Assigning and Scheduling Generalized Malleable Jobs under Subadditive or Submodular Processing Speeds

Dimitris Fotakis  
National Technical University of Athens  
fotakis@cs.ntua.gr

Jannik Matuschke  
KU Leuven  
jannik.matuschke@kuleuven.be

Orestis Papadigenopoulos  
The University of Texas at Austin  
papadig@cs.utexas.edu

March 29, 2022

Abstract

Malleable scheduling is a model that captures the possibility of parallelization to expedite the completion of time-critical tasks. A malleable job can be allocated and processed simultaneously on multiple machines, occupying the same time interval on all these machines. We study a general version of this setting, in which the functions determining the joint processing speed of machines for a given job follow different discrete concavity assumptions (subadditivity, fractional subadditivity, submodularity, and matroid ranks). We show that under these assumptions the problem of scheduling malleable jobs at minimum makespan can be approximated by a considerably simpler assignment problem. Moreover, we provide efficient approximation algorithms for both the scheduling and the assignment problem, with increasingly stronger guarantees for increasingly stronger concavity assumptions, including a logarithmic approximation factor for the case of submodular processing speeds and a constant approximation factor when processing speeds are determined by matroid rank functions. Computational experiments indicate that our algorithms outperform the theoretical worst-case guarantees.

1 Introduction

The use of parallelization, i.e., the simultaneous processing of a job on multiple machines to expedite its completion, is wide-spread in task scheduling systems, where careful planning is crucial to reduce the overall makespan of time-critical schedules. In computing, the advent of massive parallelism has enabled or significantly facilitated a diverse range of applications including Cholesky factorization (Dongarra et al., 1998), molecular simulation (Bernard et al., 1999), web search index update (Wu and Loiseau, 2015), or training neural networks (Fujiwara et al., 2018). Also in the context of production and logistics, parallelism is used in a wide variety of operational task scheduling applications, such as quay crane allocation in naval logistics (Imai et al., 2008; Blazewicz et al., 2011), workforce assignment in production (Dolgui et al., 2018), recharging electrical vehicles (Nguyen et al., 2018), textile production (Serafini, 1996; Mourtos et al., 2021), disaster relief operations (van der Ster, 2010), or cleaning activities on trains (Bartolini et al., 2017).

A frequently used model for capturing parallelization in scheduling, including many of the aforementioned examples, is that of malleable or moldable jobs (Du and Leung, 1989). At a conceptual level, a malleable job can be assigned to an arbitrary subset of the available machines to be processed non-preemptively and in unison, i.e., with the same starting and completion time on each of
the allocated machines. Importantly, the malleable scheduling model allows the scheduler to decide on the degree of parallelization for each individual job by choosing the set of machines to which it is assigned, in contrast to non-malleable parallel machine models in which each task is assigned to a single machine.

In this work, we study a new generalized model of malleable scheduling that significantly goes beyond the identical machine setting that has been predominantly studied in literature before. We show that, under natural discrete concavity assumptions on the functions determining the processing speed of machine sets (subadditivity, fractional subadditivity, submodularity, matroid ranks), increasingly strong approximation results can be obtained for this generalized model. Our first main result shows how the task of finding a schedule of small makespan can be approximated by the significantly simpler task of finding an assignment of jobs to machine sets. We further provide efficient approximation algorithms for this latter assignment problem, which via the first result also translate into approximation algorithms for the scheduling problem (see Table 1 for a summary of the resulting approximation factors). Finally, we demonstrate the applicability of our approach in a computational study for this latter heuristic and the aforementioned assignment-to-schedule transformation.

1.1 The Malleable Scheduling Model

Consider a set of jobs \( J \) to be assigned on a set of machines \( M \). Each job \( j \in J \) is equipped with a processing time function \( f_j : 2^M \to \mathbb{R}_{\geq 0} \) that specifies the time \( f_j(S) \) needed for the completion of \( j \) when assigned to a subset of machines \( S \subseteq M \). A schedule consists of an assignment \( S = (S_j)_{j \in J} \) that specifies a non-empty set of machines \( S_j \subseteq M \) for each job \( j \in J \), and of a starting time vector \( t = (t_j)_{j \in J} \). In such a schedule \( (S, t) \), the machines in \( S_j \) all jointly process job \( j \) for the same interval of time, starting at time \( t_j \) and finishing at time \( t_j + f_j(S_j) \). At any given moment, each machine can only process a single job, i.e., there are no jobs \( j, j' \in J \) with \( S_j \cap S_{j'} \neq \emptyset \) and \( t_j < t_{j'} < t_j + f_j(S_j) \). The makespan of a schedule \( (S, t) \) is \( C_{\text{max}} := \max_{j \in J} t_j + f_j(S_j) \), i.e., the time when all jobs are completed.

We are interested in finding schedules of minimum makespan. Throughout this article, we will call this the SCHEDULING problem. Related to this problem is the ASSIGNMENT problem, which asks for an assignment \( S \) minimizing the maximum load \( L(S) := \max_{i \in M} \sum_{j \in J : i \in S_j} f_j(S_j) \) over all machines. Note that the latter problem does not include information on the starting time of jobs and that the maximum load in any assignment \( S \) is a lower bound on the makespan of any schedule \( (S, t) \) using that same assignment. We will discuss the relation between SCHEDULING and ASSIGNMENT further in Section 1.3 below.

Rather surprisingly, and despite the significant theoretical and practical interest in the model, most of the work on scheduling malleable jobs considers the case of identical machines. Note that in this case, the processing time of a job only depends on the number of allocated machines, i.e., \( f_j(S) = \tilde{f}_j(|S|) \) for some function \( \tilde{f}_j : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0} \). In this setting, it is commonly assumed that the processing time \( \tilde{f}_j(k) \) is non-increasing in the number of machines \( k \), whereas the product \( k \cdot \tilde{f}_j(k) \), i.e., the number of machine hours needed to complete the job, is non-decreasing. The latter assumption is known as the monotone work assumption. It accounts for the fact that, in most practical applications, the amount of work needed to complete a job does not decrease when using parallelization (rather, parallelization often incurs a communication/coordination overhead) and that the combined processing speed of a set of machines is not larger than the sum of their individual processing speeds, i.e., no synergies are obtained from combining multiple machines.

The identical machine case is very well understood from an optimization perspective. Constant-factor approximation algorithms for the problem are known since the work of Turek et al. (1992). A
line of work establishing improved approximation results (Mounie et al., 1999; Jansen and Porkolab, 2002; Mounié et al., 2007) recently culminated in a polynomial-time approximation scheme, implicit in the combined work of Jansen and Thöle (2010) and Jansen and Land (2018). Thus, the minimum makespan can be approximated efficiently up to arbitrary precision in the identical machine case, matching the complexity of the corresponding non-malleable scheduling problem known as $P||C_{\text{max}}$ (Hochbaum and Shmoys, 1987).

Literature on malleable scheduling beyond the identical machine setting is scarce. A model similar to malleable scheduling in which unrelated machines have been studied is that of splittable jobs. In this model, each job can be split into arbitrary fractions that can be distributed on different machines (without any requirement for synchronization). The time required for processing a fraction $x_{ij} > 0$ of job $j$ on machine $i$ is given by $s_{ij} + p_{ij}x_{ij}$, where $s_{ij}$ is a fixed setup time and $p_{ij}$ is a processing time factor. Correa et al. (2015) provide a $(1 + \varphi)$-approximation algorithm for this setting, where $\varphi \approx 1.62$ is the golden ratio, and show that this factor is tight for a natural LP relaxation of the problem. In the malleable model, non-identical machines have only recently been studied for the setting of speed-implementable processing-time functions (Fotakis et al., 2019). In this setting, each machine $i$ is associated with an unrelated “speed” $s_{ij}$ for each job $j$, and processing time is a function of the total allocated speed, i.e., $f_j(S) = \bar{f}_j(\sum_{i\in S} s_{ij})$ for some function $\bar{f}_j : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. The monotone work assumption naturally extends to this setting by requiring $s \cdot \bar{f}_j(s)$ to be non-decreasing. Fotakis et al. (2019) devised an LP-based 3.16-approximation for this setting.

A significant limitation inherent to models as the two discussed above, in which the contribution of a machine to a job can be expressed by a single number, is their inability to capture interaction effects caused by different combinations of machines when processing the same job. Complicated interdependencies may arise, e.g., from combining machines of different types or characteristics, from operational requirements, or from the local topology of the processing platform. In computing, e.g., modern heterogeneous parallel systems are typically consist of CPUs, GPUs, and I/O nodes, featuring different architectures and memory restrictions (Bleuse et al., 2017). To make optimal use of such systems, scheduling algorithms need to take the different architectures and the topology of the underlying communication network into consideration (Bampis et al., 2020). Another example is workforce assignment in production, where the assumption of identically-skilled workers is inappropriate in environments characterized by short-term contracts and an absence of standardized training (Nakade and Nishiwaki, 2008), in sheltered work centers for the disabled (Borba and Ritt, 2014), or in teams combining highly-trained specialized workers (Walter and Zimmermann, 2016). These and similar situations require models capturing the combinatorial interaction effects of machines or workers assigned to the same task.

### 1.2 Generalized Malleable Scheduling with Concave Processing Speeds

In this work, we investigate a generalized setting in which processing times depend on the actual set of machines used for a job. This model not only allows for machines to be distinctively well-suited for different jobs but it also captures combinatorial relations between different machines, such as the incompatibility to jointly process the same job.

