



Many Shared Resources Scheduling Problem

Improvements and Implementations of Approximation Algorithms and Heuristics

Schemaläggning med delade resurser

Examensarbete för kandidatexamen i matematik vid Göteborgs universitet

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Hampus Kihl

Olle Björkenbacke

Olle Lapidus

Many Shared Resources Scheduling Problem

Improvements and Implementations of Approximation Algorithms and Heuristics

Examensarbete för kandidatexamen i matematik inom Matematikprogrammet, inriktning Tillämpad matematik, vid Göteborgs universitet

Olle Björkenbacke

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers

Hampus Kihl Olle Lapidus

Handledare: Malin Rau

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2025

Förord

Bidragsrapport

Hampus Kihl - Har introducerat notation för MSRS algoritmerna och omformulerat greedy-algoritmen, 5/3-algoritmen och 3/2-algoritmen till pseudokod, samt gjort ett bevis för 5/3-algoritmen. Presenterat partitionerna av klasser till algoritmerna som lemma med pseudokod och bevis. Utvecklat några fungerande steg för en 4/3-algoritm för klasser med stora jobb. Detta kan läsas i sektion 1.1, 1.3, 1.4, 2 till 2.3 (exklusive lemma 2.6 till 2.8 och deras respektive bevis), samt sektion 2.5.

Olle Björkenbacke - Har utvecklat en ny 4/3-approximationsalgoritm för instanser med vissa specifika förutsättningar, samt gjort efterforskning/hämtat inspiration för den här processen. Skrivit tillhörande sektion 2.4 med undantag för lemmorna 2.11 och 2.12. Även skrivit sektion 1.2 om relaterade arbeten samt stor del av den populärvetenskapliga presentationen.

Olle Lapidus - Har jobbat med och skrivit delarna om optimeringar och simulering, sektion 3 och 4. Har skrivit all kod till projektet, några tusen rader kod i Python och C++, med implementationer av 5/3- och 3/2-algoritmerna samt simulated annealing och andra optimeringar. Har skrivit stora delar av introduktionen, och också formulerat samt bevisat flera av de lemmorna som ligger i algoritmdelarna, bland annat Lemman 2.6, 2.11 och 2.12. Har funderat mycket på algoritmer i andra spår än de som i slutändan kom med i rapporten, algoritmer som mer följde linjär-programmering spåret från [1].

Ovan framgår vem som huvudsakligen gjort vad i arbetet. Utöver det har alla tillsammans arbetat med att finslipa och bearbeta rapportens alla delar, samt funderat över nya algoritmer och alternativa metoder. Under arbetets gång har en loggbok förts över hur mycket tid gruppsmedlemmarna lagt ner, samt vad som arbetats på under olika veckor.

Användande av AI

Generativ AI har använts i begränsad omfattning och med eftertanke. Den har främst nyttjats för att generera enklare kod, där eventuella fel är lättidentifierade, exempelvis för att skapa bilder och diagram. AI har även använts i mindre delar av skrivprocessen, såsom vid formatering av tabeller i LaTeX. Användningen har skett i situationer där det bedömts kunna spara tid utan att kompromissa med rapportens eller arbetets kvalitet. Dessutom har AI delvis använts för att omformulera just detta stycke om AI-användning.

Populärvetenskaplig presentation

Att lägga scheman är ett vanligt problem som dyker upp i många olika sammanhang. Det kan handla om allt från att någon bestämmer tider och salar för föreläsningar i en kurs, till att en satellit ska skicka observationsdata tillbaka till jorden och måste schemalägga vilket paket som ska skickas via vilken antenn och när. När man ska lägga ett schema kan det vara olika saker man vill prioritera. Kanske vill du att studenterna i kursen ska ha regelbundna föreläsningar på inte allt för konstiga tider. Men för satelliten kanske det snarare är viktigt att all data skickats så snart som möjligt, så att forskare på jorden kan analysera den.

Det här projektet berör olika algoritmer som används för schemaläggning och specifikt för problem med schemaläggning där flera av objekten som ska schemaläggas använder sig av delade resurser, vilket i detta sammanhang innebär att det finns vissa restriktioner på vilka objekt som kan schemaläggas på överlappande tider. Problemet som studeras kallas Many Shared Resources Scheduling (MSRS) Problem. Projektet är huvudsakligen baserat på Hébrard et al. som formulerade det gällande schemaläggning av satellitdata till olika nedladdningskanaler, samt Deppert et al. som utformade och bevisade några av de algoritmer som vi presenterar i denna rapport.

Schemaläggningen som algoritmerna utför är av allmän natur och kan användas inom flera områden såsom att schemalägga föreläsningar eller det tidigare nämnda satellitproblemet. Det som algoritmerna behöver ta hänsyn till är just de delade resurserna. Detta innebär i praktiken att objekt med samma resurs inte kan bearbetas samtidigt. Algoritmerna som utvecklas ska ta in 'indata' som består av så kallade 'jobb' som ska schemaläggas. Dessa jobb har varsin tidsåtgång, vilket kan vara ett godtyckligt heltal, och en specifik resurs som jobbet hör till. Jobb som hör till samma resurs kan inte bearbetas samtidigt. Till indatan anges också ett antal maskiner. Dessa maskiner kan vardera bearbeta ett jobb åt gången så länge de inte använder en delad resurs samtidigt. Vi kan göra liknelsen med ett föreläsningsschema på valfritt universitet. Jobben representerar själva föreläsningen, resursen som används kan liknas med föreläsaren, som endast kan vara på en föreläsning åt gången och maskinerna kan liknas med salarna.

I de algoritmer vi studerar, är målet alltid att minimera tiden då det sista jobbet har bearbetats färdigt. Denna tid kallar vi för 'makespan'. Det går alltid att hitta ett schema som har minimala möjliga makespan, men det kan kräva orimligt mycket processeringstid. Vi fokuserar på att hitta snabba algoritmer, som vi kan bevisa kommer att hitta ett schema som är åtminstone en given faktor från det optimala schemat. Det finns sedan tidigare snabba algoritmer som hittar ett schema som är som mest 50% sämre än det optimala, men det är en obesvarad fråga hur liten man kan göra denna faktor, och det är denna fråga vi strävar efter att besvara.

Sammandrag

Schemaläggning med många delade resurser är ett algoritmiskt problem där ett antal jobb ska schemaläggas på identiska maskiner. Varje maskin kan endast bearbeta ett jobb åt gången. Varje jobb hör till en specifik resurs, och två jobb som delar samma resurs kan inte behandlas parallellt. Att minimera makespan, vilket är tiden då det sista jobbet avslutas, är ett NP-svårt problem. Istället för att söka den optimala lösningen undersöker vi flera approximationsalgoritmer som på polynomisk tid hittar ett schema vars makespan högst överstiger den optimala makespan med en viss faktor. Vi presenterar omformuleringar av $3/2$ - och $5/3$ -algoritmerna som först introducerades av Deppert et al. [1], samt en ny $4/3$ -algoritm som fungerar under vissa förenklade antaganden om uppsättningen av jobb. Denna nya $4/3$ -algoritm funkar om det inte finns vissa typer av stora jobb, och vi presenterar även idéer för hur det skulle kunna gå att få algoritmen att fungera i det allmänna fallet. Dessutom implementerar vi flera algoritmer och vissa optimeringar och heuristiker, och jämför deras prestanda på slumpmässigt genererad data. Simuleringarna visar att även om approximationsalgoritmerna som diskuteras i tidigare avsnitt är användbara för att bevisa vissa egenskaper hos problemet, verkar heuristiska algoritmer vara bättre lämpade för verkliga tillämpningar.

Abstract

In the many shared resources problem, a set of jobs are to be scheduled on some identical machines, such that each machine can process one job at a time. Each job is associated with one resource, and two jobs that share the same resource cannot be processed in parallel. Minimizing the makespan, the time when the last job finishes, is an NP-hard problem. Instead of the optimal solution, we consider several approximation algorithms that in polynomial time find a schedule whose makespan within a factor of the optimal. We provide reformulations of the $3/2$ - and $5/3$ -algorithms first introduced by Deppert et al. [1], as well as a new $4/3$ -algorithm which works under some simplifying assumptions on the set of jobs. The $4/3$ -algorithm works when there are no big jobs of certain types, and some ideas for how to make the algorithm work in the general case are also provided. Furthermore, we implement several algorithms along with some optimizations and heuristics, and compare their performance on randomly generated data. From the simulations it is evident that while the type of approximation algorithms discussed in the earlier sections are useful for proving certain properties about the problem, the heuristic algorithms seem better suited for real-life scenarios.

Contents

1	Introduction	1
1.1	Many Shared Resources Scheduling	1
1.2	Previous and related work	1
1.3	Mathematical notation	2
1.4	Algorithmic notation	2
1.5	Overview	2
2	Algorithms	3
2.1	The greedy algorithm for scheduling	3
2.2	5/3-Algorithm	3
2.3	3/2-Algorithm	5
2.4	A 4/3-algorithm without huge jobs	9
2.5	Working steps for a 4/3-algorithm with huge jobs	14
3	Optimizations	15
3.1	ENQUEUE	15
3.2	Downshifting	16
3.3	Simulated Annealing	16
4	Simulations	17
4.1	Method	17
4.1.1	Simulated optimizations	18
4.1.2	Test data	18
4.2	Results	18
4.3	Discussion of simulation results	18

1 Introduction

Consider the problem of sending data via satellite back to earth from a space exploration mission. Imagine that you have data packets of different sizes that you want to send across m antennas, and some data packets stem from the same source, which means they cannot be transmitted at the same time. Also, each data packet must be sent in full, and its size determines how long it takes for an antenna to send it. The antennas are identical to each other and can each send one packet at a time. In order to ensure the data reaches your lab back on earth as soon as possible, it is interesting for you to assign each data packet to a time and an antenna, such that the last data packet to arrive does so in as short a time as possible.

This is an example of the many shared resources scheduling (MSRS) problem. In this problem, a set of identical machines is given upon which some jobs of varying lengths and classes are to be scheduled, under the restriction that no jobs of the same class can be scheduled at the same time, and every machine can process one job simultaneously. Minimizing the makespan, the time when the last job finishes processing, is NP-hard, meaning that unless $P=NP$ no polynomial time algorithm exists which finds an optimal schedule. However, there exist approximation algorithms which find a schedule that can be shown to lie within certain bounds of the optimal. An α -approximation algorithm is an algorithm that finds a schedule whose makespan is guaranteed to be at most α times larger than the optimal. We describe several such algorithms, and also test some of them in practice.

1.1 Many Shared Resources Scheduling

In the **Many Shared Resources Scheduling (MSRS)** problem, the goal is to schedule jobs on machines, under the constraint that each machine can process only one job at a time, and that jobs belonging to the same class must not be scheduled simultaneously. We are given a set \mathcal{J} of n jobs, a set C of classes, and m identical machines. Each job j has a processing time $p(j) \in \mathbb{N}_{\geq 0}$, and the classes form a partition of the jobs, such that each job is associated with exactly one class. Each class $c \in C$ represents a set of jobs that share a resource. A **schedule** is defined as a tuple (σ, t) , where: $\sigma : \mathcal{J} \rightarrow \{1, \dots, m\}$ and $t : \mathcal{J} \rightarrow \mathbb{N}_{\geq 0}$. The schedule is **feasible** if the following conditions hold:

- **Machine constraint:** No two jobs overlap on the same machine, i.e., if two jobs j and j' are assigned to the same machine ($\sigma(j) = \sigma(j')$), then their processing intervals $[t(j), t(j) + p(j))$ and $[t(j'), t(j') + p(j'))$ must not intersect.
- **Resource constraint:** No two jobs from the same class are processed simultaneously, meaning that if $j, j' \in c$ for some $c \in C$, then their processing intervals must not overlap.

The objective is to find a schedule that minimizes the makespan C_{\max} , which is defined as $\max_{j \in \mathcal{J}} t(j) + p(j)$. A lower bound on the makespan is a lower estimate of the schedule's processing time such that no optimal makespan can be smaller than any of these estimates.

1.2 Previous and related work

The subject of scheduling problems with additional resources has been studied extensively. Hebrard et al. [2] introduced this specific MSRS problem that originally relates to scheduling download plans for Earth observation satellites in 2016. They gave a $(2m/(m+1))$ -approximation algorithm for instances with m machines, including the ENQUEUE subroutine, along with various heuristic methods. This result was later improved by Deppert et al. in [1] with the $5/3$ -algorithm and the $3/2$ -algorithm which improve upon the previously known approximation for 6 or more and 4 or more machines, respectively. Deppert et al. also presented an efficient polynomial time approximation scheme (EPTAS) for a variant of the problem with a constant number of machines using N-fold Integer Programs.

Our problem, MSRS, is typically denoted by $P|res-111|C_{max}$ using the classification of scheduling problems introduced in [3], the so-called 3-field notation. This problem is a generalization of scheduling problems on identical parallel machines, denoted $P||C_{max}$ in the 3-field notation. An

approximation algorithm for the sub-problem of scheduling on identical parallel machines without additional resources was first presented in 1969, by Graham [4], who developed the longest processing time first (LPT) algorithm, with an approximation ratio of $4/3$. Later, this result was improved by Coffman et al. [5], who achieved a $13/11$ approximation with the MULTIFIT algorithm. The best currently known result for this sub-problem is a $(1 + \varepsilon)$ -approximation algorithm, as shown by Berndt et al. in [6].

MSRS can be viewed as a particular case of scheduling with conflicts. This type of problem was discussed by Baker and Coffman [7] for unit processing times, and includes a conflict graph where the jobs are represented by vertices that are connected by a set of edges, where two jobs that are connected by an edge can not be processed simultaneously. This relates to MSRS problems, since the classes can be seen as disjoint cliques in an especially simple conflict graph. Scheduling with conflicts is solvable in polynomial time for co-graphs with unit-size jobs and a constant number of machines [8]. Various other instances of the problem have been studied, for example, for two machines and job sizes in $\{1, 2, 3\}$ there exists a $4/3$ -approximation algorithm, provided by Even et al. [9].

