoretical worst-case guarantee of $1.4593 + \varepsilon$. Due to the very specific structure of these instances, they are highly unlikely to occur in practice.

5 Conclusion

The algorithm presented in this work achieves an approximation ratio slightly below $(73/50+\varepsilon)$ for any $\varepsilon > 0$ in time complexity $O(nm \log 1/\varepsilon)$ for the problem of scheduling monotone moldable jobs. Even more surprisingly, we also show how to apply this algorithm to the contiguous variant of the problem. Since we construct the same schedule for the contiguous variant as for the non-contiguous variant, this bounds the gap between these two variants, which is an interesting side effect of our algorithm.

We show in an implementation that the algorithm achieves a ratio of $10/7$ in most cases and the worst-case approximation ratio of $1.4593+\varepsilon$ only happens in very specific problematic cases.

5.1 Future Work

The presented algorithm breaks the long-standing barrier of a practically efficient algorithm with an approximation ratio below 1.5 for the problem of scheduling monotone moldable jobs.

We hope that this work will lead to further improvements in the field of scheduling monotone moldable jobs. In particular, we leave the following open questions for future work:

- *Can the approximation ratio be improved further?* There is no known lower bound on the approximation ratio for practically efficient algorithms. As we showed, the presented algorithm already achieves an approximation ratio below the worst case guarantee in most cases. This question is specifically interesting for the contiguous variant, since no PTAS is known.
- *Can the running time be improved?* Jansen and Land [19] introduced the concept of *compression* to achieve a logarithmic dependence on the number of machines m in the running time of the algorithm. This was improved by Grage, Jansen, and Ohnesorge [16] by utilizing (min, +)-convolution. These techniques could potentially be applied to our algorithm to reduce the dependence on m in the running time.
- *Can this result be applied to related problems?* The presented algorithm could lead to improvements in other the closely related problems such as CPU/GPU scheduling [15], 2D Knapsack [14], strip packing [17], or demand strip packing [10] (for practically efficient algorithms).

References

1 Jacek Blazewicz, T. C. Edwin Cheng, Maciej Machowiak, and Ceyda Oguz. Berth and quay crane allocation: a moldable task scheduling model. *J. Oper. Res. Soc.*, 62(7):1189–1197, 2011. doi:10.1057/jors.2010.54.

2 Raphaël Bleuse, Sascha Hunold, Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. Scheduling independent moldable tasks on multi-cores with gpus. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2689–2702, 2017. doi:10.1109/TPDS.2017.2675891.

3 Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the lambert w function. *Advances in Computational mathematics*, 5:329–359, 1996. doi:10.1007/BF02124750.

4 Thomas Decker, Thomas Lücking, and Burkhard Monien. A 54-approximation algorithm for scheduling identical malleable tasks. *Theoretical Computer Science*, 361(2-3):226–240, 2006. doi:10.1016/j.tcs.2006.05.012.

5 Xavier Delorme, Alexandre Dolgui, Sergey Kovalev, and Mikhail Y. Kovalyov. Minimizing the number of workers in a paced mixed-model assembly line. *Eur. J. Oper. Res.*, 272(1):188–194, 2019. doi:10.1016/j.ejor.2018.05.072.

6 Alexandre Dolgui, Sergey Kovalev, Mikhail Y. Kovalyov, Sergey Malyutin, and Ameur Soukhal. Optimal workforce assignment to operations of a paced assembly line. *Eur. J. Oper. Res.*, 264(1):200–211, 2018. doi:10.1016/j.ejor.2017.06.017.

7 Maciej Drozdowski. On the complexity of multiprocessor task scheduling. *Bulletin of The Polish Academy of Sciences-technical Sciences*, 43:381–392, 1995.