To extend the monotone work assumption to the set function context, we define $g_j(S) := 1/f_j(S)$ for every $j \in J$ and $S \subseteq M$ (as jobs have to be allocated to non-empty machine set, we assume w.l.o.g. that $f_j(\emptyset) = \infty$ and $g_j(\emptyset) = 0$ for all $j \in J$). Intuitively, $g_j(S)$ corresponds to the joint processing speed with which the machines in $S$ process job $j$, which is represented by one (normalized) unit of work in the numerator. We thus refer to the function $g_j : 2^M \to \mathbb{R}_{\geq 0}$ as the speed function for job $j \in J$. Unless noted otherwise, we will assume that each $g_j$ is given by a
value oracle that, given a set \( S \), returns the value \( g_j(S) \).

Recall that the monotone work assumption for the identical machine setting implies the absence of synergies in processing speed when allocating more machines to a job. To impose a similar assumption on the processing speed functions \( g_j \), we consider the following four classes of functions from the field of discrete convexity:

- **subadditivity**: A function \( g : 2^M \rightarrow \mathbb{R}_{\geq 0} \) is subadditive if \( g(S \cup T) \leq g(S) + g(T) \) for all \( S, T \subseteq M \).

- **fractionally subadditivity**: A function \( g : 2^M \rightarrow \mathbb{R}_{\geq 0} \) is fractionally subadditive (also known as XOS in the literature), if \( g(T) \leq \sum_{S \subseteq T} \alpha(S)g(S) \) for all \( T \subseteq M \) and for all \( \alpha : 2^M \rightarrow [0, 1] \) that fulfill \( \sum_{S \subseteq T, i \in S} \alpha(S) \geq 1 \) for all \( i \in T \).

- **submodularity**: A function \( g : 2^M \rightarrow \mathbb{R}_{\geq 0} \) is submodular if \( g(S \cup T) + g(S \cap T) \leq g(S) + g(T) \) for all \( S, T \subseteq M \). This definition is equivalent to the following diminishing returns property: \( g(S \cup \{i\}) - g(S) \geq g(T \cup \{i\}) - g(T) \) for all \( S \subseteq T \subseteq M \) and \( i \in M \setminus T \).

- **matroid ranks**: A matroid on the ground set \( M \) is a non-empty set family \( \mathcal{F} \subseteq 2^M \) such that (i) \( T \in \mathcal{F} \) implies \( S \in \mathcal{F} \) for all \( S \subseteq T \) and (ii) for every \( S, T \in \mathcal{F} \) with \( |S| < |T| \) there is an \( i \in T \setminus S \) such that \( S \cup \{i\} \in \mathcal{F} \). The rank function of a matroid \( \mathcal{F} \) is defined by \( r(S) := \max_{S \subseteq T, S \in \mathcal{F}} |S| \) for \( S \subseteq M \). A function \( g : 2^M \rightarrow \mathbb{R}_{\geq 0} \) is a scaled matroid rank if \( g(S) = \alpha r(S) \) for all \( S \subseteq M \), where \( \alpha > 0 \) and \( r \) is the rank function of some matroid.

Each of these classes captures the concepts of “no synergies” or “diminishing returns” for set functions with an increasing degree of restrictiveness (i.e., subadditive functions constitute a superclass of fractionally subadditive functions, which form a superclass of submodular functions etc.). All four classes have been widely studied, receiving particular attention in operations research and economics literature for their applications in diverse fields such as auction markets (Candogan et al., 2015), cooperative game theory (Bondareva, 1963; Feige, 2009), inventory management (Levi et al., 2014), machine learning (Bach et al., 2013), or queuing theory (Ahn and Lewis, 2013); also see the survey by Iwata (2008) and the textbook by Murota (2003).

We remark that it is natural to assume processing speeds to be monotone (i.e., \( S \subseteq T \) implies \( g(S) \leq g(T) \)). However, most of our results work without this assumption, and we will specifically point out when monotonicity is required. Before we elaborate our results for the generalized malleable scheduling model under the aforementioned discrete concavity assumptions, we highlight some computational and practical challenges inherent to malleable scheduling, and the resulting distinction of assigning and scheduling.

### 1.3 Assigning vs. Scheduling and Practical Challenges

A particular challenge in malleable scheduling, both in terms of computation and practical implementation of schedules, is the need to coordinate schedules on different machines to allow jobs to be processed in unison. In particular, this feature of malleable scheduling motivates the distinction between assignment and schedule.

Recall that the ASSIGNMENT problem asks for an assignment minimizing the maximum machine load (ignoring starting times), whereas the SCHEDULING problem asks for a schedule of minimum makespan (including starting times). In classic non-malleable scheduling these two problems are equivalent, as any assignment can be turned into a schedule whose makespan equals the maximum machine load, just by letting each machine process the jobs assigned to it in arbitrary order. This is no longer true in malleable scheduling, as starting times for jobs cannot be determined independently.
Figure 1: (a) Original schedule (x-axis is the time and y-axis the set of machines); (b) Delay propagating through a schedule: A unit-time delay in the upper-left job causes a cascade of delays.

on each machine. Indeed, it is easy to construct examples of assignments $S$ such that any schedule $(S, t)$ using this assignment has a makespan of at least $|J|/2$ times the maximum machine load, independent from assumptions on the processing speed functions; see Appendix A.1.

From a computational point of view, the need for synchronization seems to make finding good (i.e., low-makepan) schedules considerably harder than the task of finding good assignments. Indeed, the Assignment problem has a natural and well-structured IP formulation—based on binary variables $x(S, j)$ indicating the set of machines $S$ assigned to job $j$—that is amenable to column generation approaches; see Appendix A.2 for details. Straightforward IP formulations for the Scheduling problem, on the other hand, such as time-indexed formulations, not only incur a considerable blow-up in number of variables and constraints, but also appear to be much less well-structured.

Finally, executing schedules that rely heavily on parallelization requires frequent regrouping and well-organized synchronization of sets of machines to process different jobs in unison, creating a managerial challenge. In particular, such schedules are susceptible to delays that can easily propagate throughout the schedule. In fact, due to the interdependence of schedules on different machines, a delay in a single job can make it necessary to reschedule jobs on all machines, even those not involved in processing the delayed job; see Fig. 1. Similarly, small delays of multiple jobs, even when processed on disjoint sets of machines, can accumulate to a significant extension of the makespan.

1.4 Our Results

In the following, we give an overview of our results and point to the corresponding sections where they are discussed in detail.

1.4.1 Turning Assignments Into Schedules

Our first result is a transformation that turns any given assignment into a schedule (with a slightly modified assignment), addressing both the computational and practical challenges described in Section 1.3. We show that the resulting schedule incurs only a bounded increase in makespan compared to the original assignment. Thus, the Scheduling problem can be approximated via the simpler Assignment problem. The resulting schedules moreover have a beneficial structure that significantly facilitates their implementation: only the first job processed on any machine is processed across multiple machines, all later jobs are handled exclusively by a single machine; see Fig. 2. This makes the schedule easy to execute and robust, as delays in individual jobs can only effect other jobs on the same machine but cannot propagate through the schedule.
For fractionally subadditive speed functions, the increase in makespan is bounded by a factor of $2e/(e - 1) \approx 3.16$ in the worst-case. This carries over to subadditive functions at a logarithmic (in the number of machines) loss via a well-known pointwise approximation result. Our transformation is based on the Bondareva-Shapley theorem (Bondareva, 1963), a result from cooperative game theory that guarantees the existence of the solution to an inequality system defined by a fractionally subadditive function. While, in general, finding such a solution may require to solve an LP of exponential size, it can be found by a simple greedy algorithm in the case of submodular functions. Thus, the entire procedure can be carried out in polynomial time for submodular processing speeds (with the same worst-case guarantee of 3.16). These results are discussed in detail in Section 2. Moreover, in Section 5, we report on the results of computational experiments that demonstrate that applying the transformation in practice results in a much smaller loss than the theoretical worst-case guarantee (approximately 1.17 on average and in no case larger than 2), making it possible to find near-optimal solutions to Scheduling via the simpler Assignment problem.

### 1.4.2 Approximating the Assignment Problem

In the second part of the paper, we turn our attention to devising efficient approximation algorithms for both the Assignment and Scheduling problem. For the special case where processing speeds are defined by scaled matroid ranks, we show that a 4-approximation for Assignment can be obtained by making use of an appropriate intersection of two matroids. Using insights from graph orientation, this can be turned into a 5-approximation algorithm for Scheduling. These results are discussed in Section 3.

In Section 4, we present a greedy heuristic for approximating both Assignment and Scheduling, which applies to various classes of speed functions ranging from subadditive to submodular. The algorithm runs in phases, where, in each phase, a constant fraction of not-yet-assigned jobs is scheduled within a multiplicative factor of the target makespan. To achieve this, our heuristic makes critical use of the structural insights derived in the earlier transformation result, concurrently executing two assignment subroutines based on the Generalized Assignment problem (Shmoys and Tardos, 1993) and Welfare Maximization for the class of speed functions at hand (Schrijver, 2003; Feige, 2009), respectively.

The algorithm gives a polylogarithmic and a logarithmic approximation guarantee for subadditive and fractionally subadditive processing speeds, respectively, assuming access to a demand oracle. Further, the algorithm can be implemented efficiently using a value oracle for the case of (not necessarily monotone) submodular functions. As we show, access to a stronger demand oracle is necessary in the former case, as any algorithm obtaining a non-trivial approximation factor for Assignment with fractionally subadditive processing speeds given by a value oracle would require an exponential number of queries to this oracle. Finally, we provide improved approximation
Speed Functions & Transformation & Approximation (Distinct Jobs) & Approximation (Identical Jobs)

| Speed Functions          | Transformation Assignment → Schedule | $O(\log m)^a$ | $O(\log m \log \min\{n, m\})^{b,c}$ | $O(\log m)^{b,c}$ |
|--------------------------|---------------------------------------|--------------|-------------------------------------|-------------------|
| Subadditive              |                                       | 3.16         |                                    | 0                 |
| Fractionally Subadditive |                                       | 3.16         |                                    | 0                 |
| Submodular               |                                       | 3.16         |                                    | 0                 |
| Scaled Matroid Ranks     |                                       | 3.16         |                                    | 0                 |

Table 1: Summary of results. For each type of speed function, we report the maximum increase in makespan when transforming an assignment into a schedule (Section 2) and the factor of the approximation algorithms for the Assignment problem with distinct and identical jobs, respectively (Sections 3 and 4). We denote by $n$ and $m$ the number of jobs and machines, respectively. Results marked with $a$ are algorithmic but without polynomial run-time; results marked with $b$ require monotonicity; results marked with $c$ require a demand oracle instead of a value oracle.

guarantees for the case of identical jobs, namely, a logarithmic approximation for subadditive and a constant approximation for fractionally subadditive speed functions. In Table 1 we present a summary of our results.