1.3 Mathematical notation

For a single job i , $p(i)$ is used to denote its processing time, and for any sets of classes or jobs X , $p(X) = \sum_{x \in X} p(x)$ denotes the total processing time. The total load of jobs on machine i in the current schedule is denoted by $p_m(i)$. For a set of machines M , the total load is given by $P(M) = \sum_{i \in M} p_m(i)$. For any set $X \in \{\mathcal{J}, \mathcal{C}\}$, a relation $* \in \{<, \leq, \geq, >\}$, and a number λ , we define $X_{*\lambda} = \{x \in X | p(x) * \lambda\}$ and for a given interval v let $X_v = \{x \in X | p(x) \in v\}$. For example, the notation $\mathcal{J}_{>1/2}$ represents the set of all jobs satisfying $p(j) > 1/2$ for $j \in \mathcal{J}$. In the algorithm description, some jobs may be scheduled while others remain unscheduled. For any set X we will use the notation \bar{X} to indicate the residual set. For instance, the set of unscheduled jobs is denoted by $\bar{\mathcal{J}}$, and the set of unscheduled classes from $\mathcal{C}_{(1/2, 2/3]}$ is denoted by $\bar{\mathcal{C}}_{(1/2, 2/3]}$. The machines with load less than 1 are denoted by $\bar{M} = \{i \in M | p_m(i) < 1\}$, and the machines which also contain at least one huge job are denoted by $\bar{M}_H = \{i \in M | p_m(i) < 1 \wedge (\exists c \in \sigma(i), c \in \mathcal{C}_H)\}$. Unused machines, i.e. machines with no jobs placed on them are denoted by \bar{M}_u .

1.4 Algorithmic notation

In the algorithms described in this work we use the following notation. We denote by $\text{Schedule}(x, i, t)$ the procedure to schedule x , which is a job, class or partition of a class, on machine i , starting at time t . Each subsequent job starts immediately after the previous one finishes. We denote by $\text{Reschedule}(i, t)$ the procedure to reschedule all the jobs currently assigned on machine i , such that the first job or class starts at time t , and all subsequent ones are shifted accordingly to maintain their order. The final schedule is represented by two functions (σ, t) , where $\sigma : \mathcal{J} \rightarrow \{1, \dots, m\}$ and $t : \mathcal{J} \rightarrow \mathbb{N}_{\geq 0}$, and each entry corresponds to a job, its assigned machine, and start time. The functions are sets of ordered pairs that the algorithm constructs using the scheduled jobs. Whenever Schedule or Reschedule , schedules or reschedules classes or jobs, the schedule (σ, t) is updated accordingly. The notation $x \leftarrow y$ indicates that variable x is set to the value y . For example, in the process of partitioning classes, $\hat{c} \leftarrow \{j\}$ signifies that job j forms the partitioned subset \hat{c} . The operation that selects an element $c \in \bar{\mathcal{C}}$ and simultaneously removes c from the set $\bar{\mathcal{C}}$ is denoted by $\text{pop}(\bar{\mathcal{C}})$. The operation that selects an element $c \in \bar{\mathcal{C}}_a$ if $\bar{\mathcal{C}}_a$ is non-empty; otherwise, selects $c \in \bar{\mathcal{C}}_b$ if $\bar{\mathcal{C}}_b$ is non-empty, is denoted by $\text{coalesce}(\{\text{pop}(\bar{\mathcal{C}}_a), \text{pop}(\bar{\mathcal{C}}_b)\})$. This function is only ever called if at least one of $\bar{\mathcal{C}}_a$ or $\bar{\mathcal{C}}_b$ is non-empty.

1.5 Overview

This paper is heavily based on material from [1]. In section 2 we describe several approximation algorithms. Both the $3/2$ -algorithm and the $5/3$ -algorithm come from [1], but have been reformulated into pseudocode, and some proofs which were left out of the original paper have been added. We provide the first steps of a new $4/3$ -algorithm which works under some simplifying assumptions

on the input, as well as some ideas on how to construct a $4/3$ -algorithm which works in general. In section 3 some optimizations are described, which do not improve the theoretical bound, but still can be useful in practical implementations. Finally, in section 4 algorithms and optimizations are tested and compared on randomly generated data.

2 Algorithms

We begin by introducing known algorithms [1] for scheduling with many shared resources, followed by our own contributions, including pseudocode, proofs, and new scheduling techniques. Most (MSRS) algorithms rely on partitioning classes that satisfy certain size conditions. These class partitions are described in detail within each algorithm's description. We use a lower bound on the makespan as an approximation of the optimal scheduling time. This bound is formalized in Lemma 2.1 and subsequently proven.

Lemma 2.1. *Let $T := \max\{\frac{1}{m}p(\mathcal{J}), \max_{c \in C} p(c), p(j_m) + p(j_{m+1})\}$, where j_m and j_{m+1} are the jobs with the m -th and $(m+1)$ -th largest processing times. Then T is a lower bound of the makespan.*

Proof. The term $\frac{1}{m}p(\mathcal{J})$ is a lower estimate because the makespan cannot be less than the processing times of all jobs divided by the number of machines. Additionally, the makespan must be at least the processing time of the largest class, since jobs from the same class cannot be scheduled simultaneously. The sum of $p(j_m) + p(j_{m+1})$ is also a lower bound of the makespan, because two of the first $m+1$ jobs need to be scheduled on the same machine, and $p(j_m) + p(j_{m+1})$ is a lower bound of the processing time of those two jobs. Hence T is a lower bound on the makespan of the optimal schedule. \square

2.1 The greedy algorithm for scheduling

The greedy algorithm schedules the largest unscheduled job on the machine with the least load, continuing this process until no classes remain. This procedure is presented as pseudocode in Appendix 4.3.

2.2 5/3-Algorithm

This $5/3$ -approximation algorithm is a reformulation of the one found in [1]. Each job in the $5/3$ -algorithm is scaled by $1/T$ such that $p(j) \in (0, 1]$ for each job j , where T is the lower bound of the makespan as defined in Lemma 2.1. The $5/3$ -algorithm creates a schedule with a makespan that is at most $\frac{5}{3}T$, therefore it is a $5/3$ -approximation. One way of ensuring that all jobs are scheduled before step $i > m$, is to ensure that the average load on the set of closed machines is at least 1. A machine is considered closed once no further jobs or classes will be assigned to it in subsequent steps. To satisfy the average load condition, machines are closed incrementally in such a way that, at each step, the average load on the set of machines being closed is greater than 1. All classes containing jobs of size $p(j) > 1/2$, $j \in \mathcal{J}$ are denoted by $C_{B^+} := \{c \in C \mid |c \cap \mathcal{J}_{>1/2}| = 1\}$.

Algorithm 1: `split1(c)`

Input : $c \in C_{>2/3} \setminus C_{B^+}$

Output: (\hat{c}, \check{c})

```

1   $c_1 \leftarrow \{\arg \max_{j \in c} p(j)\}$ 
2  while  $p(c_1) \leq 1/3$  do
3  |    $c_1 \leftarrow c_1 \cup \{\arg \max_{j \in c \setminus c_1} p(j)\}$ 
4  return  $(c_1, c \setminus c_1)$ 

```

Lemma 2.2. *The procedure `split1` given a class $c \in C_{>2/3} \setminus C_{B^+}$ partitions c into new sets \hat{c} and $\check{c} = c \setminus \hat{c}$ such that it holds that $1/3 \leq p(\hat{c}) \leq 2/3$ and $p(\check{c}) \leq 2/3$.*

Proof. `split1` aims to partition c into two subsets c_1 and $c_2 = c \setminus c_1$ such that $1/3 \leq p(c_1) \leq 2/3$ and $p(c_2) \leq 2/3$. We consider two possible cases:

Case 1: There exists a job $j \in c$ such that $p(j) > 1/3$. The algorithm defines $c_1 = \{j\}$ in line 1. As $p(j) > 1/3$ and $p(j) \leq 1/2$ because $c \notin C_{B^+}$, the while statement is not true and the algorithm returns $(\{j\}, c \setminus \{j\})$. As $p(j) \in (1/3, 2/3)$, it holds that $1/3 \leq p(c_1) \leq 2/3$ and $p(c_2) = p(c) - p(c_1) \leq 1 - 1/3 = 2/3$.

Case 2: Otherwise all jobs $j \in c$ satisfy $p(j) \leq 1/3$. The algorithm constructs c_1 by greedily adding jobs from $c \setminus c_1$ until $p(c_1) > 1/3$, and then returns $(c_1, c \setminus c_1)$. Since no job has size greater than $1/3$, it follows that when a job is added such that $p(c_1) > 1/3$, the total will satisfy $p(c_1) \leq 1/3 + 1/3 = 2/3$. Thus, it holds that $1/3 \leq p(c_1) \leq 2/3$, and $p(c_2) = p(c) - p(c_1) \leq 1 - 1/3 = 2/3$. \square

Algorithm 2: 5/3-algorithm

```

Input :  $C$ 
Output:  $(\sigma, t)$ 
1   $\sigma \leftarrow \emptyset, t \leftarrow \emptyset$ 
2   $i \leftarrow 1$ 
3  foreach  $c \in C_{B^+}$  do                                     // Step 1
4  |   Schedule( $c, i, 0$ )
5  |    $i \leftarrow i + 1$ 
6   $i \leftarrow 1$ 
7  foreach  $c \in C_{>2/3} \setminus C_{B^+}$  do                       // Step 2
8  |   if  $p_m(i) \leq 5/3 - p(c)$  then
9  |   |   Schedule( $c, i, p_m(i)$ )
10 |   else
11 |   |    $(\hat{c}, \check{c}) \leftarrow \{\text{split1}(c)\}$ 
12 |   |   Schedule( $\hat{c}, i, 5/3 - p(\hat{c})$ ),  $i \leftarrow i + 1$  // machine  $i$  is closed
13 |   |   Reschedule( $i, p(\check{c})$ ), Schedule( $\check{c}, i, 0$ )
14 |   if  $p_m(i) > 1$  then
15 |   |    $i \leftarrow i + 1$  // machine  $i$  is closed
16 |   foreach  $c \in C_{<2/3} \setminus C_{B^+}$  do                       // Step 3
17 |   |   Schedule( $c, i, p_m(i)$ )
18 |   |   if  $p_m(i) > 1$  then
19 |   |   |    $i \leftarrow i + 1$  // machine  $i$  is closed
20 return  $(\sigma, t)$ 

```

Lemma 2.3. *Algorithm 2 is a 5/3-approximation.*

Proof. To prove that the constructed schedule is a 5/3-approximation, we must verify the following conditions:

1. Only available machines are used, i.e. $i \leq m$ for each call to `Schedule`.
2. (Load constraint) No machine exceeds a load of $5/3$.
3. (Machine constraint) No two jobs overlap on the same machine.
4. (Resource constraint) No two jobs from the same class are scheduled simultaneously.

We analyze the schedule incrementally through its three steps:

By scaling the algorithm with T , as described in Lemma 2.1 and ensuring that the average load of all closed machines is greater than 1, we ensure that all jobs are scheduled on machines satisfying $i \leq m$. Throughout the algorithm, condition 3 is trivially satisfied because classes or class partitions are scheduled sequentially on each machine. Condition 4 is also satisfied in all steps by design.

Step 1: In this step, the algorithm assigns each class $c \in C_{B^+}$ to an individual machine. Note that each of these classes contains a job with processing time $p(j) > 1/2$. Hence in the optimal schedule no two such jobs are scheduled on the same machine, thus $i \leq m$. Since each class is placed on a separate machine, the machine, load, and resource constraints are trivially satisfied.

Step 2: In this step we schedule classes $c \in C_{>2/3} \setminus C_{B^+}$. After Step 1, each machine either has one class $c \in C_B^+$ scheduled on it or is empty. Note that in line 8, it always holds that $p_m(i) \leq 1$. This holds when we enter the loop for the first time, the first machine then contains at most one class and each class has a load of at most 1. For the following loops, this holds as we go to the next machine in line 15 if the load of the current machine is larger than 1.

Since no machine will be closed unless $p_m(i) > 1$ and c can be partitioned using `split1` into parts \check{c} and \hat{c} with load less than $2/3$, the algorithm can always add classes or partitions of classes until the load of the machine is greater than 1, without violating the load constraint. Here are the possible cases when scheduling in step 2:

- *Case 1:* Line 8 is true, which means that $p_m(i) \leq 5/3 - p(c)$. This ensures that the Load constraint is satisfied when scheduling class c on machine i . If the load of machine i exceeds 1, then line 14 is true and the algorithm proceeds to machine $i + 1$. Since all closed machines have load larger than 1, it follows that $i + 1 < m$. otherwise, the algorithm continues to schedule on machine i .
- *Case 2:* If $p_m(i) \geq 5/3 - p(c)$ is true, then line 8 is false and the algorithm goes to line 10. Note that $p_m(i) \leq 1$ as mentioned earlier, and that $p_m(i) \geq 2/3$, as otherwise $p_m(i) + p(c) < 5/3$. Then c is partitioned using `split1` into parts with load less than $2/3$. Scheduling \hat{c} on i , the load of i is bounded by $1 + 2/3 = 5/3$, hence satisfying load constraint. The load of the machine is at least 1, since $p(\hat{c}) \geq 1/3$ and therefore line 14 is true. Then \check{c} is scheduled on the next machine ($i + 1$). Since all closed machines have load larger than 1, it follows that $i + 1 < m$.