8 Maciej Drozdowski. *Scheduling for parallel processing*, volume 18. Springer, 2009. doi:10.1007/978-1-84882-310-5.

9 Jianzhong Du and Joseph Y-T Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989. doi:10.1137/0402042.

10 Franziska Eberle, Felix Hommelsheim, Malin Rau, and Stefan Walzer. A tight (3/2 + ∈)-approximation algorithm for demand strip packing. In *SODA*, pages 641–699. SIAM, 2025. doi:10.1137/1.9781611978322.20.

11 Dror G. Feitelson and Larry Rudolph. Towards convergence in job schedulers for parallel supercomputers. In *JSSPP*, volume 1162 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 1996. doi:10.1007/BFb0022284.

12 Dimitris Fotakis, Jannik Matuschke, and Orestis Papadigenopoulos. Malleable scheduling beyond identical machines. In *APPROX-RANDOM*, volume 145 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.17.

13 Dimitris Fotakis, Jannik Matuschke, and Orestis Papadigenopoulos. Assigning and scheduling generalized malleable jobs under subadditive or submodular processing speeds. *Oper. Res.*, 73(3):1598–1614, 2025. doi:10.1287/opre.2022.0168.

14 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *FOCS*, pages 260–271. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.32.

15 Bernhard Sebastian Germann, Klaus Jansen, Felix Ohnesorge, and Malte Tutas. 3/2-dual approximation for cpu/gpu scheduling. In *22nd International Symposium on Experimental Algorithms (SEA 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SEA.2024.13.

16 Kilian Grage, Klaus Jansen, and Felix Ohnesorge. Improved algorithms for monotone moldable job scheduling using compression and convolution. In *Euro-Par*, volume 14100 of *Lecture Notes in Computer Science*, pages 503–517. Springer, 2023. doi:10.1007/978-3-031-39698-4_34.

17 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A (5/3 + ϵ)-approximation for strip packing. In *WADS*, volume 6844 of *Lecture Notes in Computer Science*, pages 475–487. Springer, 2011. doi:10.1007/978-3-642-22300-6_40.

18 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

19 Klaus Jansen and Felix Land. Scheduling monotone moldable jobs in linear time. In *2018 IEEE International Parallel and Distributed Processing Symposium, (IPDPS 2018)*, pages 172–181. IEEE Computer Society, 2018. doi:10.1109/IPDPS.2018.00027.

20 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM J. Discret. Math.*, 30(1):343–366, 2016. doi:10.1137/140952636.

21 Klaus Jansen and Lorant Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. In *Proceedings of the tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 490–498. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314870.

22 Klaus Jansen and Malin Rau. Closing the gap for pseudo-polynomial strip packing. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *LIPIcs*, pages 62:1–62:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.62.

23 Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 234–245. Springer, 2008. doi:10.1007/978-3-540-70575-8_20.

24 Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM J. Comput.*, 39(8):3571–3615, 2010. doi:10.1137/080736491.

25 Berit Johannes. Scheduling parallel jobs to minimize the makespan. *J. Sched.*, 9(5):433–452, 2006. doi:10.1007/s10951-006-8497-6.

26 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi:10.1007/978-3-540-24777-7.

27 Walter Ludwig and Prasoon Tiwari. Scheduling malleable and nonmalleable parallel tasks. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA 1994)*, pages 167–176. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314491.

28 Grégory Mounié, Christophe Rapine, and Denis Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of the Eleventh Annual Symposium on Parallel Algorithms and Architectures (SPAA 1999)*, pages 23–32. ACM, 1999. doi:10.1145/305619.305622.

29 Gregory Mounie, Christophe Rapine, and Denis Trystram. A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM J. Comput.*, 37(2):401–412, 2007. doi:10.1137/S0097539701385995.

30 John Turek, Joel L Wolf, and Philip S Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures (SPAA 1992)*, pages 323–332, 1992. doi:10.1145/140901.141909.

31 Fangfang Wu, Xiandong Zhang, and Bo Chen. An improved approximation algorithm for scheduling monotonic moldable tasks. *Eur. J. Oper. Res.*, 306(2):567–578, 2023. doi:10.1016/j.ejor.2022.08.034.