In Section 5 we further discuss computational results on the empirical performance of our heuristic for submodular processing speeds, showing that it significantly outperforms its worst-case guarantees for a number of problem instances.

## 2 Transforming Assignments into Schedules

In this section, we present a transformation that turns any given assignment of jobs to machine sets into a schedule, at a bounded increase in makespan, depending on the assumptions on processing speeds, thus bounding the gap between optimal values for Assignment and Scheduling problem. The transformation is based on the following concept of a well-structured assignment.

We say an assignment $S$ is well-structured if for every $i \in M$ there is at most one $j \in J$ with $i \in S_j$ and $|S_j| > 1$, i.e., every machine shares at most one job with other machines. A well-structured assignment can be turned into a schedule whose makespan equals the maximum machine load: At time 0, let every machine process its only shared job (if it exists), then process the remaining jobs assigned to each machine in arbitrary order on that machine. Thus it suffices to show that arbitrary assignments can be transformed into well-structured assignments at a bounded increase in makespan. We first establish that this is possible for fractionally subadditive functions, in the following theorem.

**Theorem 1.** Assume that all processing speed functions are fractionally subadditive. For any assignment $S$, there exists a well-structured assignment $S'$ with $L(S') \leq \frac{2e}{e-1} L(S)$.

The proof of Theorem 1 is based on an interesting connection to cooperative game theory and uses the Bondareva-Shapley theorem (Bondareva, 1963), which states that cost-sharing games with fractionally subadditive cost functions have a non-empty core. In general, computing such an element of this core may require solving an exponentially sized LP. However, in the case of submodular processing speeds, such an element can be found by a simple greedy algorithm. With this, the entire transformation can be carried out in polynomial time for submodular processing speeds. We hence obtain the following theorem.

**Theorem 2.** Assume that all processing speed functions are submodular. There is an algorithm that, given an assignment $S$, computes in polynomial time a well-structured assignment $S'$ with $L(S') \leq \frac{2e}{e-1} L(S)$.
Before we describe and analyze the transformation procedure that proves Theorems 1 and 2, we also remark that the transformation result for fractionally subadditive functions extends, with a logarithmic loss, to the case that processing speeds are submodular. This immediately follows from the fact that any subadditive function can be pointwise \( \ln(|M|) \)-approximated by a fractionally subadditive function; see Appendix B.1 for details.

**Corollary 3.** Assume that all processing speed functions are subadditive. For any assignment \( \mathbf{S} \), there exists a well-structured assignment \( \mathbf{S}' \) with \( L(\mathbf{S}') \leq \frac{2e}{e-1} \ln(|M|) \cdot L(\mathbf{S}) \).

### 2.1 Proof of Theorem 1

Assume that \( g_j \) is fractionally subadditive for all \( j \in J \), let \( \mathbf{S} \) be an assignment, and \( C := L(\mathbf{S}) \). The fractional subadditivity of the processing speed functions implies the following lemma.

**Lemma 4.** There exists \( \Delta \in \mathbb{R}^{M \times J} \) such that

1. \( \sum_{i \in S_j} \Delta_{ij} = g_j(S_j) \) for all \( j \in J \),
2. \( \sum_{i \in T} \Delta_{ij} \leq g_j(T) \) for all \( j \in J \) and \( T \subseteq S_j \).

**Proof.** Noting that constraints 1 and 2 describe, for each \( j \in J \), the core of a cost-sharing game with player set \( S_j \) and cost function \( g_j \), Lemma 4 follows immediately from the Bondareva-Shapley theorem, which guarantees that the core of a game with fractionally subadditive cost functions is non-empty. For completeness, a direct proof of the lemma is given in Appendix B.2.

Using the coefficients \( \Delta_{ij} \) given by Lemma 4, let \( S_j^+ := \{ i \in S_j : \Delta_{ij} > 0 \} \) for \( j \in J \) and consider the following feasibility LP with decision variables \( x_{ij} \) for every \( j \in J \) and \( i \in S_j^+ \).

\[
\begin{align*}
\sum_{i \in S_j^+} x_{ij} &\geq 1 \quad \forall j \in J \\
\sum_{j \in J : i \in S_j^+} \frac{1}{\Delta_{ij}} x_{ij} &\leq C \quad \forall i \in M \\
&x \geq 0
\end{align*}
\]  

(LP1)

We first observe that \((LP1)\) has a feasible solution.

**Lemma 5.** \((LP1)\) has a feasible solution.

**Proof.** Define \( \tilde{x} \) by \( \tilde{x}_{ij} := \frac{\Delta_{ij}}{g_j(S_j)} \) for \( j \in J \) and \( i \in S_j^+ \). Note that \( \sum_{i \in S_j^+} \frac{1}{g_j(S_j)} \tilde{x}_{ij} = \sum_{i \in S_j^+} \frac{\Delta_{ij}}{g_j(S_j)} = 1 \) by constraint 1 of Lemma 4. Moreover, \( \sum_{j \in J : i \in S_j^+} \frac{1}{\Delta_{ij}} \tilde{x}_{ij} = \sum_{j \in J : i \in S_j^+} \frac{1}{g_j(S_j)} \leq C \) by definition of \( \tilde{x} \) and the fact that the maximum machine load of \( \mathbf{S} \) is \( C \). Hence \( \tilde{x} \) is a feasible solution to \((LP1)\).

Now consider an extreme point solution \( x' \) to \((LP1)\). We will turn it into a well-structured assignment. To this end, consider the support graph \( G(x') \) of \( x' \) on the node set \( J \cup M \), which has an edge \( \{i, j\} \) for every \( j \in J \) and \( i \in S_j^+ \) with \( x'_{ij} > 0 \). Noting that \((LP1)\) has the same structure as the assignment LP used for \( R||C_{\text{max}} \), the following lemma follows from the classic result by Lenstra et al. (1990); see Appendix A.3 for details.

**Lemma 6.** The graph \( G(x') \) has an orientation such that any node has an in-degree of at most 1.
Thus fix an orientation of $G(x')$ with in-degree at most one. For each job $j \in J$, let $i_j \in S_j$ denote the unique machine such that $(i_j,j)$ is an in-edge of node $j$ (if such a machine exists), and let $\bar{S}_j$ the set of machines $i \in S_j$ such that there is an edge $(j,i)$ in the oriented graph. We construct a well-structured assignment $S'$ as follows. For each job $j \in J$, let $S'_j = \{i_j\}$ if $i_j$ exists and $x'_{i,j} > 1/2$ (“Type 1 assignment”) and let $S'_j = \bar{S}_j$ otherwise (“Type 2 assignment”). Note that, because every node has in-degree at most 1, any machine can be assigned to at most one job via a Type 2 assignment. Hence $S'$ is well-structured.

We will show that the maximum load in this assignment is bounded by $4C$ and then explain how to further modify it so as to reduce the load to the value guaranteed in the statement of Theorem 1.

We first argue that the load incurred by Type 1 assignments is bounded by $4C$.

**Lemma 7.** For $i \in M$ let $J^1_i := \{j \in J : i_j = i, \ x'_{i,j} > 1/2\}$. Then $\sum_{j \in J^1_i} f_j(\{i\}) \leq 2C$.

**Proof.** The lemma follows from

$$\sum_{j \in J^1_i} f_j(\{i\}) = \sum_{j \in J^1_i} \frac{1}{g_j(\{i\})} \leq \sum_{j \in J^1_i} \frac{1}{\Delta_{ij}} \leq 2 \sum_{j \in J^1_i} \frac{1}{\Delta_{ij}} x'_{i,j} \leq 2C,$$

where the first inequality follows from constraint 2 in Lemma 4, the second inequality follows from the fact that $x'_{i,j} > 1/2$ for all $j \in J^1_i$, and the final inequality follows from feasibility of $x'$.

Similarly, also the load incurred by Type 2 assignments can be bounded by $2C$.

**Lemma 8.** Let $j \in J$ with $S'_j = \bar{S}_j$. Then $f_j(\bar{S}_j) \leq 2C$.

**Proof.** Note that $S'_j = \bar{S}_j$ implies that $\sum_{i \in S_j} x'_{i,j} \geq 1/2$ by construction of $S'$. We conclude that

$$g_j(\bar{S}_j) \geq \sum_{i \in \bar{S}_j} \Delta_{ij} \geq \frac{1}{C} \sum_{i \in \bar{S}_j} x'_{i,j} \geq \frac{1}{2C},$$

where the first inequality follows again from constraint 2 in Lemma 4 and the second inequality follows from $x'_{i,j} \leq \Delta_{ij}$ by feasibility of $x'$.