Step 3: Finally, the remaining classes $c \in C_{\leq 2/3} \setminus C_B^+$ are greedily added to a machine until the load of the machine is at least 1 or all remaining classes have been scheduled. When the algorithm enters line 16, we have $p_m(i) \leq 1$. As the load of i is at most 1 before scheduling c and $p(c) \leq 2/3$, the load of i after scheduling is at most $5/3$. In the next line the algorithm checks whether the load of machine i exceeds 1 and closes the machine if true. Since the average load of the closed machine is greater than 1, it follows that $i + 1 < m$ and the algorithm proceeds to machine $i + 1$. If line 18 is false, the algorithm continues to schedule classes on i until $p_m(i) > 1$.

Since no machine exceeds load of $5/3$ at any step, the load constraint is satisfied. Moreover, as no machine at any step is closed before its load exceeds 1, all classes are scheduled on machines satisfying $i \leq m$. \square

2.3 3/2-Algorithm

The 3/2-algorithm constructs a schedule with a makespan that is at most $\frac{3}{2}T$, where T is defined as in Lemma 2.1. The approximation algorithm is divided into two different cases: one in which there exists no job with size greater than $\frac{3}{4}T$, and one in which at least one job is greater than $\frac{3}{4}T$. The algorithm is a reformulation of the one found in [1].

Each job is scaled by $1/T$ such that $p(j) \in (0, 1]$. The jobs are classified as follows:

- **Huge:** $\mathcal{J}_H = \{j \in \mathcal{J} \mid p(j) > 3/4\}$
- **Big:** $\mathcal{J}_B = \{j \in \mathcal{J} \mid p(j) \in (1/2, 3/4]\}$
- **Medium:** $\mathcal{J}_M = \{j \in \mathcal{J} \mid p(j) \in (1/4, 1/2]\}$
- **Small:** $\mathcal{J}_S = \mathcal{J}_{\leq 1/4}$

Class subsets are defined as:

- **Containing a huge job:** $C_H = \{c \in C \mid |\mathcal{J}_H \cap c| = 1\}$
- **Containing a big job:** $C_B = \{c \in C \mid |\mathcal{J}_B \cap c| = 1\}$

The 3/2-approximation algorithm for the case without classes containing huge jobs is given in pseudocode as Algorithm 8. This is a sub-algorithm for the general case presented in Algorithm 15 as shown in Appendix 4.3. For Algorithm 8, we also introduce sub-algorithms Algorithm 5, Algorithm 6 and Algorithm 7. The 3/2-algorithm relies on partitioning classes that satisfy certain size conditions, as stated in Lemma 2.4 and Lemma 2.5. We will prove these lemmas and then present

the corresponding procedures as pseudocode in Algorithm 3 and Algorithm 4.

Algorithm 3: split2(c)

Input : $c \in C_{\geq 3/4} \setminus C_H$
Output: (\hat{c}, \check{c})

- 1 $c_1 \leftarrow \{\arg \max_{j \in c} p(j)\}$
- 2 **while** $p(c_1) \leq 1/4$ **do**
- 3 | $c_1 \leftarrow c_1 \cup \{\arg \max_{j \in c \setminus c_1} p(j)\}$
- 4 **if** $p(c_1) \geq p(c \setminus c_1)$ **then**
- 5 | **return** $(c_1, c \setminus c_1)$
- 6 **return** $(c \setminus c_1, c_1)$

Lemma 2.4. *The procedure `split2` given a class $c \in C_{\geq 3/4}$ and where $\max_{j \in c} p(j) \leq 3/4$, partitions c into new sets \check{c} and \hat{c} such that $p(\check{c}) \leq 1/2$, $p(\hat{c}) \leq 3/4$ and $p(\check{c}) \leq p(\hat{c})$.*

Proof. `split2` aims to partition c into two subsets c_1 and $c_2 = c \setminus c_1$ such that it holds that, $p(c_2) \leq 1/2$, $p(c_1) \leq 3/4$ and $p(c_2) \leq p(c_1)$. We consider two possible cases:

Case 1: there exists a job $j \in c$ such that $p(j) > 1/4$. The algorithm defines $c_1 = \{j\}$ in line 1. If $p(j) \geq p(c \setminus \{j\})$ then it holds that $p(c \setminus \{j\}) \leq 1/2$ and the algorithm return $(\{j\}, c \setminus \{j\})$. Else $p(j) \leq p(c \setminus \{j\})$ and then it holds that $p(j) \leq 1/2$, $p(c \setminus \{j\}) \leq 3/4$ and the algorithm return $(c \setminus \{j\}, \{j\})$.

Case 2: Otherwise all jobs $j \in c$ satisfy $p(j) \leq 1/4$. The algorithm constructs c_1 by greedily adding jobs from $c \setminus c_1$ until $p(c_1) \geq 1/4$. If $p(c_1) \geq p(c \setminus c_1)$ then it holds that $p(c \setminus c_1) \leq 1/2$ and the algorithm returns $(c_1, c \setminus c_1)$. Else $p(c_1) \leq p(c \setminus c_1)$ and then it holds that $p(c_1) \leq 1/2$, $p(c \setminus c_1) \leq 3/4$ and the algorithm return $(c \setminus c_1, c_1)$. □

Algorithm 4: split3(c)

Input : $c \in C_{(1/2, 3/4)} \setminus C_B$
Output: (\hat{c}, \check{c})

- 1 $c_1 \leftarrow \{\arg \max_{j \in c} p(j)\}$
- 2 **while** $p(c_1) \leq 1/4$ **do**
- 3 | $c_1 \leftarrow c_1 \cup \{\arg \max_{j \in c \setminus c_1} p(j)\}$
- 4 **if** $p(c_1) \geq p(c \setminus c_1)$ **then**
- 5 | **return** $(c_1, c \setminus c_1)$
- 6 **return** $(c \setminus c_1, c_1)$

Lemma 2.5. *The procedure `split3` given a class $c \in C_{(1/2, 3/4)}$ and $\max_{j \in c} p(j) \leq 1/2$ partitions c into \check{c} and \hat{c} such that $p(\check{c}) \leq p(\hat{c}) \leq 1/2$ and $p(\hat{c}) > 1/4$.*

Proof. `split3` aims to partition c into two subsets c_1 and $c_2 = c \setminus c_1$ such that it holds that, $p(\check{c}) \leq p(\hat{c}) \leq 1/2$ and $p(\hat{c}) > 1/4$. We consider two cases:

Case 1: there exists a job $j \in c$ such that $p(j) > 1/4$ and by definition all jobs $j \in c$ satisfies $p(j) \leq 1/2$. The algorithm defines $c_1 = \{j\}$ in line 1. If $p(j) \geq p(c \setminus \{j\})$ then the algorithm return $(\{j\}, c \setminus \{j\})$. Else $p(j) \leq p(c \setminus \{j\})$ and the algorithm return $(c \setminus \{j\}, \{j\})$.

Case 2: Otherwise all jobs $j \in c$ satisfy $p(j) \leq 1/4$. The algorithm constructs c_1 by greedily adding jobs from $c \setminus c_1$ until $p(c_1) \geq 1/4$. since no job in c is greater than $1/4$, we can greedily add jobs to c_1 until $p(c_1) \geq 1/4$ without violating $p(c_1) \leq 1/2$ If $p(c_1) \geq p(c \setminus c_1)$ then the algorithm return $(c_1, c \setminus c_1)$. Else $p(c_1) \leq p(c \setminus c_1)$ and then it holds that $p(c \setminus c_1) \leq 1/2$ since $p(c) \leq 3/4$ and $p(c_1) \geq 1/4$. The algorithm then return $(c \setminus c_1, c_1)$. □

Algorithm 5: Four large classes

Input : $(c_1, c_2, c_3, c_4 \in C_{>3/4}, i, \sigma)$ **Output:** (σ, t)

```
1   $(\hat{c}_1, \check{c}_1) \leftarrow \{\text{split2}(c_1)\}, (\hat{c}_2, \check{c}_2) \leftarrow \{\text{split2}(c_2)\}$ 
2   $\text{Schedule}(\hat{c}_1, i, 0), \text{Schedule}(\hat{c}_2, i, 3/2 - p(\hat{c}_2))$ 
3   $\text{Schedule}(c_3, i + 1, 0), \text{Schedule}(\check{c}_1, i + 1, 3/2 - p(\check{c}_1))$ 
4   $\text{Schedule}(\check{c}_2, i + 2, 0), \text{Schedule}(c_4, i + 2, 3/2 - p(c_4))$ 
5  return  $(\sigma, t)$ 
```

Lemma 2.6. *The procedure in Algorithm 5 creates a feasible schedule with makespan $3/2$ for four classes $c_1, c_2, c_3, c_4 \in C_{(>3/4)}$ on three machines. The average load on the machines is at least 1.*

Proof. On machine i , the total load is given by $p(\hat{c}_1) + p(\hat{c}_2) \leq 1$. On machine $i + 1$, the load is given by $p(c_3) + p(\check{c}_1) \leq \frac{3}{2}$, and the scheduling time of \check{c}_1 is disjoint from the scheduling time of \hat{c}_1 . Lastly, for the third machine the total duration is also bounded by $\frac{3}{2}$, and \check{c}_2 is scheduled disjoint in time from \hat{c}_2 . The total time is given from $p(c_1) + p(c_2) + p(c_3) + p(c_4) \geq 3$, so the average load is at least 1. \square

Algorithm 6: Two classes

Input : $(c_1, c_2 \in \overline{C}_{\geq 1/2}, i, \sigma)$ **Output:** (σ, t)

```
1  if  $p(c_2) \leq 3/4$  then
2  |   if  $p(c_1) + p(c_2) \leq 3/2$  then
3  | |    $\text{Schedule}(c_1, i, 0), \text{Schedule}(c_2, i, 3/2 - p(c_2)), i \leftarrow i + 1$ 
4  | |   if  $p(c_1) + p(c_2) > 3/2$  then
5  | | |    $(\hat{c}_1, \check{c}_1) \leftarrow \{\text{split2}(c_1)\}$ 
6  | | |    $\text{Schedule}(c_2, i, 0), \text{Schedule}(\hat{c}_1, i, 3/2 - p(c_2)), i \leftarrow i + 1$ 
7  | | |    $\text{Schedule}(\check{c}_1, i, 0)$ 
8  |   if  $p(c_2) \geq 3/4$  then
9  | |    $(\hat{c}_1, \check{c}_1) \leftarrow \{\text{split2}(c_1)\}$ 
10 | |   if  $p(\hat{c}_1) + p(\hat{c}_2) \leq 1$  then
11 | | |    $\text{Schedule}(c_2, i, 0), \text{Schedule}(\hat{c}_1, i, 3/2 - p(\hat{c}_1)), i \leftarrow i + 1$ 
12 | | |    $\text{Schedule}(\check{c}_1, i, 0)$ 
13 | | |   if  $p(\hat{c}_1) + p(\hat{c}_2) > 1$  then
14 | | | |    $(\hat{c}_2, \check{c}_2) \leftarrow \{\text{split2}(c_2)\}$ 
15 | | | |    $\text{Schedule}(\hat{c}_1, i, 0), \text{Schedule}(\hat{c}_2, i, 3/2 - p(\hat{c}_2)), i \leftarrow i + 1$ 
16 | | | |    $\text{Schedule}(\check{c}_2, i, 0), \text{Schedule}(\check{c}_1, i, 3/2 - p(\check{c}_1))$ 
17 return  $(\sigma, t)$ 
```

Lemma 2.7. *The procedure in Algorithm 6 schedules two classes $c_1, c_2 \in C_{\{\geq 1/2\}}$, first on one machine such that it has load at least 1, and then possibly some residual jobs at the beginning of a second machine. The partial schedule produced is feasible.*

Proof. In each case where a class c is split into two parts \hat{c} and \check{c} , the parts are scheduled one starting at 0 and the other ending at $3/2$, meaning they are never overlapping. The load on the first machine is, in each of the four cases read from top to bottom, lower bounded by $p(c_1) + p(c_2) \geq 1$, $p(c_2) + p(\hat{c}_1) \geq 1$, $p(c_2) + p(\hat{c}_1) \geq 1$, and finally $p(\hat{c}_1) + p(\hat{c}_2) \geq 1$, which follows from Lemma 2.4. \square

Algorithm 7: Three classes

Input : $(c_1, c_2, c_3 \in \overline{C}_{\geq 3/4}, i, \sigma)$
Output: (σ, t)

```
1   $(\hat{c}_1, \check{c}_1) \leftarrow \{\text{split2}(c_1)\}$ 
2  if  $\exists i \in \{1, 2, 3\} \mid p(\hat{c}_i) \leq 1/2$  then
3  |   Relabel indices such that  $\hat{c}_1 \leq 1/2$ 
4  |    $\text{Schedule}(\hat{c}_1, i, 0), \text{Schedule}(c_2, i, 3/2 - p(c_2)), i \leftarrow i + 1$ 
5  |    $\text{Schedule}(c_3, i, 0), \text{Schedule}(\check{c}_1, i, 3/2 - p(\check{c}_1))$ 
6  if  $\forall i \in \{1, 2, 3\} \mid p(\hat{c}_i) > 1/2$  then
7  |    $(\hat{c}_2, \check{c}_2) \leftarrow \{\text{split2}(c_2)\}$ 
8  |    $\text{Schedule}(\hat{c}_1, i, 0), \text{Schedule}(\hat{c}_2, i, 3/2 - p(\hat{c}_2)), i \leftarrow i + 1$ 
9  |   if  $p(\check{c}_1) + p(\check{c}_2) + p(c_3) \leq 3/2$  then
10 | |    $\text{Schedule}(\check{c}_2, i, 0), \text{Schedule}(c_3, i, p(\check{c}_2)), i \leftarrow i + 1$ 
11 | |    $\text{Schedule}(\check{c}_1, i, 3/2 - p(\check{c}_1))$ 
12 | |   if  $p(\check{c}_1) + p(\check{c}_2) + p(c_3) > 3/2$  then
13 | | |    $\text{Schedule}(c_3, i, 0), \text{Schedule}(\check{c}_1, i, 3/2 - p(\check{c}_1)), i \leftarrow i + 1$ 
14 | | |    $\text{Schedule}(\check{c}_2, i, 0)$ 
15 return  $(\sigma, t)$ 
```

Lemma 2.8. *The procedure described in Algorithm 7, given three classes $c_1, c_2, c_3 \in \overline{C}_{\geq 3/4}$, creates a feasible partial schedule with makespan $3/2$, and each machine except the last considered will have together have average load at least 1.*

Proof. In each case where a class c is split into two parts \hat{c} and \check{c} , the parts are scheduled one starting at 0 and the other ending at $3/2$, meaning they are never overlapping. In the first case, everything is scheduled on two machines, so the average load of those two machines is at least 1 since all classes have $p(c) \geq 3/4$. In the second case, the first machine will have load $p(\hat{c}_1) + p(\hat{c}_2) \geq 1$, and the second machine will have a load of either $p(\check{c}_2) + p(c_3) \geq 5/4$ or $p(c_3) + p(\check{c}_1) \geq 5/4$. In all these cases, the inequality $\text{load} \leq 3/2$ also holds. \square

Algorithm 8: 3/2-algorithm without huge jobs

Input : C
Output: (σ, t)

```
1  foreach  $c \in C_{>3/4}$  do // Step 1
2  |    $(\hat{c}, \check{c}) \leftarrow \{split2(c)\}$ 
3  |    $i = 1$ 
4  |   while  $|\overline{C}_{(1/2,3/4)}| \geq 2$  do // Step 2
5  |   |    $(c_1, c_2) \leftarrow \{\text{pop}(\overline{C}_{(1/2,3/4)}), \text{pop}(\overline{C}_{(1/2,3/4)})\}$ 
6  |   |   Schedule( $c_1, i, 0$ ), Schedule( $c_2, i, 3/2 - p(c_2)$ ),  $i \leftarrow i + 1$ 
7  |   |   while  $|\overline{C}_{\geq 3/4}| \geq 4$  do // Step 3
8  |   |   |    $(c_1, c_2, c_3, c_4) \leftarrow \{\text{pop}(\overline{C}_{\geq 3/4}), \text{pop}(\overline{C}_{\geq 3/4}), \text{pop}(\overline{C}_{\geq 3/4}), \text{pop}(\overline{C}_{\geq 3/4})\}$ 
9  |   |   |    $\sigma \leftarrow \text{Algorithm 5}(c_1, c_2, c_3, c_4, i, \sigma)$ ,  $i \leftarrow i + 3$ 
10 |   |   if  $|\overline{C}_{\geq 3/4}| \geq 2$  and  $|\overline{C}_{(1/2,3/4)}| = 1$  then // Step 4
11 |   |   |    $(c_1, c_2) \leftarrow \{\text{pop}(\overline{C}_{\geq 3/4}), \text{pop}(\overline{C}_{\geq 3/4})\}$ ,  $c_3 \leftarrow \{\text{pop}(\overline{C}_{(1/2,3/4)})\}$ 
12 |   |   |   Schedule( $c_3, i, 0$ ), Schedule( $\hat{c}_1, i, 3/2 - p(\hat{c}_1)$ ),  $i \leftarrow i + 1$ 
13 |   |   |   Schedule( $\check{c}_1, i, 0$ ), Schedule( $c_2, i, 3/2 - p(c_2)$ ),  $i \leftarrow i + 1$ 
14 |   |   if  $|\overline{C}_{\geq 1/2}| \leq 1$  then // Step 5
15 |   |   |    $c \leftarrow \{\text{pop}(\overline{C}_{\geq 1/2})\}$ 
16 |   |   |   Schedule( $c, i, 0$ )
17 |   |   if  $|\overline{C}_{\geq 1/2}| = 2$  then // Step 6
18 |   |   |    $(c_1, c_2) \leftarrow \{\text{pop}(\overline{C}_{\geq 1/2}), \text{pop}(\overline{C}_{\geq 1/2})\}$ , where we know that  $p(c_1) \geq p(c_2)$  and
19 |   |   |    $p(c_1) \geq 3/4$ 
20 |   |   |    $\sigma \leftarrow \text{Algorithm 6}(c_1, c_2, i, \sigma)$ ,  $i \leftarrow i + 1$ 
21 |   |   if  $|\overline{C}_{\geq 1/2}| = 3$  then // Step 7
22 |   |   |    $|\overline{C}_{\geq 1/2}| = |\overline{C}_{\geq 3/4}|$ 
23 |   |   |    $(c_1, c_2, c_3) \leftarrow \{\text{pop}(\overline{C}_{\geq 3/4}), \text{pop}(\overline{C}_{\geq 3/4}), \text{pop}(\overline{C}_{\geq 3/4})\}$ 
24 |   |   |    $\sigma \leftarrow \text{Algorithm 7}(c_1, c_2, c_3, i, \sigma)$ ,  $i \leftarrow i + 1$ 
25 |   GreedyAlgorithm( $\overline{C}_{\leq 1/2}, \overline{M}$ )
26 |   return  $(\sigma, t)$ 
```

2.4 A 4/3-algorithm without huge jobs

In this section, we present the first steps of a 4/3-approximation algorithm. Similarly to the 3/2-algorithm, we will first consider a sub-algorithm for instances without huge jobs, and then describe the first ideas for dealing with huge jobs while also generating a schedule with a makespan of $\frac{4}{3}T$.

Throughout this section, we need to refer to classes containing jobs of at least load $\frac{1}{3}T$, with total load that falls in some interval. As a shorthand for this, for all $a, b \in \mathbb{R}_+$ we define $C'_{[a,b]} := \{c \in C \mid p(c) \in [a, b), \exists j \in c : p(j) > \frac{1}{3}T\}$.

Theorem 2.9. *Let T be a target makespan and I be an instance such that there are no huge jobs ($p(j) > \frac{2}{3}T$), no class can have two jobs with $p(j) > \frac{1}{3}T$ and there are no classes $c \in C'_{(4/9, 1/2]}$. There is then a polynomial-time algorithm that can find a schedule σ for I with makespan $\frac{4}{3}T$ or decide that there is no schedule with makespan T .*

Lemma 2.10. *For any normalized optimal schedule with the corresponding partition of C into sets $C_{\geq 5/6}, C_{[3/4, 5/6)}, C_{[2/3, 3/4)}, C_{[1/2, 2/3)}, C_{[1/3, 1/2)}, C_{< 1/3}$ it holds that*

$$m \geq \lceil \frac{1}{12}(10|C_{\geq 5/6}| + 9|C_{[3/4, 5/6)}| + 8|C_{[2/3, 3/4)}| + 6|C_{[1/2, 2/3)}| + 4|C_{[1/3, 1/2)}|) \rceil.$$

Proof. This inequality follows directly from the fact that for a normalized, optimal schedule, the

total processing time of all jobs must be less or equal to the number of machines. We must have

$$\begin{aligned}
m &\geq \lceil p(\mathcal{J}) \rceil = \lceil p(C) \rceil \\
&= \lceil p(C_{\geq 5/6}) + p(C_{[3/4, 5/6)}) + p(C_{[2/3, 3/4)}) + p(C_{[1/2, 2/3)}) + p(C_{[1/3, 1/2)}) + p(C_{< 1/3}) \rceil \\
&\geq \lceil (\frac{5}{6}|C_{\geq 5/6}| + \frac{3}{4}|C_{[3/4, 5/6)}| + \frac{2}{3}|C_{[2/3, 3/4)}| + \frac{1}{2}|C_{[1/2, 2/3)}| + \frac{1}{3}|C_{[1/3, 1/2)}|) \rceil \\
&= \lceil \frac{1}{12}(10|C_{\geq 5/6}| + 9|C_{[3/4, 5/6)}| + 8|C_{[2/3, 3/4)}| + 6|C_{[1/2, 2/3)}| + 4|C_{[1/3, 1/2)}|) \rceil
\end{aligned}$$

to accommodate for the jobs in a normalized optimal schedule. \square

Given T , we start by scaling the instance by dividing each processing time by T . For the scaled instance we verify that $1 \geq \max\{\frac{1}{m}p(\mathcal{J}), \max_{c \in C} p(c), p(j_m) + p(j_{m+1})\}$ as by Lemma 2.1 otherwise we know that a schedule with makespan T does not exist. Further we verify that

$$m \geq \lceil \frac{1}{12}(10|C_{\geq 5/6}| + 9|C_{[3/4, 5/6)}| + 8|C_{[2/3, 3/4)}| + 6|C_{[1/2, 2/3)}| + 4|C_{[1/3, 1/2)}|) \rceil,$$

and if not we know by Lemma 2.10 that a schedule with makespan T does not exist. These two conditions ensure that

$$m \geq \max \left\{ \lceil \frac{1}{12}(10|C_{\geq 5/6}| + 9|C_{[3/4, 5/6)}| + 8|C_{[2/3, 3/4)}| + 6|C_{[1/2, 2/3)}| + 4|C_{[1/3, 1/2)}|) \rceil, \lceil p(\mathcal{J}) \rceil \right\}.$$

In this algorithm, we use two different partitions of the classes $c \in C_{\geq 5/6}$ as stated in the following two lemmas. Note that both of these partitions can be achieved using a greedy algorithm, where the jobs in the classes are iterated over and assigned to the larger partition, until the desired maximum size of that partition was to break, for which the algorithm would then assign the remaining jobs into the smaller partition.

Lemma 2.11. *Split4.* For instances, I , where there are no jobs, j , s.t. $p(j) > 2/3$, and no $c \in C$ where there are jobs, $j_i, j_k \in c$, s.t. $p(j_i) > 1/3 \wedge p(j_k) > 1/3$, $c \in C_{\geq 5/6}$ can be split into \hat{c}, \check{c} such that $p(\hat{c}) \leq 5/6$ and $p(\check{c}) \leq 1/3$.

Proof. Let $c' = \{j \in c : p(j) \geq \frac{1}{6}\}$. If $p(c') \leq \frac{5}{6}$, then initialize $\hat{c} = c'$, and then add jobs from $c \setminus c'$ to \hat{c} in any order until it is no longer possible without breaking $p(\hat{c}) \leq \frac{5}{6}$. Now, $\frac{2}{3} \leq p(\hat{c}) \leq \frac{5}{6}$, since all small jobs j satisfy $p(j) < \frac{1}{6}$, so letting all remaining jobs make up \check{c} gives $p(\check{c}) \leq \frac{1}{3}$. If $p(c') > \frac{5}{6}$, then c' contains at least two jobs, so there is a $j' \in c'$ with $\frac{1}{6} \leq p(j') \leq \frac{1}{3}$. In that case, let $\check{c} = \{j'\}$ and $\hat{c} = c \setminus \{j'\}$. \square

Lemma 2.12. *Split5.* For instances, I , where there are no jobs, j , s.t. $p(j) > 2/3$, and no $c \in C$ where there are jobs, $j_i, j_k \in c$, s.t. $p(j_i) > 1/3 \wedge p(j_k) > 1/3$, $c \in C_{\geq 5/6}$ can be split into \hat{c}, \check{c} such that $p(\hat{c}) \leq 2/3$ and $p(\check{c}) \leq 1/2$.

Proof. Let j' be the biggest job in c . If $\frac{1}{2} < p(j') \leq \frac{2}{3}$, then let $\hat{c} = \{j'\}$ and $\check{c} = c \setminus \{j'\}$. If $\frac{1}{3} < p(j') \leq \frac{1}{2}$ then let $\check{c} = \{j'\}$ and $\hat{c} = c \setminus \{j'\}$. If neither are true, then all jobs j in c satisfy $p(j) \leq \frac{1}{3}$. In that case, initialize $c' = \emptyset$, and in any order, add jobs to c' until $p(c') \geq \frac{1}{3}$. Now, $\frac{1}{3} \leq p(c') \leq \frac{2}{3}$. If $p(c') \leq \frac{1}{2}$, let $\check{c} = c'$ and $\hat{c} = c \setminus c'$. Otherwise, let $\hat{c} = c'$ and $\check{c} = c \setminus c'$. \square

Lemma 2.13. *Split6.* We can always partition the classes $c \in C_{[2/3, 5/6)}$ into $\hat{c}, \check{c} \subseteq c$ s.t. inequalities $p(\hat{c}) \leq 1/3, p(\check{c}) \leq 2/3$ hold, using a greedy partitioning algorithm given that no job has processing time $p(j) > 2/3$ and no two jobs in c have $p(j) > 1/3$.

Proof. Partition the classes as following: sort the jobs into descending order, iterate over the jobs and assign j to \hat{c} if $p(\hat{c}) + p(j) \leq 2/3$, otherwise assign j to \check{c} .

Assume that we have ended up with $p(\check{c}) > 1/3$. Then, the last job assigned to \check{c} , say j_n , must fulfil $p(\hat{c}) + p(j_n) > 2/3$. But it must hold that $p(\hat{c}) + p(\check{c}) \leq 5/6$ for this class c and as $p(\check{c}) > 1/3 \implies p(\hat{c}) < \frac{5}{6} - \frac{1}{3} = 1/2$. This in turn means that $p(j_n) > 1/6$, and also there must be another job in \check{c} with processing time $\geq p(j_n)$. But if there are two jobs with $p(j) > 1/6$ in \check{c} , $p(\hat{c}) + p(j_n) > 2/3$, and $p(j_m) > 1/6$ for the other job in \check{c} , then $p(\hat{c}) + p(j_n) + p(j_m) > 5/6$ so this class can not be within the set $C_{[2/3, 5/6)}$. \square

4/3-algorithm for instances without huge jobs and $|C'_{(4/9, 1/2)}| = 0$: We will now first present the aforementioned algorithm and then prove the validity of Theorem 2.9. We will keep the following invariant throughout this algorithm:

Invariant 1. *The total load of scheduled jobs is lower bounded by the number of closed machines, and after each step the cardinality of the set of unused machines, \overline{M}_u , is at least*

$$|\overline{M}_u| \geq \max \left\{ \left\lceil \frac{1}{12} (10|\overline{C}_{\geq 5/6}| + 9|\overline{C}_{[3/4, 5/6]}| + 8|\overline{C}_{[2/3, 3/4]}| + 6|\overline{C}_{[1/2, 2/3]}| + 4|\overline{C}_{[1/3, 1/2]}|) \right\rceil, \lceil p(\overline{\mathcal{J}}) \rceil \right\}.$$

Observe that the following algorithm is defined in a way such that no two jobs overlap on the same machine, and no two jobs from the same class are scheduled simultaneously.