Because any machine participates in at most one Type 2 assignment, Lemmas 7 and 8 already imply that the load of any machine in the well-structured assignment $S'$ is bounded by $4C$. The maximum machine load can be further reduced by modifying $S'$ as follows: We reassign all jobs $j \in J$ that are assigned via a Type 2 assignment to $\bar{S}_j$ as follows. For each machine $i \in \bar{S}_j$, let $\ell_i := \sum_{j' \in J^1_i} \frac{1}{\sum_{j' \neq j'} x'_{i,j'}}$ be the load of machine $i$ due to jobs in $J^1_i$ in the LP solution. For $\theta > 0$, let $T_j(\theta) := \{i \in \bar{S}_j : \ell_i \leq \theta\}$. Instead of assigning $j$ to $\bar{S}_j$, assign it to $T_j(\theta^*)$, where $\theta^*$ is a minimizer of $\max_{i \in T_j(\theta)} 2\ell_i + f_j(T_j(\theta))$.

**Lemma 9.** Let $T$ be the assignment defined by $T_j = T_j(\theta^*)$ if $S'_j = \bar{S}_j$ and $T_j = \{i_j\}$ otherwise. Then $T$ is well-structured and $L(T) \leq \frac{2e}{e-1} L(S)$.

The proof of Lemma 9 follows the lines of the analysis for the approximation algorithm for malleable scheduling with speed-implementable processing times (Fotakis et al., 2019) and is presented in Appendix B.3. This completes the proof of Theorem 1.
2.2 Proof of Theorem 2

Note that the proof of Theorem 1 discussed above in fact describes an algorithm for turning a given assignment into a well-structured assignment: (i) find the coefficient vector $\Delta$ as guaranteed by Lemma 4, (ii) compute an extreme point $x'$ solution to (LP1), (iii) orient the support graph $G(x')$ so that every node has in-degree at most 1, (iv) construct the assignment $S'$, (v) compute optimal values $\theta_j^*$ for every $j \in J$ that is assigned via a Type 2 assignment, and (vi) reassign those jobs to the corresponding sets $T_j(\theta_j^*)$.

For steps (ii)-(iv) and (vi) it is easy to see that they can be carried out in polynomial time, even independently from any assumptions on the functions $g_j$. Also determining the values $\theta_j^*$ in step (v) can be done in polynomial time, as the sets $T_j(\theta)$ only change at $\theta \in \{ \ell_i : i \in S_j \}$ and thus only $|S_j| + 1$ different values have to be considered. It remains to argue that step (i) can be carried out efficiently when $g_j$ is submodular for each $j \in J$.

Indeed, for submodular $g_j$, the constraints in Lemma 4 describe a polymatroid for each $j \in J$. Thus, a feasible solution $\Delta$ can be found by the following simple greedy algorithm. Fix any arbitrary order $\preceq$ of the machines and for each $S \subseteq M$ let $\pi_i(S) := \{ i' \in S : i' \preceq i \}$. For each $j \in J$ and $i \in S_j$, define $\Delta_{ij} := g_j(\pi_i(S_j)) - g_j(\pi_i(S_j) \setminus \{ i \})$. It is not hard to see that $\sum_{i \in S_j} \Delta_{ij} = g_j(S_j)$ for all $j \in J$ by construction. Moreover, $\sum_{i \in T} \Delta_{ij} \leq \sum_{i \in T} g_j(\pi_i(S_j \cap T)) - g_j(\pi_i(S_j \cap T) \setminus \{ i \}) = g_j(T)$ for all $T \subseteq S_j$, where the inequality follows from submodularity of $g_j$. Hence, the greedily constructed coefficients $\Delta_{ij}$ fulfill the constraints of Lemma 4. This concludes the proof of Theorem 2.

3 Approximation for Scaled Matroid-Rank Processing Speeds

In this section, we consider the case where each job $j \in J$ is associated with a matroid $F_j$ on $M$ and a quota $q_j \in \mathbb{R}_{\geq 0}$. The processing speed function for job $j \in J$ is defined by $g_j(S) := r_j(S)/q_j$, where $r_j$ is the rank function of $F_j$.

We describe an algorithm which, given an instance of the problem and a target load $C$, either computes an assignment of maximum load at most $4C$ and a schedule of makespan at most $5C$, or asserts that no assignment with maximum load at most $C$ exists. Via standard arguments (see Appendix A.3), such a decision procedure implies a 4-approximation for ASSIGNMENT and a 5-approximation for SCHEDULING, respectively.

**Theorem 10.** There exists a polynomial-time 4-approximation algorithm for ASSIGNMENT and a 5-approximation algorithm for SCHEDULING when all processing speed functions are scaled matroid ranks. If all jobs are furthermore identical, the approximation factor improves to 2 for ASSIGNMENT and 3 for SCHEDULING, respectively.

3.1 The Algorithm

The algorithm splits the set of jobs in two classes: those that can be processed within time $C$ on a single machine, and those that require multiple machines to do so. It then solves the problem separately for each of the two classes, solving a classic scheduling problem for the first class and a matroid intersection problem and graph orientation problem for the second.

**Partitioning the job set**

We first partition the set of jobs into two classes. To this end, define $t_j := \lceil q_j/C \rceil$ for all $j \in J$ as well as $J_1 := \{ j \in J : t_j = 1 \}$ and $J_2 := J \setminus J_1$.
Step 1: Single-Machine Assignments for \(J_1\)

The first step of the algorithm handles the jobs in \(J_1\) by constructing an instance of the classic problem of makespan minimization on unrelated machines (with non-malleable jobs). For \(i \in M\) and \(j \in J_1\) let \(p_{ij} = q_j\) if \(\{i\} \in F_j\), and \(p_{ij} = \infty\) otherwise. We set up the assignment LP for the instance of the non-malleable unrelated machine scheduling problem \(R||C_{\text{max}}\) with machine set \(M\), job set \(J_1\), and processing times \(p_i\).

\[
\sum_{i \in M} x_{ij} = 1 \quad \forall j \in J_1 \tag{LP2}
\]

\[
\sum_{j \in J_1} q_j x_{ij} \leq C \quad \forall i \in M
\]

\[x \geq 0\]

Notice that the fact that \(t_j = 1\) implies that \(q_j \leq C\) for any job \(j \in J_1\). If the LP is feasible, we obtain an assignment \(S^1\) for the jobs in \(J_1\) with maximum load at most \(2C\) using the rounding algorithm of Lenstra et al. (1990) (see Appendix A.3 for details). Otherwise, if the LP is infeasible, then we can conclude by Lemma 11 (see Section 3.2) that there is no assignment for with makespan at most \(C\) for the given instance and stop the algorithm.

Step 2: Assignment via Matroid Intersection for \(J_2\)

In the second step of the algorithm, we define two matroids with ground set \(M \times J_2\) as follows. First, we define \(F'\) by

\[
F' := \{S \subseteq M \times J_2 : \{i \in M : (i, j) \in S\} \in F_j, \text{ and } |\{i \in M : (i, j) \in S\}| \leq t_j \text{ for all } j \in J_2\}.
\]

Note that \(F'\) is the direct sum of the matroids \(F_j\) for all \(j \in J_2\), each truncated at cardinality \(t_j\), respectively. In particular, \(F'\) is itself a matroid. Second, we define \(F''\) by

\[
F'' := \{S \subseteq M \times J_2 : |\{j \in J_2 : (i, j) \in S\}| \leq 2 \text{ for all } i \in M\},
\]

i.e., \(F''\) is the partition matroid that allows at most two copies of each machine to be selected. Let \(B\) be a maximum cardinality common independent set in \(F' \cap F''\) (such a set can be computed using a polynomial-time algorithm for matroid intersection (Schrijver, 2003)). If \(|B| < \sum_{j \in J_2} t_j\), then by Lemma 12 below, we can conclude that there is no assignment of makespan at most \(C\) for the given instance and stop the algorithm. Otherwise, let \(S^2\) be the assignment for the jobs in \(J_2\) induced by \(B\) via \(S^2_j := \{i \in M : (i, j) \in B\}\).

Step 3a: Merging the Assignments

If the algorithm did not terminate in either of the preceding steps with a negative answer, it combines the two assignments \(S^1\) and \(S^2\) into an assignment for all jobs in \(J\) and returns this assignment. In the proof of Theorem 13 below, we show that the returned assignment has maximum machine load of at most \(4C\). This assignment could be turned into a schedule of makespan at most \(8e/(e-1) \cdot C\) via Theorem 2. However, the more careful procedure described below shows that we can obtain a (well-structured) schedule of makespan at most \(5C\) by a slightly different way.
Lemma 11. If there is an assignment of maximum load at most \( C \), then \( \text{(LP2)} \) has a feasible solution.

Proof. Let \( S \) be an an assignment with maximum load at most \( C \). We first observe that, without loss of generality, \( S_j \in \mathcal{F}_j \) for each \( j \in J \) (otherwise some machines can be removed from \( S_j \) without increasing the processing time). We construct a feasible solution to \( \text{(LP2)} \) as follows. For any \( i \in M \) and \( j \in J \), we set \( x'_{ij} = \frac{1}{|S_j|} \) if \( i \in S_j \), and \( x'_{ij} = 0 \) otherwise. For this assignment, the first and third set of constraints of \( \text{(LP2)} \) are trivially satisfied. Finally, for the second set of constraints we obtain:

\[
\sum_{j \in \mathcal{F}_j} q_j x'_{ij} = \sum_{j \in \mathcal{F}_j \cap S_j} \frac{q_j}{|S_j|} = \sum_{j \in \mathcal{F}_j \cap S_j} \frac{q_j}{r_j(S_j)} = \sum_{j \in \mathcal{F}_j \cap S_j} f_j(S_j) \leq C,
\]

for every \( i \in M \), where the second equality follows from our assumption that \( S_j \in \mathcal{F}_j \) and the last inequality by the feasibility of the optimal solution.

The next lemma shows that if the algorithm terminates in Step 2 it correctly concludes that there is no assignment with maximum load \( C \).

Lemma 12. If there is an assignment of makespan at most \( C \), then there is a \( B \in \mathcal{F}' \cap \mathcal{F}'' \) with \(|B| = \sum_{j \in J_2} t_j\).

Proof. Let \( S \) be an an assignment with maximum load at most \( C \). As in the proof of Lemma 11, we assume without loss of generality that \( S_j \in \mathcal{F}_j \).
For $j \in J_2$ and $i \in S_j$, we define $x'_{ij} = t_j/|S_j|$. Consider the following LP:

$$\begin{align*}
\sum_{i \in S_j} x_{ij} &= t_j & \forall j \in J_2 \\
\sum_{j \in J_2: i \in S_j} x_{ij} &\leq 2 & \forall i \in M \\
0 \leq x_{ij} &\leq 1 & \forall j \in J_2, i \in S_j
\end{align*}$$

Note that $x'$ is a feasible solution to the LP. The first type of constraints is trivially satisfied by $x'$, while for the third set of constraints we have $0 \leq x'_{ij} = t_j/|S_j| = \frac{t_j}{r(S_j)} \leq 1$ for each $j \in J_2$ and $i \in M$, since $r(S_j) \geq \lceil q_j/C \rceil = t_j$ in any feasible solution.

Furthermore, note that

$$f_j(S_j) = \frac{q_j}{|S_j|} \geq \frac{(t_j - 1)C}{|S_j|} \geq \frac{C}{2} \cdot \frac{t_j}{|S_j|} = \frac{C}{2} x'_{ij}$$

for all $j \in J_2$ and $i \in S_j$ and, hence, $
\sum_{j \in J_2} x'_{ij} \leq \frac{2}{C} \sum_{j \in J_2} f_j(S_j) \leq \frac{2}{C} \cdot C$ by feasibility of the assignment for makespan $C$.

Notice that the constraint matrix of the above LP is totally unimodular, as it corresponds to the adjacency matrix of a bipartite graph. Thus, since the right hand side of the constraints is integral, we know that the LP has an integral solution $x^*$. Let $S^*_j = \{i \in M : x_{ij} = 1\}$. Note that $S^*_j \subseteq S_j \in F_j$ and $|S^*_j| = t_j$ for all $j \in J_2$. Hence $B = \{(i, j) : j \in J_2, i \in S^*_j\} \in F'$. Furthermore, $|\{j \in J_2 : i \in S^*_j\}| \leq 2$ and, thus, $B \in F''$.

We now show that the assignment constructed in Step 3a has maximum load of at most $4C$, establishing the first part of Theorem 10.

**Theorem 13.** If the algorithm reaches Step 3a, it constructs an assignment of maximum machine load at most $4C$.

**Proof.** Let $S$ be the assignment for all jobs in $J$ arising from combining $S^1$ and $S^2$. Note that $S_j \in F_j$ and $|S_j| \leq t_j$ for all $j \in J_2$ by definition of $F'$ and construction of $S^2$ from $B \in F'$. Thus, $|B| = \sum_{j \in J_2} t_j$ implies that $|S_j| = t_j$ for all $j \in J_2$. Therefore $f_j(S_j) = \frac{q_j}{r_j(S_j)} = \frac{q_j}{t_j} \leq C$ for each $j \in J_2$. Moreover, because $|\{j \in J_2 : i \in S_j\}| \leq 2$ for each $i \in M$ by definition of $F''$ and construction of $S^2$ from $B \in F''$. We conclude that $\sum_{j \in J_2: i \in S_j} f_j(S_j) \leq 2C$ for each $i \in M$. Because the maximum machine load of the assignment $S^1$ is also bounded by $2C$, we obtain $\sum_{j \in J_2: i \in S_j} f_j(S_j) \leq 4C$.

Finally, we show that the well-structured assignment constructed in Step 3b has maximum load of at most $5C$, establishing the second part of Theorem 10.

**Theorem 14.** If the algorithm reaches Step 3b, it constructs a well-structured assignment with maximum machine load at most $5C$.

**Proof.** As already argued in the description of Step 3b, the produced assignment is well-structured. Because the load of each machine in the assignment $S^1$ is bounded by $2C$, it remains to show that $\sum_{j \in J_2: i \in S^2_j} f_j(S^2_j) \leq 3C$ for each $i \in M$. Note that for $i \in M$ there is at most one $j \in J_2$ with $i \in S^2_j$. Moreover, $f_j(S_j) = q_j/r_j(S_j) \leq q_j/|t_j/2| \leq 3q_j/t_j \leq 3C$ for each $j \in J_2$, where we use the definition of $t_j \geq 2$ for $j \in J_2$ and the fact that $|t_j/2| \geq t_j/3$ for any non-negative integer $t_j \geq 2$. Hence, we can conclude that the maximum machine load is bounded by $5C$. 

13
4 Approximation for Subadditive or Submodular Speeds

In this section, we develop a practically efficient heuristic for both Assignment and Scheduling for a class of processing speed functions that includes submodular functions and monotone subadditive functions. The algorithm yields logarithmic approximation guarantees for submodular or monotone fractionally subadditive processing speeds, and polylogarithmic approximation guarantees for monotone subadditive processing speeds. Furthermore, a simple modification of our heuristic yields improved approximation guarantees for the case of identical jobs.

For fractionally subadditive or subadditive processing speeds, the algorithm requires the access to demand oracle for each speed function $g$, which given a vector $p \in \mathbb{R}^M$ returns a maximizer of $g(S) - \sum_{i \in S} p_i$. We point out that access to such a stronger oracle is necessary for (fractionally) subadditive processing speeds, as for these functions, no meaningful approximation guarantees can be achieved using the standard value oracle within a polynomial number of oracle queries, as we show in the following result (see Appendix D.1 for the proof):

**Theorem 15.** Any algorithm with approximation guarantee $o(n^{3/2})$ for monotone fractionally subadditive processing speeds requires an exponential number of queries to a value oracle.

Before we present our algorithm and its analysis, we first define the class of instances to which our results apply.

**$(\alpha, \beta)$-good functions.** Let $G$ be a class of set functions. We say that $G$ is $(\alpha, \beta)$-good for $\alpha, \beta \geq 1$ if the following three properties are fulfilled:

1. For any instance of Assignment with processing speeds from $G$ and any assignment $S$ for that instance, there exists well-structured assignment $S'$ with $L(S') \leq \alpha L(S)$.

2. The class of functions $G$ is closed under truncation, i.e., for any $g \in G$ and any $t \in \mathbb{R}_{\geq 0}$, the function $g^t$ defined by $g^t(S) := \min\{g(S), t\}$ is contained in $G$.

3. There exists a $\beta$-approximation algorithm for the following Welfare Maximization problem: Given a set of set functions $g_1, \ldots, g_k$ from $G$ on the same ground set $M$, find $k$ disjoint (possibly empty) subsets $T_j \subseteq M$ for $j \in \{1, \ldots, k\}$ so as to maximize $\sum_{j=1}^k g_j(T_j)$.

Our algorithm and its analysis will establish the following metatheorem, which implies the approximation results for subadditive, fractionally subadditive, and submodular processing speeds, respectively, as we show further below. For the remainder of this section, let $n := |J|$ and $m := |M|$.

**Theorem 16.** Let $G$ be an $(\alpha, \beta)$-good class of set functions. Then there exists a polynomial-time $O(\alpha \beta^2 \log \min\{n, m\})$-approximation algorithm for Assignment and Scheduling restricted to instances with processing speeds from $G$.

At the heart of the algorithm lies a subroutine that concurrently runs two assignment procedures: one that tries to assign a maximum number of jobs to individual machines within a given makespan and another that tries to assign a large number of jobs to disjoint sets of machines via a Welfare Maximization problem. These procedures are described and analyzed in Section 4.2, where we show that properties of $(\alpha, \beta)$-good functions ensure that a significant fraction of the jobs can be assigned by at least one of the two procedures. This establishes the following lemma, whose repeated application implies Theorem 16.
Lemma 17. Let $G$ be an $(\alpha, \beta)$-good class of set functions. There is a polynomial-time algorithm that, given an instance of Assignment with processing speeds form $G$ and $C > 0$, returns a subset $J' \subseteq J$ with $|J'| \geq \max\{n - m, \frac{n}{4\beta}\}$ and a well-structured assignment $S'$ for the jobs in $J'$ with $L(S') \leq 4\alpha\beta C$, or correctly outputs that no assignment with maximum load at most $C$ exists for the jobs in $J$.

The algorithm proving Theorem 16 runs in phases. In each phase, the algorithm established in the proof of Lemma 17 is applied to assign a subset of the still unassigned jobs (initially $J$). Lemma 17 guarantees that in each phase, a $\frac{1}{4\beta}$-fraction of the remaining jobs is assigned. Therefore, after at most $\log_{\frac{4\beta}{4\beta - 1}} n = O(\beta \log n)$ phases, all jobs are assigned. Moreover, at most $m$ jobs remain after the first phase and thus the number of phases is also bounded by $O(\beta \log m)$. As the assignments constructed in each phase have a maximum load of at most $4\alpha\beta C$, the concatenation of all these assignments results in an assignment of maximum load at most $O(\alpha\beta^2 \log \min\{n, m\}) C$.

As the partial assignments constructed in each phase are also well-structured, they can be turned into schedules and concatenated without increasing the makespan; see Appendix A.3 for details.

4.1 Consequences of Theorem 16

We now present several corollaries of Theorem 16 for the various classes of speed functions of interest. We remark that the classes of subadditive, fractionally subadditive, and submodular set functions are all closed under truncation.