Step 1. While $|\overline{C}_{\geq 5/6}| \geq 6$: Take $c_1, c_2, c_3, c_4, c_5, c_6 \in \overline{C}_{\geq 5/6}$. Use **Split4** for c_2, c_5 and **Split5** for c_3, c_4 . Schedule c_1 on one machine, starting at 0, and \hat{c}_2 ending at $4/3$. On a second machine, schedule \hat{c}_2 starting at 0, and \hat{c}_3 ending at $4/3$. On a third machine schedule \hat{c}_3 starting at 0, and \hat{c}_4 ending at $4/3$. On a fourth machine, schedule \hat{c}_4 starting at 0, and \hat{c}_5 ending at $4/3$. On a fifth machine schedule \hat{c}_5 , starting at 0, and c_6 ending at $4/3$. Close all five machines.

Observation: For a single iteration of this step, the classes c_2, c_5 can be partitioned like **Split4** by Lemma 2.11 and c_3, c_4 can be partitioned like **Split5** by Lemma 2.12. After scheduling the classes $c_1, c_2, c_3, c_4, c_5, c_6$, each machine will have a load in $[1, 4/3]$, since $1 \leq p(c_1) + p(\hat{c}_2), p(\hat{c}_2) + p(\hat{c}_3), p(\hat{c}_3) + p(\hat{c}_4), p(\hat{c}_4) + p(\hat{c}_5), p(\hat{c}_5) + p(c_6) \leq 4/3$. Invariant 1 still holds: The total load of these classes is at least 5 and we closed 5 machines, therefore $|\overline{M}_u| \geq \lceil p(\overline{\mathcal{J}}) \rceil$. Before this step, it holds that

$$|\overline{M}_u| \geq \left\lceil \frac{1}{12} (10|\overline{C}_{\geq 5/6}| + 9|\overline{C}_{[3/4, 5/6]}| + 8|\overline{C}_{[2/3, 3/4]}| + 6|\overline{C}_{[1/2, 2/3]}| + 4|\overline{C}_{[1/3, 1/2]}|) \right\rceil.$$

For each time this step is executed, the left hand side is reduced by 5, and the right hand side by $\frac{1}{12} \cdot 10 \cdot 6 = 5$ as it removed 5 classes from $\overline{C}_{\geq 5/6}$. Note that after this step, $|\overline{C}_{\geq 5/6}| \leq 5$.

Step 2. While $|\overline{C}_{[1/2, 2/3]}| \geq 2$, take $c_1, c_2 \in \overline{C}_{[1/2, 2/3]}$ and schedule them on one machine, then close it.

Observation: For every iteration of this step, the machine will have a load in $[1, 4/3]$, as $1 \leq p(c_1) + p(c_2) \leq 4/3$. After scheduling classes c_1, c_2 , Invariant 1 still holds: The total load of these classes is at least 1 and we closed 1 machine, so $|\overline{M}_u| \geq \lceil p(\overline{\mathcal{J}}) \rceil$. Before this step it holds that

$$|\overline{M}_u| \geq \left\lceil \frac{1}{12} (10|\overline{C}_{\geq 5/6}| + 9|\overline{C}_{[3/4, 5/6]}| + 8|\overline{C}_{[2/3, 3/4]}| + 6|\overline{C}_{[1/2, 2/3]}| + 4|\overline{C}_{[1/3, 1/2]}|) \right\rceil.$$

For each time this step is executed, the left-hand side is reduced by 1, and the right-hand side is reduced by $\frac{1}{12} \cdot 6 \cdot 2 = 1$ as it removed 2 classes from $\overline{C}_{[1/2, 2/3]}$. Note that after this step, $|\overline{C}_{\geq 5/6}| \leq 5, |\overline{C}_{[1/2, 2/3]}| \leq 1$.

Step 3. While $|\overline{C}_{[1/3, 1/2]}| \geq 1$ and $|\overline{C}_{[2/3, 5/6]}| \geq 1$, schedule $c_b \in \overline{C}_{[2/3, 5/6]}$ on an open machine and $c_s \in \overline{C}_{[1/3, 1/2]}$ right after c_b is finished and close the machine.

Observation: For every iteration of this step, the machine will have a load in $[1, 4/3]$, as $1 \leq p(c_s) + p(c_b) \leq 4/3$. Invariant 1 still holds after scheduling classes c_s, c_b : The total load of these classes is at least 1 and we closed 1 machine, thus $|\overline{M}_u| \geq \lceil p(\overline{\mathcal{J}}) \rceil$. Before this step, it holds that

$$|\overline{M}_u| \geq \left\lceil \frac{1}{12} (10|\overline{C}_{\geq 5/6}| + 9|\overline{C}_{[3/4, 5/6]}| + 8|\overline{C}_{[2/3, 3/4]}| + 6|\overline{C}_{[1/2, 2/3]}| + 4|\overline{C}_{[1/3, 1/2]}|) \right\rceil.$$

For each time this step is executed, the left-hand side is reduced by 1, and the right-hand side is reduced by at least $\frac{1}{12} (8 \cdot 1 + 4 \cdot 1) = 1$ since it removed 1 class from $\overline{C}_{[2/3, 5/6]}$ and 1 class from $\overline{C}_{[1/3, 1/2]}$. Note that after this step, $|\overline{C}_{\geq 5/6}| \leq 5, |\overline{C}_{[1/2, 2/3]}| \leq 1$, and either $|\overline{C}_{[1/3, 1/2]}| = 0, |\overline{C}_{[2/3, 5/6]}| = 0$, or both.

Step 4. If $|\overline{C}_{[2/3,5/6]}| \neq 0$: Distinguish classes from $\overline{C}_{[2/3,5/6]}$ into $\overline{C}_{[2/3,3/4]}$ and $\overline{C}_{[3/4,5/6]}$.

1) While $|\overline{C}_{[2/3,3/4]}| \geq 3$: Take $c_1, c_2, c_3 \in \overline{C}_{[2/3,3/4]}$. If at least one of the classes does not contain a job with $p(j) > 7/12$, w.l.o.g. let that class be denoted c_1 . We can partition this class, c_1 , so that $p(\hat{c}_1) \leq 7/12, p(\check{c}_1) \leq 7/12$ (proved in the following observation). Schedule c_2 on one machine, starting at 0, and \hat{c}_1 ending at $4/3$. Then schedule \check{c}_1 on the next machine starting at 0, and c_3 after it. Close both machines. If all of the classes contain jobs with $p(j) > 7/12$, we can partition them so that $p(\hat{c}) \leq 2/3, p(\check{c}) \leq 1/4$ (proved in the following observation). Then schedule \hat{c}_1 on the first machine, starting at 0, and follow it by \hat{c}_2 so that it ends at $4/3$. On the second machine, schedule \check{c}_2 , starting at 0, c_3 , starting at $1/4$ and ending before 1, and \check{c}_1 ending at $4/3$.

2) While $|\overline{C}_{[3/4,5/6]}| \geq 4$: Take $c_1, c_2, c_3, c_4 \in \overline{C}_{[3/4,5/6]}$. These classes can be partitioned so that $p(\hat{c}_i) \leq 2/3, p(\check{c}_i) \leq 1/3$ for $i \in \{1, 2, 3, 4\}$. Now, on the first machine, schedule \hat{c}_1 starting at 0 and \hat{c}_2 ending at $4/3$. On a second machine, schedule \hat{c}_3 starting at 0, and \hat{c}_4 ending at $4/3$. On a third machine, schedule \check{c}_2 starting at 0, \check{c}_4 starting at $1/3$, \check{c}_1 starting at $2/3$ and \check{c}_3 starting at 1.

Observation: Invariant 1 will always hold each time 1) or 2) is executed: In 1), the total load of the placed classes is at least 2 and we closed 2 machines, so $|\overline{M}_u| \geq \lceil p(\overline{\mathcal{J}}) \rceil$. Before this step,

$$|\overline{M}_u| \geq \lceil \frac{1}{12}(10|\overline{C}_{\geq 5/6}| + 9|\overline{C}_{[3/4,5/6]}| + 8|\overline{C}_{[2/3,3/4]}| + 6|\overline{C}_{[1/2,2/3]}| + 4|\overline{C}_{[1/3,1/2]}|) \rceil$$

is satisfied. For each time 1) is executed, the left-hand side is reduced by 2 and the right-hand side is reduced by $\frac{1}{12} \cdot 8 \cdot 3 = 2$ since it removed 3 classes from $\overline{C}_{[2/3,3/4]}$.

In 2), the total load of the placed classes is at least 3 and the number of closed machines will be 3, so $|\overline{M}_u| \geq \lceil p(\overline{\mathcal{J}}) \rceil$. Before 2),

$$|\overline{M}_u| \geq \lceil \frac{1}{12}(10|\overline{C}_{\geq 5/6}| + 9|\overline{C}_{[3/4,5/6]}| + 8|\overline{C}_{[2/3,3/4]}| + 6|\overline{C}_{[1/2,2/3]}| + 4|\overline{C}_{[1/3,1/2]}|) \rceil$$

holds. For each time 2) is executed the left hand side is reduced by 3, and the right hand side is reduced by $\frac{1}{12} \cdot 9 \cdot 4 = 3$ since it removed 4 classes from $\overline{C}_{[3/4,5/6]}$.

Claim: Executing this step ensures that no two jobs overlap on the same machine, and no two jobs from the same class are scheduled simultaneously.

Proof. In 1), if there is one of the chosen classes, which does not contain a job with $p(j) > 7/12$, we can partition this class so that $p(\check{c}) \leq p(\hat{c}) \leq 7/12$, using a greedy algorithm, which can be proven similar to Lemma 2.13. This ensures that $p(c_2) + p(\hat{c}_1) \leq 4/3$ and $p(\check{c}_1) + p(c_3) \leq 4/3$. Note that the partitions will not overlap as $p(c_1) \leq 3/4$. If all of the chosen classes contain jobs with $p(j) > 7/12$ we can partition these classes as stated with a greedy algorithm by Lemma 2.13. As we place three classes, with a minimum processing time of $2/3$ on two machines, we ensure that the average load on these machines will be ≥ 1 .

In 2) we know that the classes can be split in this way by Lemma 2.13. No classes will have overlapping jobs since \hat{c}_1, \check{c}_3 will be scheduled in the time frame $(0, 2/3)$ and their respective smaller partitions \check{c}_1, \hat{c}_3 will be scheduled within $(2/3, 4/3)$ and vice versa for c_2, c_4 . As we here place four classes with a minimum processing time of $3/4$ on three machines, we ensure that the average load on these machines will be ≥ 1 . As $p(\check{c}_3) \leq 1/3$ it will end before $4/3$ when starting at 1. \square

Note that after this step, $|\overline{C}_{\geq 5/6}| \leq 5$, $|\overline{C}_{[1/2,2/3]}| \leq 1$, and either $|\overline{C}_{[1/3,1/2]}| = 0$, $|\overline{C}_{[2/3,5/6]}| = 0$, or both (but it is guaranteed that $|\overline{C}_{[2/3,5/6]}| \leq 5$).

Step 5a. If $|\overline{C}_{[2/3,5/6]}| = 0$: We know that $|\overline{C}_{\geq 5/6}| \leq 5, |\overline{C}_{[1/2,2/3]}| \leq 1$ and $|\overline{C}_{\leq 1/2}| \geq 0$. Place all the classes from $\overline{C}_{\geq 5/6}$ on separate machines. Then use a rotational procedure for attempting to place the residual classes from $\overline{C}_{[1/2,2/3]}$ and $\overline{C}_{[1/3,1/2]}$ on these machines. If $|\overline{C}_{[1/2,2/3]}| = 1$, try to place this class c_m on the first machine. Let c_1 denote the huge class placed on the first machine in this step. If $p(c_m) + p(c_1) > 4/3$, use **Split5** on c_1 , so that $p(\hat{c}_1) \leq 2/3, p(\check{c}_1) \leq 1/2$ and "rotate" \check{c}_1 onto the second machine, so that it ends at $4/3$. If this rotation were to place a load $> 4/3$ on the second machine, use **Split4** on the huge class already placed on it, c_2 , so that $p(\hat{c}_2) \leq 5/6, p(\check{c}_2) \leq 1/3$ and rotate \check{c}_2 on to the next machine. Close the machines with load > 1 .

Rotation procedure for classes in $\overline{C}_{[1/3,1/2]}$: If $|\overline{C}_{[1/3,1/2]}| \geq 1$, try to place this class, c_s , onto the first open machine. Let c_i denote the huge class on the first open machine for this sub step. If $p(c_s) + p(c_i) > 4/3$, use **Split4** on c_i and "rotate" \check{c}_i onto the next open machine, so that it ends at $4/3$. Close the machines with load > 1 . Repeat this procedure until all machines where there were a huge class placed in the beginning of this step are iterated over.