We first observe that submodular functions given by a value oracle are $(\frac{2e}{e-1}, 4 + \varepsilon)$-good for any $\varepsilon > 0$. This follows from Theorem 1 and the $(4 + \varepsilon)$-approximation algorithm by Lee et al. (2009) for Submodular Maximization Over Matroid Constraints, which generalizes the Welfare Maximization problem for (possibly non-monotone) submodular functions.

Corollary 18. There exists a polynomial-time $O(\log \min\{n, m\})$-approximation algorithm for Assignment and Scheduling restricted to submodular processing speeds.

For monotone fractionally subadditive functions given by a demand oracle, we can obtain similar results. This follows from Theorem 1, which allows us to set $\alpha = \frac{2e}{e-1}$, and the 2-approximation for Welfare Maximization for monotone subadditive functions using a demand oracle, which allows us to set $\beta = 2$.

Corollary 19. There exists a polynomial-time $O(\log \min\{n, m\})$-approximation algorithm for Assignment and Scheduling restricted to monotone fractionally subadditive processing speeds given by a demand oracle.

Finally, note that monotone subadditive functions given via a demand oracle are $(\frac{2e}{e-1} \ln(m), 2)$-good. This follows from Corollary 3 and 2-approximation for Welfare Maximization by Feige (2009), mentioned above.

Corollary 20. There exists a polynomial-time $O(\log m \log \min\{n, m\})$-approximation algorithm for Assignment and Scheduling restricted to monotone subadditive processing speeds given by a demand oracle.

For the case of identical jobs (i.e., with identical processing speed functions) a simple modification of the above heuristic yields a $O(1)$-approximation for Assignment and Scheduling. Indeed, notice that, by Lemma 17, at least $\frac{n}{4\beta}$ jobs are scheduled with a makespan at most $4\alpha\beta C$ during the first phase of the heuristic. Now, instead of moving on to the subsequent phase and repeating the process for the remaining jobs, we create $4\beta$ identical copies of the schedule produced in the first
Corollary 21. There exists a polynomial-time $O(1)$-approximation algorithm for Assignment when all jobs are identical with submodular processing speeds, or when all jobs are identical with monotone fractionally subadditive processing speeds that are given by demand oracle.

Corollary 22. There exists a polynomial-time $O(\log m)$-approximation algorithm for Assignment and Scheduling when all jobs are identical with monotone subadditive processing speed function that are given by demand oracle.

4.2 Proof of Lemma 17

The lemma is based on the following corollary, which is a direct consequence of the properties of $(\alpha, \beta)$-good functions, which guarantee the existence of an $\alpha$-approximate well-structured assignment.

Corollary 23. Assume processing speeds are $(\alpha, \beta)$-good and there is an assignment of maximum load $C$. Then there is an assignment $S$ and sets $J_1, J_2 \subseteq J$ with $J = J_1 \cup J_2$ such that

- $|S_j| = 1$ for all $j \in J_1$ and $\sum_{j \in J_1} f_j(S_j) \leq \alpha C$ for all $i \in M$,
- $S_j \cap S_{j'} = \emptyset$ for all $j, j' \in J_2$ with $j \neq j'$ and $f_j(S_j) \leq \alpha C$ for all $j \in J_2$.

Thus, there exists an assignment such that at least the half the jobs are assigned to individual machines with maximum load $\alpha C$, or at least half the jobs are assigned to disjoint sets of machines, each with a processing time of at most $\alpha C$. As a consequence, one of the two assignment procedures described below will be able to assign either at least $n/2$ jobs with maximum load $2\alpha C$, or at least $n/4$ jobs with maximum load of $4\alpha \beta C$.

The first assignment procedure, which assigns a maximal number of jobs to individual machines within a given target makespan, is based on the following result that follows directly from a classic approximation result for the GENERALIZED ASSIGNMENT problem (Shmoys and Tardos, 1993); see Appendix D.2 for details.

Lemma 24. Assume there exists an assignment $S$ for some set of jobs $J_1 \subseteq J$ with maximum load $C'$ such that $|S_j| = 1$ for all $j \in J_1$. Then we can construct in polynomial-time an assignment that assigns at least $|J_1|$ jobs (possibly different from $J_1$) with a maximum load at most $2C'$.

The second assignment procedure, which assigns jobs to disjoint sets of machines, is based on an approximation algorithm for the WELFARE MAXIMIZATION problem for submodular functions.

Lemma 25. For any $(\alpha, \beta)$-instance of our problem, assume there exists a set $J_2 \subseteq J$ with $|J_2| \geq |J|/2$ and an assignment $S$ of the jobs in $J_2$ with maximum load $C'$ such that $|\{j \in J_2 : i \in S_j\}| \leq 1$ for each $i \in M$. Then we can construct in polynomial time a set of jobs $J' \subseteq J$ with $|J'| \geq |J|/(4\beta - 1)$ and an assignment $S'$ of the jobs in $J'$ with maximum load at most $4\beta C'$.

Proof. For $j \in J$ and $S \subseteq M$ define $h_j(S) := \min\{g_j(S), 1/C'\}$, namely, the truncation of $g_j(S)$ at $1/C'$. Consider the WELFARE MAXIMIZATION problem, which asks for disjoint (possibly empty) subsets $T_j \subseteq M$ for $j \in J$ so as to maximize $\sum_{j \in J} h_j(T_j)$. Note that setting $T_j = S_j$ for $j \in J_2$ and $T_j = \emptyset$ for $j \notin J \setminus J_2$ induces a feasible solution of value $\sum_{j \in J_2} h_j(S_j) \geq |J|/(2C')$. 
In this section, we provide an empirical evaluation of our algorithm. We consider the empirical approximation ratios for the WELFARE MAXIMIZATION problem on the following three classes of processing speed functions. The functions from each of these three classes are known to be monotone submodular.

Let $J := \{ j \in J : h_j(T_j) \geq 1/(4\beta C') \}$ and note that

$$\frac{|J|}{2\beta C'} \leq \sum_{j \in J} h_j(T_j) \leq \frac{|J'|}{C'} + \frac{|J \setminus J'|}{4\beta C'},$$

where the second inequality follows from the fact that $h_j(S) \leq 1/C'$ for all $S \subseteq M$. Rearranging yields $|J| \leq (4\beta - 1)|J'|$, hence setting $S'_j = T'_j$ for $j \in J'$ yields the statement of the lemma.

By Corollary 23, either the premise of Lemma 25 is fulfilled with $C' = \alpha C$, implying $|J|/(4\beta - 1)$ jobs can be assigned within makespan $(4\alpha \beta C)$, or Lemma 24, again with $C' = \alpha C$, guarantees the assignment of $|J_1| = |J \setminus J_2| \geq |J|/2$ jobs in time $2\alpha C$. Finally, for sets $J_1$ and $J_2$ of Corollary 23 it holds $|J_2| \leq m$ (as jobs in $J_2$ are processed on disjoint machine sets) and hence, $|J_1| = |J| - |J_2| \geq |J| - m$, where $m$ is the number of machines. Therefore, $J'$ must also satisfy $|J'| = \max\{|J_1|, |J_2|\} \geq |J| - m$. This concludes the proof of Lemma 17.

5 Computational Study

In this section, we provide an empirical evaluation of our $O(\log \min\{n, m\})$-approximation algorithm presented in Section 4 on synthetic data. We start by describing the submodular speed functions we consider and the LP-based lower bound we use as a baseline. Using this baseline, we then present the empirical approximation ratios for the ASSIGNMENT and SCHEDULING problem for each of these functions on randomly generated input. Finally, we provide examples of the application of Theorem 2 on small instances where the optimal assignment can be computed via an MILP formulation.

5.1 Implementation Details

We remark that we use a simple variant of our $O(\log \min\{n, m\})$-approximation heuristic, where we use the standard greedy $\frac{1}{2}$-approximation algorithm for solving the submodular welfare problem (with monotone functions) of Lemma 25. Even though using the $(1 - \frac{1}{2})$-approximation algorithm due to Vondrak (2008) might lead to better constants in the approximation guarantee, the greedy approach is significantly more practical and easy to implement. For solving the scheduling problem, we first compute an approximate assignment using the heuristic of Section 4 and then, we apply the transformation of Section 2 on the produced assignment.

The algorithm has been implemented in Python 3.8. We have used Gurobi\textsuperscript{TM} 9.1 Solver for all the associated optimization problems (i.e., computation of the optimal solution via an MILP formulation, computation of LP-based lower bounds, and implementation of the algorithm). Finally, we have used SageMath for implementing various operations on matroids. The experiments have been conducted on a machine with 3.2 GHz 8-core CPU and 8 GB RAM.

5.2 Instance Sets

We compute the empirical approximation ratio of our heuristic for a varying number of $n$ jobs and $m$ machines on the following three classes of processing speed functions. The functions from each of these three classes are known to be monotone submodular.
• **Coverage**: We consider $k$ slots and a frequency parameter $p \in (0, 1)$. For each job $j$ and machine $i$, let $B_{i,j} \subseteq [k]$ be a subset of the $k$ slots, such that each slot is contained in $B_{i,j}$ independently with probability $p$. The speed function of each job $j$ on a set $S \subseteq M$ is defined as

$$g_j(S) = \frac{1}{L_j} \cdot \left| \bigcup_{i \in M} B_{i,j} \right|,$$

where $L_j$ is a random constant indicating the total load of job $j$.

In the following simulations, we choose frequency $p = 0.2$ and number of slots $k = m$, where $m$ is the number of machines. For each job $j$, we choose load $L_j \sim U[1, 1000]$.

• **Budget-Additive**: For each job $j$, we define a budget $W_j$ and a set of weights $w_{i,j}$ for each $i \in M$. For the budget and the weights, we have $W_j \sim U[1, \text{Wmax}]$ and $w_{i,j} \sim U[1, \text{wmax}]$, where $\text{Wmax}, \text{wmax}$ fixed constants, with $\text{Wmax} \gg \text{wmax}$. The speed function of each job $j$ of load $L_j$ on a set $S \subseteq M$ is defined as

$$g_j(S) = \frac{1}{L_j} \cdot \min \left\{ \sum_{i \in S} w_{i,j}, W_j \right\}.$$

Specifically, for each job $j$, we choose $L_j \sim U[1, 1000]$, $\text{wmax} = 100$, and $\text{Wmax} = 100 \cdot m$, where $m$ is the number of machines.

• **Matroid-Matching**: For each job $j$, we define $k$ slots, each associated with a weight $w_{\ell,j} \sim U[1, \text{wmax}]$. We define a bipartite graph $G_j$ between the set of slots, $[k]$, and the set of machines, $M$. For each slot $\ell$ and machine $i$, with probability $p$, we consider an edge $\{\ell, i\}$ with weight equal to $w_{\ell,j}$ (such that all the edges adjacent to the same slot have the same weight). For every matching $\sigma \subseteq [k] \times M$ on $G_j$, we denote by $w_j(\sigma)$ the total weight of the edges is $\sigma$.

Further, we consider a random partition $\bigcup_{\ell \in [b]} B_{\ell,j}$ on the $k$ slots, such that each slot is placed into some group uniformly at random. We define a partition matroid on the slots such that any set $R \subseteq [k]$ is independent if $|R \cap B_{\ell,j}| \le r$ for each $\ell \in [b]$, for some rank $r$. Let $I_j$ be the family of independent sets in the above matroid.

For some random load $L_j$, the speed function for job $j$ is defined as

$$g_j(S) = \frac{1}{L_j} \cdot \max \left\{ w_j(\sigma) \mid \sigma \subseteq [k] \times M \text{ a matching on $G_j$ between } R \in I_j \text{ and } S' \subseteq S \right\}.$$

In our simulations, we set $\text{wmax} = 100$, $p = 0.1$, $r = 2$, $k = \left\lceil \frac{n}{4} \right\rceil + 1$, and $b = \left\lceil \frac{2n}{3} \right\rceil + 1$, where $m$ is the number of machines. Finally, for each job $j$, we have $L_j \sim U[1, 1000]$.

### 5.3 Lower Bounds

In order to estimate the empirical approximation ratio of our heuristic, we need to be able to compute (or lower bound) the makespan of the optimal assignment in any given instance.

We take two different routes depending on the size of the instance:
• **MILP formulation:** For relatively small instances, we are able to exactly compute the makespan of the optimal assignment by constructing the following *mixed integer linear programming* (MILP) formulation:

\[
\text{minimize:} \quad C \\
\text{s.t.:} \quad \sum_{S \subseteq M} x(S,j) \geq 1 \quad \forall j \in J \\
\sum_{j \in J} \sum_{S \subseteq M : i \in M} \frac{1}{g_j(S)} x(S,j) \leq C \quad \forall i \in M \\
x(S,j) \in \{0, 1\} \quad \forall j \in J, S \subseteq M \setminus \emptyset .
\]

Note that in order to model all possible assignments from jobs to (subsets of) machines, we need exactly \( n \cdot (2^m - 1) \) binary variables. Unfortunately, the exponential dependence on \( m \) allows the usage of the above formulation only for a small number of machines.

• **LP relaxation:** For larger instances where the MILP approach becomes computationally intractable, we can compute a lower bound to the makespan via the following relaxation:

\[
\text{minimize:} \quad C \\
\text{s.t.:} \quad \sum_{i \in M} x_{ij} \geq 1 \quad \forall j \in J \\
\sum_{j \in J} \frac{1}{g_j(\{i\})} x_{ij} \leq C \quad \forall i \in M \\
\max_{j \in J} \frac{1}{g_j(M)} \leq C \quad \forall j \in J \\
x \geq 0
\]

Let \( C^* \) be the optimal solution of the above LP. It is not hard to see that \( C^* \) is essentially the maximum between a naive relaxation of the problem, where each job \( j \) can be fractionally assigned to each machine \( i \) with processing time \( 1/g_j(\{i\}) \), and the maximum processing time of a job using the whole set of machines.

We remark that \( C^* \) can be quite far from the optimal solution, particularly in the case where the number of jobs is smaller than the number of machines – a fact which is reflected in our simulations. Nevertheless, the above lower bound appears to be sufficient for showing that the empirical approximation ratio of our algorithms relatively small in practice.

### 5.4 Empirical Evaluation of the \( O(\log \min\{n, m\}) \)-Approximation Heuristic

For each of the instances described above we computed a feasible assignment using the heuristic from Section 4 and compared this against the LP-based lower bound described above. The observed ratios of assignment makespan to the respective lower bound for that instance, grouped by number of jobs and machines, are plotted in subfigure (a) of Figs. 3 to 5 for the three different types of speed functions, respectively. We further transformed each of the resulting assignments into a feasible schedule using the algorithm described in Theorem 2. Again we compare the makespan of the resulting schedule against the LP-based lower bound on the makespan of an optimal assignment.
The observed ratios, again grouped by number of jobs and machines, are plotted in subfigure (b) of Figs. 3 to 5 for the different types of speed functions, respectively.

As we observe, the empirical approximation ratios reported in Figs. 3 to 5 take significantly smaller values comparing to the theoretically proven guarantees. It is important to note that, for the scheduling problem, we use as a baseline the same LP-based lower bound used for the assignment problem. This fact implies that the actual approximation ratios for this case can be even smaller than they appear. Finally, for all the speed functions, and especially for Coverage, we observe the following interesting phenomenon: the empirical approximation guarantee appears to be higher when the number of jobs is smaller than the number of machines. In particular, we believe that the above phenomenon is partially explained by the fact that the LP-based lower bound we use is weak for these instances. Indeed, this is because the lower bound interpolates between the case of scheduling the heaviest job on all the machines (which is weak for more than one jobs), and the case where each job can be fractionally assigned to more than one machines (which has an unbounded integrality gap).

5.5 Empirical Evaluation of Theorem 2 on Optimal Assignments

In addition to the empirical evaluation of our heuristic, we further experimentally study the empirical loss due to the application of Theorem 2 on optimal assignments. In this case, for small enough number of jobs and machines, denoted by $n$ and $m$, respectively, an optimal assignment is computed
efficiently via the MILP formulation described above. In Fig. 6, we report the ratio between the makespan of the scheduled produced by applying Theorem 2 on the optimal assignment and the makespan of this assignment, for each of the speed functions defined above. We remark that for the case of MATROID-MATCHING, we slightly modify the number of slots and groups to be $k = \lfloor m \rfloor + 1$ and $b = \lfloor m \rfloor + 1$, respectively. The reason behind this modification is that, for the original values of $k, b$ and for small number of machines, the optimal assignment is always a valid schedule where each job is assigned to a single machine.

As we can see, the approximation loss for all instances lies between 1 and 2, which is smaller than the theoretically proven guarantee of $\frac{2e}{e-1} \approx 3.163$. In fact, the average empirical approximation ratio for all the instances is approximately 1.179. An interesting observation is that in many examples, the reported ratio is exactly 1. This implies that in the optimal assignment, either all jobs are assigned to machines as singletons (and, thus, Theorem 2 has no effect), or that the assignment is already a schedule of nice structure (where each machine contributes to the execution of at most one parallel job) which happens to be maintained by the rounding procedure of Theorem 2. Finally, we can see that for any speed function the approximation loss decreases as the ratio of jobs to machines increases. This phenomenon matches our intuition that, for a fixed number of machines and as the number of jobs increases, more and more jobs are already assigned as singletons in an optimal assignment. The reasoning behind this intuition is that this type of allocation minimizes the total work (number of machines times processing time) invested in each job, and is thus, on a
(a) Assignment using the heuristic of Section 4.

(b) Scheduling via applying Theorem 2 to the produced assignments.

Figure 5: Empirical approximation ratio for MATROID-MATCHING speed functions. Each point in the x-axis is labeled as \( \frac{n}{m} \), where \( n \) and \( m \) is the number of jobs and machines, respectively.

large scale, the most efficient way of using the machines.

References

Hyun-Soo Ahn and Mark E Lewis. Flexible server allocation and customer routing policies for two parallel queues when service rates are not additive. *Operations Research*, 61(2):344–358, 2013.

Francis Bach et al. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3):145–373, 2013.

Evripidis Bampis, Konstantinos Dogeas, Alexander Kononov, Giorgio Lucarelli, and Fanny Pascual. Scheduling malleable jobs under topological constraints. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 316–325. IEEE, 2020.

Enrico Bartolini, Mauro Dell’Amico, and Manuel Iori. Scheduling cleaning activities on trains by minimizing idle times. *Journal of Scheduling*, 20(5):493–506, 2017.

P-E Bernard, Thierry Gautier, and Denis Trystram. Large scale simulation of parallel molecular dynamics. In *Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. IPPS/SPDP 1999*, pages 638–644. IEEE, 1999.
Figure 6: Empirical approximation loss due to Theorem 2 for (a) \textsc{Coverage}, (b) \textsc{Budget-Additive}, and (c) \textsc{Matroid-Matching} speed functions. Each point in the $x$-axis is labeled as $\frac{n}{m}$, where $n$ and $m$ is the number of jobs and machines, respectively.

Kshipra Bhawalkar and Tim Roughgarden. Welfare guarantees for combinatorial auctions with item bidding. In SODA ’11, 2011.

Jacek Blazewicz, TC Edwin Cheng, Maciej Machowiak, and Ceyda Oguz. Berth and quay crane allocation: a moldable task scheduling model. 
Journal of the Operational Research Society, 62 (7):1189–1197, 2011.