Then, while $|\overline{C}_{(1/3,4/9)}| \leq 3$: take classes $c_1, c_2, c_3 \in \overline{C}_{(1/3,4/9)}$ and place them on an empty machine. If after this $|\overline{C}_{(1/3,4/9)}| \geq 1$, take one class $c \in \overline{C}_{(1/3,4/9)}$ and schedule it on any open machine. If $|\overline{C}_{(1/3,4/9)}| = 1$ at this point, greedily try to schedule this class in the latest possible time slot on any open machine. After this, place jobs from $c \in \overline{C}_{(4/9,1/2)}$ greedily onto open machines, splitting them into $c', c \setminus c'$ s.t. $p(c') \leq 1/3$ if this were to place a load $> 4/3$ on a machine. Finally, greedily schedule the remaining classes from $\overline{C}_{\leq 1/3}$ on the remaining open machines, closing all machines with load > 1 .

Claim: This step always generates a feasible schedule with no job ending after $4/3$. If this step is executed, then all jobs have been scheduled.

Proof. Prior to this step we ensure that Invariant 1 holds. This means that there will always be enough machines left to schedule all the huge classes on separate machines, as $|\overline{M}_u| \geq \lceil \frac{5}{6} |\overline{C}_{\geq 5/6}| \rceil$ and there are at most 5 of these classes left. For the cases where the algorithm would have to schedule the last smaller partition of a huge class on a new, open, empty machine, it is always possible as this would mean that the total load on the used machines, including those used by the rotation procedure, would exceed or equal the number of machines, so for an optimal schedule, there must be at least one more machine to accommodate for this remaining smaller partition as we have $|\overline{M}_u| \geq \lceil p(\overline{J}) \rceil$.

Furthermore it is possible to schedule the remaining classes in $\overline{C}_{\leq 1/2}$. Scheduling 3 classes from $\overline{C}_{(1/3,4/9)}$ on a machine guarantees a load in $(1, 4/3)$. For the case where the rotational procedure has rotated a partition of a huge class onto an empty machine, that partition must have a processing time $p(\check{c}) \leq 1/2$, which means that at least one more class from $\overline{C}_{(1/3,4/9)}$ can be scheduled on it. If there is still one class from $\overline{C}_{(1/3,4/9)}$, we can either schedule it on this partially filled machine if the load was not to exceed $4/3$ or schedule it on a new machine, which must exist by the invariant. Then we can continue with greedily scheduling the remaining classes in $\overline{C}_{(4/9,1/2)}$. As $|\overline{C}'_{(4/9,1/2)}| = 0$ for this instance, whenever we were to overload a machine with more than $1/3$ by placing these classes, we can simply split $c \in \overline{C}_{(4/9,1/2)}$ in a way such that $c' \subseteq c, p(c') \leq 1/3$, since the largest job in c has $p(j) \leq 1/3$, and schedule $c \setminus c'$ on the next open machine. Note that this will not cause any overlap since by scheduling the eventual last class from $\overline{C}_{(1/3,4/9)}$ at the latest possible time slot, we ensure that subsequent open machines do not have open space in the same time frame where these partitions of c could overlap. Finally, we can greedily schedule classes from $\overline{C}_{\leq 1/3}$, on the remaining open machines as these guarantee a load in $[1, 4/3]$ with the existing configuration. \square

Step 5b. If $|\overline{C}_{[2/3,5/6]}| \geq 1$, we know that $|\overline{C}_{[2/3,5/6]}| \leq 5$ (and that $|\overline{C}_{[2/3,3/4]}| \leq 2$ and $|\overline{C}_{[3/4,5/6]}| \leq 3$), $|\overline{C}_{[1/3,1/2]}| = 0$, and $|\overline{C}_{\leq 1/3}| \geq 0$. In this step, we will schedule the remaining classes from $\overline{C}_{\geq 5/6}$ iteratively like in step 1, and based on how many there are in this set, we will choose a different iterating approach for the remaining classes in $\overline{C}_{[2/3,5/6]}$. Let $c_1, c_2, c_3, c_4, c_5 \in \overline{C}_{[2/3,5/6]}$ if they exist.

1) If $|\overline{C}_{\geq 5/6}| \geq 4$ before this step: iterate over the classes in $\overline{C}_{[2/3,5/6]}$. Do the following steps if possible ($|\overline{C}_{[2/3,5/6]}| \geq 1$): Place c_1 on the first open machine (this will be where a partition of a huge class is scheduled starting at 0), ending at $4/3$. Place \hat{c}_2 on the second open machine starting at 0, and \check{c}_2 on the third open machine, ending at $4/3$. Place \hat{c}_3 on the second open machine, ending at $4/3$, and \check{c}_3 on the third open machine, starting at 0. Place c_4 on a fourth open machine starting at 0.

2) If $|\overline{C}_{\geq 5/6}| = 3$ before this step: iterate over the classes in $\overline{C}_{[2/3,5/6]}$. Do the following steps if possible ($|\overline{C}_{[2/3,5/6]}| \geq 1$): Place \hat{c}_1 on the first open machine, ending at $4/3$ (this will be where a partition of a huge class is scheduled starting at 0). On the next open machine place \check{c}_1 starting at 0. Place c_2 on this second open machine, ending at $4/3$. Place \hat{c}_3 on a third open machine, starting

at 0, and \check{c}_3 on a fourth open machine, ending at $4/3$. Place \hat{c}_4 on the third open machine, so that it ends at $4/3$, and \check{c}_4 on the fourth open machine starting at 0. Go to the fifth open machine and place c_5 starting at 0.

3) If $|\overline{C}_{\geq 5/6}| = 2$ or $|\overline{C}_{\geq 5/6}| = 1$ before this step: iterate over the classes in $\overline{C}_{[2/3, 5/6]}$. Do the following steps if possible ($|\overline{C}_{[2/3, 5/6]}| \geq 1$): On the first open machine, place \check{c}_1 ending at $4/3$ and \hat{c}_1 on the second open machine, starting at 0. Place \hat{c}_2 on the second open machine, ending at $4/3$ and \check{c}_2 on a third open machine, starting at 0. Place c_3 on the third open machine so that it ends on $4/3$. Place c_4 on a fourth open machine, starting at 0. Place \check{c}_5 on the fourth open machine, ending at $4/3$ and \hat{c}_5 on a fifth open machine, starting at 0.

4) If $|\overline{C}_{\geq 5/6}| = 0$ before this step: iterate over the classes in $\overline{C}_{[2/3, 5/6]}$. Do the following steps if possible ($|\overline{C}_{[2/3, 5/6]}| \geq 1$): Place \hat{c}_1 on the first open machine, starting at 0 and \check{c}_1 on the second open machine, ending at $4/3$. Place \hat{c}_2 on the first machine, ending at $4/3$ and \check{c}_2 on the second machine, starting at 0. Place \hat{c}_3 on the second machine, starting at $1/3$ and \check{c}_4 on a third open machine, starting at 0. Place c_4 on the third machine, ending at $4/3$. Place \hat{c}_5 on a fourth open machine, starting at 0 and \check{c}_5 on a fifth open machine, ending at $4/3$.

If $|\overline{C}_{[1/2, 2/3]}| = 1$ greedily schedule this class on an open machine. Then schedule the residual classes $\overline{C}_{\leq 1/3}$ greedily on the remaining open machines.

Claim: After this step, all jobs have been scheduled, the schedule is feasible and no job ends after $4/3$.

Proof. Prior to this step, we ensure that Invariant 1 holds. When iterating over the remaining classes in $\overline{C}_{\geq 1/2}$ we ensure that we use no more machines than the cardinality of each set of classes times its minimum amount of time slots used. It is then possible to greedily schedule the residual classes in $\overline{C}_{\leq 1/3}$ as we then guarantee that the load of the residual machines will be in $[1, 4/3]$. \square

Finally, it remains to show that the algorithm has scheduled all the given classes in the instance. Note that after step 1, $|\overline{C}_{\geq 5/6}| \leq 5$, after step 2, $|\overline{C}_{[1/2, 2/3]}| \leq 1$, after step 3, either $|\overline{C}_{[1/3, 1/2]}| = 0$, $|\overline{C}_{[2/3, 5/6]}| = 0$, or both. After step 4, it is certain that $|\overline{C}_{[2/3, 5/6]}| \leq 5$. Here we make a case distinction: assuming that $|\overline{C}_{[2/3, 5/6]}| = 0$ at this point, step 5a is executed, which schedules all remaining classes in $\overline{C}_{\geq 5/6}$, the eventual remaining class in $\overline{C}_{[1/2, 2/3]}$ and all remaining classes in $\overline{C}_{\leq 1/2}$. Thus, all classes are scheduled if step 5a is executed.

If instead $|\overline{C}_{[2/3, 5/6]}| \geq 1$, then we know that $|\overline{C}_{[1/3, 1/2]}| = 0$ and we instead execute step 5b. We then also schedule the remaining classes in $\overline{C}_{\geq 5/6}$, the remaining classes in $\overline{C}_{[2/3, 5/6]}$, the eventual remaining class in $\overline{C}_{[1/2, 2/3]}$ and all residual classes in $\overline{C}_{\leq 1/3}$. Hence, all classes are scheduled after step 5b.

The existence and validity of the described algorithm proves Theorem 2.9.

2.5 Working steps for a 4/3-algorithm with huge jobs

This is working steps to a 4/3-Algorithm for scheduling the classes which was not scheduled in section 2.4. That is, classes containing a huge job ($p(j) > \frac{2}{3}T$) denoted by C_H and classes that contain at least two jobs with $p(j) > \frac{1}{3}T$ denoted by C_M . By the design of the schedules, no two jobs overlap on the same machine, no two jobs from the same class are processed simultaneously, and no machine exceeds a load of $4/3$. While these steps do not constitute a complete algorithm, they provide a framework for scheduling certain combinations of the remaining unscheduled classes.

Partitioning of classes for scheduling:

splitA: $c \in C_{\leq 11/12} \cap C_M$ can be partitioned into \hat{c} and \check{c} , such that $1/3 < p(\hat{c}) \leq 7/12$ and $1/3 < p(\check{c}) < 11/24$.

splitB: $c \in C_{\geq 11/12} \cap C_M$ can be partitioned into \hat{c} and \check{c} , such that $1/3 < p(\hat{c}) \leq 2/3$ and $1/3 < p(\check{c}) < 1/2$.

splitC: $c \in C_{(3/4, 11/12)} \setminus (C_H \cup C_M)$ can be partitioned into \hat{c} and \check{c} , such that $5/12 < p(\hat{c}) \leq 3/4$ and $1/6 < p(\check{c}) < 1/3$.

Lemma 2.14. *While $|c_0 \in \overline{C}_{\geq 7/8} \cap C_H| \geq 1$, $|c_1 \in \overline{C}_{(5/6, 7/8)} \cap C_H| \geq 1$, $|c_2 \in \overline{C}_{\leq 11/12} \cap C_M| \geq 1$, $|c_3 \in \overline{C}_{\geq 11/12} \cap C_M| \geq 1$, $|c_4 \in \overline{C}_{\leq 5/6} \cap C_H| \geq 1$ and $\sum_{c \in \overline{C}_{\leq 1/3}} c \geq 1/24$, there exists a schedule of the classes c_0, c_1, c_2, c_3, c_4 and some classes from $C_{\leq 1/3}$ that can be scheduled on four machines, such that the average load is at least 1 and each machine has load at most $4/3$.*

Proof. Partition c_2 into \hat{c}_2 and \check{c}_2 using **splitA** such that $1/3 < p(\hat{c}_2) \leq 7/12$ and $1/3 < p(\check{c}_2) < 11/24$. Then partition c_3 into \hat{c}_3 and \check{c}_3 using **splitB** such that $1/3 < p(\hat{c}_3) \leq 2/3$ and $1/3 < p(\check{c}_3) < 1/2$. Schedule c_0 with start time 0 on the first open machine, meaning that there is nothing scheduled in the time interval $[1, 4/3]$. Then greedily schedule $c \in C_{\leq 1/3}$ starting at $p(c_0) \leq 1$ until the sum of the greedily schedule classes exceeds $1/24$. This is possible because either there exists one class $c \in C_{(1/24, 1/3]}$ or a sequence of classes $c \in C_{< 1/24}$ exist such that their total load exceeds $1/24$, but remains below $1/3$, allowing then to be scheduled before time $4/3$. Schedule c_1 with start time 0 and \check{c}_2 with end time $4/3$ on the second. Schedule \hat{c}_2 with start time 0 and \hat{c}_3 with end time $4/3$ on the third. Schedule \check{c}_3 with start time 0 and c_4 with end time $4/3$ on the fourth. Since the total load of the classes c_0, c_1, c_2, c_3 and c_4 is at least $7/8 + 5/6 + 2/3 + 11/12 + 2/3 = 3 + 23/24$ and they are scheduled together with classes $c \in C_{\leq 1/3}$ until load of them are greater than $1/24$. Thus, the average load per machine is at least 1, and each machine's load does not exceed $4/3$. \square

Lemma 2.15. *While $|c_1 \in C_{(5/6, 7/8)} \cap C_H| \geq 1$, $|c_2 \in C_{\leq 11/12} \cap C_M| \geq 1$, $|c_3 \in C_{\geq 11/12} \cap C_M| \geq 1$ and $|c_4 \in C_{\leq 5/6} \cap C_H| \geq 1$, there exists a schedule of the classes c_1, c_2, c_3 , and c_4 that can be scheduled on three machines such that the average load is at least 1 and the load on each machine is at most $4/3$.*

Proof. Partition c_2 into \hat{c}_2 and \check{c}_2 using **splitA** such that $1/3 < p(\hat{c}_2) \leq 7/12$ and $1/3 < p(\check{c}_2) < 11/24$. Then partition c_3 into \hat{c}_3 and \check{c}_3 using **splitB** such that $1/3 < p(\hat{c}_3) \leq 2/3$ and $1/3 < p(\check{c}_3) < 1/2$. Schedule c_1 with start time 0 and \check{c}_2 to end at time $4/3$ on the first machine. Then schedule \hat{c}_2 with start time 0 and \hat{c}_3 to end at time $4/3$ on the second machine. Finally, schedule \check{c}_3 with start time 0 and c_4 to end at time $4/3$ on the third machine. The total load of the classes c_1, c_2, c_3 , and c_4 is at least $5/6 + 2/3 + 11/12 + 2/3 = 3 + 1/12$. Thus, the average load per machine is at least 1, and each machine's load does not exceed $4/3$. So the average load is at least 1 and each machine's load does not exceed $4/3$. \square

Lemma 2.16. *While $|c_2 \in C_{\leq 11/12} \cap C_M| \geq 1$, $|c_3 \in C_{\geq 11/12} \cap C_M| \geq 1$, $|c_4 \in C_{\leq 5/6} \cap C_H| \geq 1$ and $|c \in C_{(3/4, 11/12)} \setminus (C_H \cup C_M)| \geq 1$, there exists a schedule of the classes c_2, c_3, c_4 , and c that can be scheduled on three machines such that the average load is at least 1 and the load on each machine is at most $4/3$.*

Proof. Partition c to \hat{c} and \check{c} using **splitC** such that $5/12 < p(\hat{c}) \leq 3/4$ and $1/6 < p(\check{c}) < 1/3$. Then partition c_2 into \hat{c}_2 and \check{c}_2 using **splitA** such that $1/3 < p(\hat{c}_2) \leq 7/12$ and $1/3 < p(\check{c}_2) < 11/24$. Schedule c_3 with start at 0 and \check{c} to end at $4/3$ on the first machine. Then schedule \hat{c} with start at 0 and \hat{c}_2 to end at $4/3$ on the second machine. Finally, schedule \check{c}_2 with start at 0 and c_4 to end at $4/3$ on the third machine. The total load of the classes c_2, c_3, c_4 , and c is at least $2/3 + 11/12 + 2/3 + 3/4 = 3$. So the average load is at least 1 and each machine's load does not exceed $4/3$. \square

3 Optimizations

In this section the three algorithms ENQUEUE, Downshifting and simulated annealing are presented.

3.1 ENQUEUE

The ENQUEUE algorithm is a 2-approximation from [2]. It is defined in Algorithm 9.

Algorithm 9: ENQUEUE

Input : An ordering of n jobs \mathcal{J} , m machines

```
1
2 for each job  $j$  in  $\mathcal{J}$  do
3   | if there exists a machine  $M$  whose last job is of class  $c(j)$  then
4   |   | Schedule  $j$  on  $M$ 
5   | else
6   |   | Schedule  $j$  on the machine with lowest completion time
7 return schedule as  $t, \sigma$ 
```

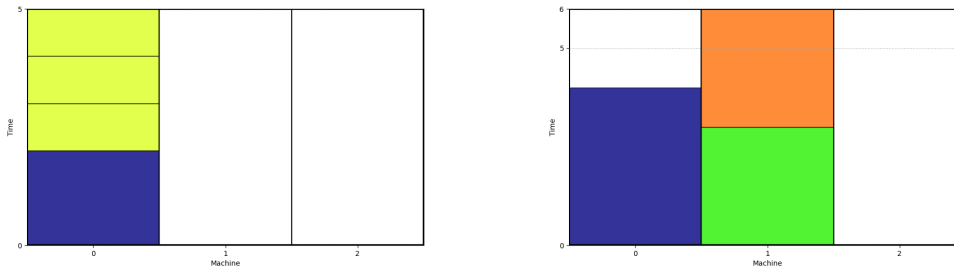
3.2 Downshifting

For the 3/2- and 5/3-algorithms, jobs are sometimes scheduled with a certain end time, regardless of whether it is possible to schedule them earlier, which would potentially give a shorter makespan (see Figures 2(a) and 2(e)). The Downshifting algorithm takes a schedule as input, and reschedules jobs earlier whenever possible. Figure 1 shows that downshifting does not improve the makespan in general, as in this case there is no job that can be scheduled earlier. On the other hand, downshifting will never make the makespan worse.

Algorithm 10: Downshifting

Input : n jobs, m machines, a scheduling σ, τ

```
1
2 for  $i$  in  $\{1, 2, \dots, n\}$  in order of increasing  $\tau(i)$  do
3   |  $\tau'(i) \leftarrow$  minimum value such that  $\tau', \sigma$  is a feasible schedule
4 return  $\tau', \sigma$ 
```



(a) 5/3-algorithm. One class with a job of size 2 and one class with three jobs of size 1 gives a makespan of 5, but the optimal makespan is 3.

(b) 3/2-algorithm. Three classes, one with a single job of size 4, and two with jobs of size 3. Makespan becomes 6 instead of the optimal 4.

Figure 1: Small test instances where downshifting does not improve the results for 5/3- and 3/2-algorithms.

3.3 Simulated Annealing

Simulated annealing (SA) is a well-known technique within the field of optimization [10], which traverses the space of possible solutions, hopefully finding a solution close to the optimal. A candidate solution is represented by a permutation π of job priorities, and the algorithm uses `modifyPermutation` as described in Algorithm 11 in order to move from one candidate solution to another.

Using ENQUEUE as defined in Algorithm 9 with jobs ordered according to π generates a schedule (t, σ) . We define a cost function to minimize the makespan $H = \max_i(t(i) + p_i)$ of the

schedule.

Algorithm 11: modifyPermutation

Input : A permutation π

```

1 Sample  $p \sim \text{Uniform}([0, 1])$ 
2 if  $p < 0.1$  then
3   | shuffle( $\pi$ )
4 else if  $p < 0.3$  then
5   | // Choose subset of  $\pi$  and shuffle.
6   | Sample  $P \sim \text{Uniform}(\{S \in \mathcal{P}(\pi) \mid |S| = 5\})$ 
7   | shuffle( $\pi_P$ )
8 else
9   | Sample  $i \sim \text{Uniform}(\pi)$ 
10  | Sample  $j \sim \text{Uniform}(\pi)$ 
11  | temp  $\leftarrow \pi_i$ 
12  |  $\pi_i \leftarrow \pi_j$ 
13  |  $\pi_j \leftarrow \text{temp}$ 
14 return  $\pi$ 

```

Algorithm 12: Simulated Annealing

Input : A set \mathcal{J} of jobs, m machines, *iterationCount*, a permutation π , an initial temperature T , a cooling factor α

```

1
2  $H \leftarrow \text{makespan}(\text{ENQUEUE}(\mathcal{J} \text{ ordered by } \pi, m))$ 
3 for iteration up to iterationCount do
4   |  $\pi' \leftarrow \text{modifyPermutation}(\pi)$ 
5   |  $H' \leftarrow \text{makespan}(\text{ENQUEUE}(\mathcal{J} \text{ ordered by } \pi', m))$ 
6   |  $\Delta \leftarrow H' - H$ 
7   | Sample  $p \sim \text{Uniform}([0, 1])$ 
8   | if  $p > \exp(-\frac{\Delta}{T})$  then
9   |   |  $\pi \leftarrow \pi'$ 
10  |   |  $T \leftarrow \alpha T$ 
11 return  $\text{ENQUEUE}(\mathcal{J} \text{ ordered by } \pi, m)$ 

```

Given the parameters T and α as well as an initial permutation π , **Algorithm 12** uses simulated annealing to find a schedule with a low makespan. In a code implementation of the algorithm, the best schedule found so far is kept track of to ensure best performance.

4 Simulations

In order to gain a more practical understanding of the applications of the algorithms and related optimizations, some simulations were performed allowing performance to be compared.

4.1 Method

The 5/3- and 3/2-algorithms were implemented in Python, along with a simulated annealing approach implemented in C++. Furthermore, two optimized versions of the each of the approximation algorithms were implemented, one where ENQUEUE is applied on the sorted output of the algorithms, and another with Downshifting applied. Test data was generated according to the described procedures. A test set consisting of 10 test files (labeled T1-T10) was created through randomization. Each algorithm was run on the test set, and each algorithm's makespan was recorded as a ratio to the optimal.

All code for algorithms, testing, and data generation, can be found on GitHub.¹

4.1.1 Simulated optimizations

A trivial optimization is to consider all jobs, and greedily schedule them earlier when possible using downshifting. Another way to optimize the result is to use the output from the algorithm as input to a call of ENQUEUE. Take the order generated by sorting all jobs according to the time they were scheduled at in the algorithm, and call ENQUEUE with the jobs in this order in this order, to get an optimized schedule.

Both of these optimizations were implemented and tested on both the 3/2- and the 5/3- algorithms. Additionally, ENQUEUE was simulated with a random ordering of the jobs.

4.1.2 Test data

The test data was generated such that there exists a perfect optimal solution, that is, if any job of any duration and any class is added to the considered set of jobs, the optimal makespan T increases. The generation procedure is described in detail in Algorithm 13.

Algorithm 13: Test Instance Generator

```

1  Sample  $m \sim \text{Uniform}(\{10, \dots, 30\})$ 
2  Sample  $T \sim \text{Uniform}(\{5, \dots, 100\})$ 
3  Sample  $\bar{p} \sim \text{Uniform}(\{5, \dots, T\})$ 
4  Let  $\alpha : \{0, \dots, m-1\} \times \{0, \dots, T-1\} \rightarrow \{-1\} \cup \mathbb{N}$  initialized as  $\alpha(i, t) = -1$  for all  $i, t$ 
5   $C_t \leftarrow \emptyset$  for each  $t \in \{0, 1, \dots, T-1\}$  // Active classes at time  $t$ 
6   $\mathcal{R} \leftarrow \{(i, t) \mid i \in \{0, \dots, m-1\}, t \in \{0, \dots, T-1\}\}$  // Unassigned slots
7   $\mathcal{J} \leftarrow \emptyset$  // Set of generated jobs

8  Define  $\text{mex}(S) = \min\{x \in \mathbb{N}_{>0} \mid x \notin S\}$ 
9  while  $\mathcal{R} \neq \emptyset$  do
10 |   Select  $(i, t) \in \mathcal{R}$  uniformly at random
11 |   Sample  $l' \sim \text{Poisson}(\bar{p} - 1) + 1$ 
12 |    $l \leftarrow \min(l', T - t)$ 
13 |   for  $k = t$  to  $t + l - 1$  do
14 |     |   if  $\alpha(i, k) \neq -1$  then
15 |     |     |    $l \leftarrow k - t$ 
16 |     |     |   break
17 |    $c \leftarrow \text{mex}(\bigcup_{r=t}^{t+l-1} C_r)$ 
18 |    $\mathcal{J} \leftarrow \mathcal{J} \cup \{(l, c)\}$ 
19 |   for  $k = t$  to  $t + l - 1$  do
20 |     |    $\alpha(i, k) \leftarrow c$ 
21 |     |    $C_k \leftarrow C_k \cup \{c\}$ 
22 |     |    $\mathcal{R} \leftarrow \mathcal{R} \setminus \{(i, j)\}$ 

23 Output: Number of machines  $m$  and job set  $\mathcal{J} = \{(p_j, c_j)\}$ 

```

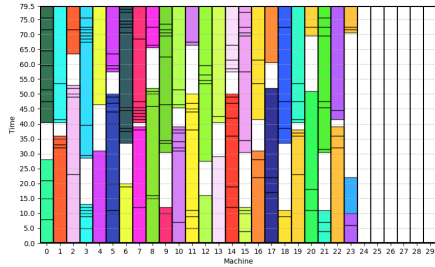
4.2 Results

The results of the simulations are seen in Table 1. They are given as a fraction of the optimal solution; despite the NP-hardness, it is possible to know the optimal solution for these test cases due to the special way that they were generated. In Figure 2 a visual representation of the solutions found on T3 by each algorithm is shown.

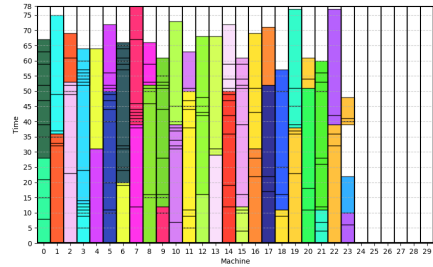
4.3 Discussion of simulation results

Among all the approaches evaluated, simulated annealing consistently delivers the best results. This is expected, as it uses significantly more computational resources and is a well-established

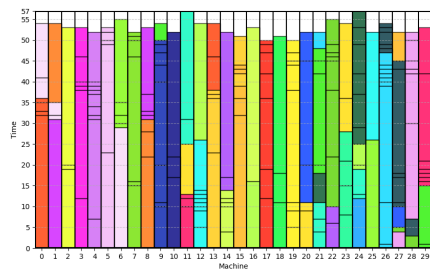
¹<https://github.com/ollelapidus/bachelor-thesis-project-MSRS>



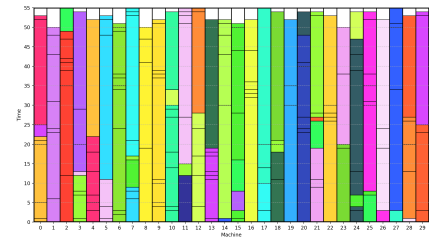
(a) 3/2-algorithm.



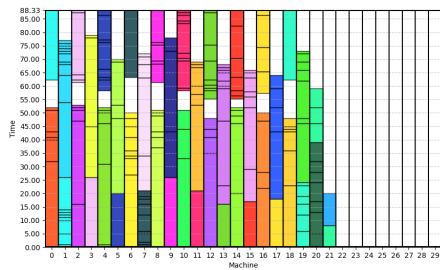
(b) 3/2-algorithm with downshifting.



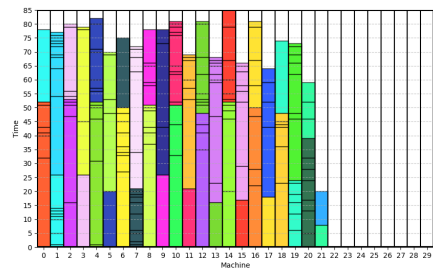
(c) 3/2-algorithm with ENQUEUE.



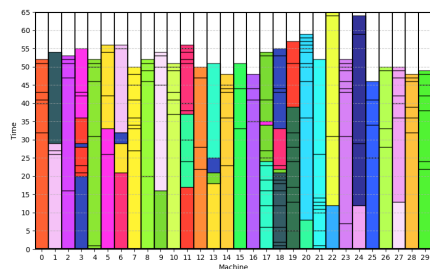
(d) Simulated annealing.



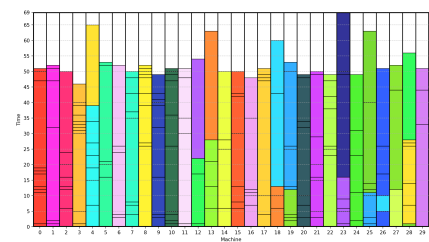
(e) 5/3-algorithm.



(f) 5/3-algorithm with downshifting.



(g) 5/3-algorithm with ENQUEUE.



(h) ENQUEUE.

Figure 2: All algorithms, visualized on data in T3. Each column is a machine, each rectangle a job, and each color a class.

	5/3	5/3D	5/3E	3/2	3/2D	3/2E	SA	ENQUEUE
T1	1.667	1.650	1.200	1.500	1.500	1.050	1.050	1.350
T2	1.667	1.597	1.182	1.500	1.481	1.091	1.013	1.195
T3	1.667	1.604	1.226	1.500	1.472	1.076	1.038	1.396
T4	1.667	1.606	1.169	1.500	1.465	1.155	1.014	1.352
T5	1.667	1.576	1.141	1.500	1.500	1.065	1.011	1.446
T6	1.667	1.584	1.202	1.500	1.449	1.180	1.011	1.416
T7	1.667	1.667	1.222	1.500	1.467	1.133	1.022	1.200
T8	1.667	1.571	1.095	1.500	1.429	1.048	1.000	1.095
T9	1.667	1.667	1.154	1.500	1.487	1.077	1.025	1.538
T10	1.667	1.600	1.244	1.500	1.467	1.156	1.022	1.422
Average	1.667	1.592	1.184	1.500	1.472	1.103	1.021	1.341
Worst	1.667	1.667	1.244	1.500	1.500	1.180	1.050	1.538

Table 1: Fraction of makespan generated compared to optimal solution for each algorithm on each test. Algorithm names are abbreviated, with D for downshifting and E for ENQUEUE.

technique in combinatorial optimization [10]. Its superior performance illustrates an important distinction: while heuristic methods like simulated annealing are practical tools for solving real-world problems, the primary role of the approximation algorithms discussed throughout this paper is theoretical. They serve to demonstrate that certain bounds are achievable, rather than to provide optimal or near-optimal solutions in practice.

The downshifting technique offers some improvement, but its benefits are modest and highly instance-dependent. The data show that in the worst case, downshifting yields no improvement over the base algorithm. This confirms that while downshifting can occasionally reduce makespan, it does not fundamentally change the performance guarantees of the algorithm, and there exist inputs for which it is entirely ineffective.

Interestingly, the ENQUEUE heuristic performs better when it operates on the output of an approximation algorithm (the 5/3- or 3/2-algorithms) than when it is applied to a random job ordering. This suggests that even imperfect schedules produced by approximation algorithms contain useful structural information that ENQUEUE can exploit to refine the result further. In contrast, with random input, ENQUEUE lacks this structure and performs worse. The likely explanation for this is that ENQUEUE on its own is only a 2-approximation, meaning that in the worst case it yields a schedule two times worse than the optimal. On the other hand, ENQUEUE called on the output of the 3/2-algorithm is a 3/2-approximation, because it cannot make the makespan worse, and thus inherits the approximation factor from the 3/2-algorithm.

The visualizations seen in Figure 2 reveal a common shortcoming of the approximation algorithms: they do not utilize all available machines. This inefficiency contributes to their higher makespans and reflects the fact that these algorithms are designed to ensure provable bounds rather than to maximize resource utilization.

References

- [1] M. A. Deppert, K. Jansen, M. Maack, S. Pukrop, and M. Rau, “Scheduling with many shared resources,” in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 413–423, 2023.
- [2] E. Hebrard, M.-J. Huguet, N. Jozefowicz, A. Maillard, C. Pralet, and G. Verfaillie, “Approximation of the parallel machine scheduling problem with additional unit resources,” *Discrete Applied Mathematics*, vol. 215, pp. 126–135, 2016.
- [3] J. Blazewicz, J. Lenstra, and A. Kan, “Scheduling subject to resource constraints: classification and complexity,” *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983.
- [4] R. L. Graham, “Bounds on multiprocessing timing anomalies,” *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

- [5] E. G. Coffman, M. R. Garey, and D. S. Johnson, “An application of bin-packing to multiprocessor scheduling,” *SIAM J. Comput.*, vol. 7, pp. 1–17, 1978.
- [6] S. Berndt, M. Deppert, K. Jansen, and L. Rohwedder, *Load Balancing: The Long Road from Theory to Practice*, pp. 104–116. 01 2022.
- [7] B. S. Baker and E. G. Coffman, “Mutual exclusion scheduling,” *Theoretical Computer Science*, vol. 162, no. 2, pp. 225–243, 1996.
- [8] H. L. Bodlaender and K. Jansen, “On the complexity of scheduling incompatible jobs with unit-times,” in *Mathematical Foundations of Computer Science 1993* (A. M. Borzyszkowski and S. Sokołowski, eds.), (Berlin, Heidelberg), pp. 291–300, Springer Berlin Heidelberg, 1993.
- [9] G. Even, M. Halldórsson, L. Kaplan, and D. Ron, “Scheduling with conflicts: Online and offline algorithms,” *Journal of Scheduling*, vol. 12, pp. 199–224, 04 2009.
- [10] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.

Appendix

Pseudocode for greedy algorithm

Algorithm 14: GreedyAlgorithm(C, M)

Input : (C, M)
Output: (σ, t)

```
1   $\sigma \leftarrow \emptyset, t \leftarrow \emptyset$ 
2  Sort( $C$ ) // Step 1
   /* Sort classes in descending order of processing time. */
3  for  $k = 1$  to  $|C|$  do // Step 2
4  |    $i \leftarrow \arg \min\{p_m(i) \mid i = 1, \dots, m\}$ 
5  |   Schedule( $c_k, i, p_m(i)$ )
6  return ( $\sigma, t$ )
```

Pseudocode for 3/2-algorithm with huge jobs

Algorithm 15: 3/2-algorithm with huge jobs

```

Input :  $C$ 
Output :  $(\sigma, t)$ 
1    $\sigma \leftarrow \emptyset, t \leftarrow \emptyset$ 
2   foreach  $c \in C$  do // Step 1
3   |   if  $p(c) > 3/4$  and  $c \notin C_H$  then
4   |   |    $(\hat{c}, \check{c}) \leftarrow \{\text{split2}(c)\}$ 
5   |   |   if  $1/2 < p(c) < 3/4$  and  $c \in C_B$  then
6   |   |   |    $\hat{c} \leftarrow \{\arg \max_{j \in c} p(j)\}, \check{c} \leftarrow c \setminus \hat{c}$ 
7   |   |   if  $1/2 < p(c) < 3/4$  and  $c \notin C_B$  then
8   |   |   |    $(\hat{c}, \check{c}) \leftarrow \{\text{split3}(c)\}$ 
9   |    $i = 1$ 
10  |   foreach  $c \in C_H$  do // Step 2
11  |   |    $\text{Schedule}(c, i, 0), i \leftarrow i + 1$ 
12  |   while  $|\overline{C}_{\leq 1/2}| \geq 1$  AND  $|\overline{M}_H| \geq 1$  do // Step 3
13  |   |    $\text{GreedyAlgorithm}(\overline{C}_{\leq 1/2}, \overline{M}_H)$ 
14  |   if  $|\overline{M}_H| = 0$  then
15  |   |    $\text{Algorithm 8 } (\overline{C})$ 
16  |   while  $|\overline{M}_H| \geq 2$  and  $|\overline{C}_{(1/2, 3/4)} \setminus C_B| \geq 1$  do // Step 4
17  |   |   |    $(a, b) \leftarrow \{\text{pop}(\overline{M}_H), \text{pop}(\overline{M}_H)\}, c \leftarrow \{\text{pop}(\overline{C}_H \setminus \overline{C}_{(1/2, 3/4)})\}$ 
18  |   |   |    $\text{Reschedule}(b, 3/2 - p_m(b)), \text{Schedule}(\hat{c}, a, 3/2 - p(\hat{c})), \text{Schedule}(\check{c}, b, 0)$ 
19  |   |   |   if  $|\overline{M}_H| = 0$  then
20  |   |   |   |    $\text{Algorithm 8 } (\overline{C})$ 
21  |   if  $|\overline{M}_H| = 1$  then // Step 5
22  |   |   |    $a \leftarrow \{\text{pop}(\overline{M}_H)\}$ 
23  |   |   |   if  $|\overline{C} \setminus C_B| > 0$  then
24  |   |   |   |    $c \leftarrow \{\text{pop}(\overline{C}_{(1/2, 3/4)})\}, (\hat{c}, \check{c}) \leftarrow \text{split3}(c)$ 
25  |   |   |   |    $\text{Reschedule}(a, 3/2 - p_m(a)), \text{Schedule}(\hat{c}, a, 0)$ 
26  |   |   |   |    $\text{Algorithm 8 } (\overline{C})$ 
27  |   |   |   if  $|\overline{C} \setminus C_B| = 0$  then
28  |   |   |   |   foreach  $c \in \overline{C}$  do
29  |   |   |   |   |    $\text{Schedule}(c, i, 0), i \leftarrow i + 1$ 
30  |   while  $|\overline{M}_H| \geq 1$  AND  $|\overline{C}_{(1/2, 3/4)} \cap C_B| \geq 1$  AND  $|\overline{C}_{\geq 3/4}| \geq 1$  do // Step 6
31  |   |   |    $a \leftarrow \{\text{pop}(\overline{M}_H)\}, c_1 \leftarrow \{\text{pop}(\overline{C}_{\geq 3/4})\}, (\hat{c}_1, \check{c}_1) \leftarrow \{\text{split2}(c_1)\},$ 
32  |   |   |   |    $c_2 \leftarrow \{\text{pop}(\overline{C}_{(1/2, 3/4)} \cap C_B)\}$ 
33  |   |   |   |    $\text{Schedule}(\hat{c}_1, a, 3/2 - p(\hat{c}_1)), \text{Schedule}(\hat{c}_1, i, 0)$ 
34  |   |   |   |    $\text{Schedule}(c_2, i, 3/2 - p(c_2)), i \leftarrow i + 1$ 
35  |   |   |   if  $|\overline{M}_H| = 0$  then
36  |   |   |   |    $\text{Algorithm 8 } (\overline{C})$ 
37  |   foreach  $c \in \overline{C}_{(1/2, 3/4)} \cap C_B$  do // Step 7
38  |   |   |    $\text{Schedule}(c, i, 0), i \leftarrow i + 1$ 
39  |   while  $|\overline{M}_H| \geq 2$  AND  $|\overline{C}_{\geq 3/4}| \geq 2$  do // Step 8
40  |   |   |   |    $(a, b) \leftarrow \{\text{pop}(\overline{M}_H), \text{pop}(\overline{M}_H)\}$ 
41  |   |   |   |    $c_1 \leftarrow \text{coalesce}(\{\text{pop}(\overline{C}_{\geq 3/4} \cap \overline{C}_B)\}, \{\text{pop}(\overline{C}_{\geq 3/4})\})$ 
42  |   |   |   |    $c_2 \leftarrow \text{coalesce}(\{\text{pop}(\overline{C}_{\geq 3/4} \cap \overline{C}_B)\}, \{\text{pop}(\overline{C}_{\geq 3/4})\})$ 
43  |   |   |   |    $(\hat{c}_1, \check{c}_1) \leftarrow \{\text{split2}(c)\}$ 
44  |   |   |   |    $\text{Reschedule}(b, 3/2 - p_m(b)), \text{Schedule}(\check{c}_2, b, 0)$ 
45  |   |   |   |    $\text{Schedule}(\hat{c}_1, a, 3/2 - p(\hat{c}_1)), \text{Schedule}(\hat{c}_1, i, 0)$ 
46  |   |   |   |    $\text{Schedule}(\hat{c}_2, i, 3/2 - p(\hat{c}_2)), i \leftarrow i + 1$ 
47  |   |   |   |   if  $|\overline{M}_H| = 0$  then
48  |   |   |   |   |    $\text{Algorithm 8 } (\overline{C})$ 
49  |   if  $|\overline{M}_H| \geq 2$  OR  $|\overline{C} \setminus C_B| = 0$  then // Step 9
50  |   |   |   |   foreach  $c \in \overline{C}$  do
51  |   |   |   |   |    $\text{Schedule}(c, i, 0), i \leftarrow i + 1$ 
52  |   if  $|\overline{M}_H| = 1$  then // Step 10
53  |   |   |   |    $(a) \leftarrow \{\text{pop}(\overline{M}_H)\}, c \leftarrow \{\text{pop}(\overline{C} \setminus C_B)\}, (\hat{c}, \check{c}) \leftarrow \{\text{split3}(c)\}$ 
54  |   |   |   |    $\text{Reschedule}(a, 3/2 - p_m(a)), \text{Schedule}(\hat{c}, a, 0)$ 
55  |   |   |   |    $\text{Algorithm 8 } (\overline{C})$ 
56  |   return  $(\sigma, t)$ 

```