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

Olga N Bondareva. Some applications of linear programming methods to the theory of cooperative games. Problemy kibernetiki, 10(119):139, 1963.

Leonardo Borba and Marcus Ritt. A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. Computers & Operations Research, 45:87–96, 2014.

Ozan Candogan, Asuman Ozdaglar, and Pablo A Parrilo. Iterative auction design for tree valuations. Operations Research, 63(4):751–771, 2015.

José Correa, Alberto Marchetti-Spaccamela, Jannik Matuschke, Leen Stougie, Ola Svensson, Víctor Verdugo, and José Verschae. Strong LP formulations for scheduling splittable jobs on unrelated machines. Mathematical Programming, 154(1):305–328, 2015.

Alexandre Dolgui, Sergey Kovalev, Mikhail Y Kovalyov, Sergey Malyutin, and Ameur Soukhal. Optimal workforce assignment to operations of a paced assembly line. European Journal of Operational Research, 264(1):200–211, 2018.

Jack J Dongarra, Iain S Duff, Danny C Sorensen, and Henk A Van der Vorst. Numerical linear algebra for high-performance computers. SIAM, 1998.

Jianzhong Du and Joseph Y-T Leung. Complexity of scheduling parallel task systems. SIAM Journal on Discrete Mathematics, 2(4):473–487, 1989.
Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39(1):122–142, 2009.

Dimitris Fotakis, Jannik Matuschke, and Orestis Papadigenopoulos. Malleable Scheduling Beyond Identical Machines. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:14, Dagstuhl, Germany, 2019.

Ikki Fujiwara, Masahiro Tanaka, Keniiro Taura, and Kentaro Torisawa. Effectiveness of moldable and malleable scheduling in deep learning tasks. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 389–398. IEEE, 2018.

S.L. Hakimi. On the degrees of the vertices of a directed graph. *Journal of the Franklin Institute*, 279(4):290–308, 1965. ISSN 0016-0032.

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

Akio Imai, Hsieh Chia Chen, Etsuko Nishimura, and Stratos Papadimitriou. The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(5):900–920, 2008.

Satoru Iwata. Submodular function minimization. *Mathematical Programming*, 112(1):45–64, 2008.

Klaus Jansen and Felix Land. Scheduling monotone moldable jobs in linear time. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 172–181. IEEE, 2018.

Klaus Jansen and Lorant Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.

Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing*, 39(8):3571–3615, 2010.

Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *STOC ’09*, pages 323–332, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062.

Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

David Simchi Levi, Xin Chen, and Julien Bramel. *The logic of logistics: theory, algorithms, and applications for logistics management*. Springer, 2014.

Gregory Mounie, Christophe Rapine, and Denis Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures*, pages 23–32, 1999.

Grégory Mounié, Christophe Rapine, and Denis Trystram. A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM Journal on Computing*, 37(2):401–412, 2007.
Ioannis Mourtos, Stavros Vatikiotis, and Georgios Zois. Scheduling jobs on unrelated machines with job splitting and setup resource constraints for weaving in textile manufacturing. In *IFIP International Conference on Advances in Production Management Systems*, pages 424–434. Springer, 2021.

Kazuo Murota. *Discrete Convex Analysis*, volume 10. SIAM, 2003.

Koichi Nakade and Rei Nishiwaki. Optimal allocation of heterogeneous workers in a u-shaped production line. *Computers & Industrial Engineering*, 54(3):432–440, 2008.

Nhan-Quy Nguyen, Farouk Yalaoui, Lionel Amodeo, Hicham Chehade, and Pascal Toggenburger. Electrical vehicle charging coordination algorithms framework. In *Energy Management—Collective and Computational Intelligence with Theory and Applications*, pages 357–373. Springer, 2018.

Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

Paolo Serafini. Scheduling jobs on several machines with the job splitting property. *Operations Research*, 44(4):617–628, 1996.

David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1):461–474, 1993.

John Turek, Joel L Wolf, and Philip S Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 323–332, 1992.

Suzanne van der Ster. *The allocation of scarce resources in disaster relief*, 2010. PhD thesis, MSc-Thesis in Operations Research at VU University Amsterdam, 2010.

Jan Vondrak. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC ’08, pages 67–74, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580470.

Matthias Walter and Jürgen Zimmermann. Minimizing average project team size given multi-skilled workers with heterogeneous skill levels. *Computers & Operations Research*, 70:163–179, 2016.

Xiaohu Wu and Patrick Loiseau. Algorithms for scheduling deadline-sensitive malleable tasks. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 530–537. IEEE, 2015.
A Appendix to Section 1

A.1 An Example with Large Assignment-Schedule Gap

Consider \( n \) jobs and \( \frac{n(n-1)}{2} \) machines, where each machine contributes to the execution of exactly two distinct jobs, and each job requires a specific subset of \( n-1 \) machines in order to be executed with processing time 1 (otherwise the job cannot be executed, i.e., \( f_j(S) = \infty \) for all jobs \( j \in J \) and all machine sets \( S \subseteq M \) such that do not contain all \( n-1 \) machines required for job \( j \)). In order to visualize the allocation of machines to jobs, one can think of a clique graph \( K_n \) with \( n \) nodes, where each node represents a distinct job and every edge corresponds to a machine that contributes to the execution of both its adjacent job nodes.

In the only feasible assignment for the above instance, the load of each machine is exactly 2, as each machine processes exactly two jobs of processing time 1. On the other hand, it is not hard to verify that the makespan of any feasible schedule for the same assignment cannot be less than \( n \). Indeed, by construction of our instance, every two jobs share exactly one machine (i.e., the one corresponding to their common edge in the clique) and, thus, their executions cannot be overlapping. Hence, since jobs have to be processed sequentially, each with processing time 1, we conclude that any feasible schedule has makespan at least \( n \).

A.2 IP Formulation for Assignment

The following IP is an exact formulation for the Assignment problem, with binary variables \( x(S,j) \) for every \( j \in J \) and every \( S \subseteq M \), indicating that job \( j \) should be processed on machine set \( S \).

\[
\begin{align*}
\text{minimize:} & \quad C \\
\text{s.t.:} & \quad \sum_{S \subseteq M : S \neq \emptyset} x(S,j) = 1 \quad \forall j \in J \\
& \quad \sum_{j \in J} \sum_{S \subseteq M : i \in S} \frac{1}{g_j(S)} x(S,j) \leq C \quad \forall i \in M
\end{align*}
\]

x \in \{0, 1\}^{|M| \times |J|}

A.3 Approximation Algorithms for Scheduling

Relaxed decision procedures. Many approximation algorithms for makespan minimization rely on the construction of a \( \rho \)-relaxed decision procedure (Lenstra et al., 1990), namely, a polynomial-time routine which, given a value \( C \), either correctly asserts that there exists no assignment (resp., schedule) with makespan less than \( C \), or returns a feasible assignment (resp., schedule) with makespan at most \( \rho C \). It is well-known that access to such a routine immediately implies a polynomial-time \( \rho \)-approximation algorithm for the underlying problem of makespan minimization. Indeed, this can be achieved by binary search over the possible range of \( C \), given that lower and upper bounds on the optimal makespan can be computed. Since all the algorithms presented in this work follow the above regime, we focus on constructing such procedures.

Rounding theorem for scheduling unrelated machines A recurrent theme in our analysis it the use of the following rounding theorem due to Lenstra et al. (1990). Let \( M \) be a set of jobs and \( J \) be a set of machines, where each job must be assigned to a single machine (that is, jobs are non-malleable). Let \( p_{ij} \in \mathbb{R}_{\geq 0} \) be the unrelated processing time for assigning job \( j \) to machine \( i \).
For any $C$, we consider the following feasibility LP:

$$\sum_{i \in M} x_{ij} = 1 \quad \forall j \in J$$
$$\sum_{j \in J} p_{ij} x_{ij} \leq C \quad \forall i \in M$$
$$x_{ij} = 0, \text{ if } p_{ij} > C \quad \forall j \in J \forall i \in M$$
$$x \geq 0$$

**Theorem 26** (Rounding theorem (Lenstra et al., 1990)). Any feasible (fractional) extreme point solution to the above LP can be rounded into an integral solution of makespan at most $2C$.

We now briefly discuss the main idea behind the above result. Given any feasible extreme point solution $x$ to the above LP, we can construct the support graph $G$ as follows: We consider a node $u_j$ for each job $j$ and a node $v_i$ for each machine $i$. For each $x_{ij} > 0$, we define an edge $\{u_j, v_i\}$. It can be proved that $G$ is a pseudoforest, in the sense that all its connected components are either trees, or contain at most one cycle. Hence, the edges of $G$ (and of any pseudoforest) can be oriented in a way that each node has an in-degree of at most one. Given such an orientation, the rounding proceeds in two steps: First, all integral assignments are maintained and the involved edges and job-nodes are removed from $G$. Then, every fractionally assigned job is assigned to any of each children machines. Note that each fractionally assigned job always has at least one child machine by construction of the orientation and by the constraints of the LP. By feasibility of the LP, the load of each machine due to the integrally assigned jobs is at most $C$. Further, due to the orientation, each machine is assigned at most one job from the assignment graph and this job has processing time $p_{ij} \leq C$. Thus, the makespan of the produced schedule is at most $2C$.

**B Appendix to Section 2**

**B.1 Transformation for Subadditive Processing Speeds**

As we already remark in Section 2, the following approximation result is well-known for subadditive functions:

**Lemma 27** (Bhawalkar and Roughgarden, 2011). Let $g : 2^M \rightarrow \mathbb{R}_{\geq 0}$ be a subadditive function. There exists a fractionally subadditive function $h : 2^M \rightarrow \mathbb{R}_{\geq 0}$, such that for any $S \subseteq M$ it holds

$$g(S) \geq h(S) \geq \frac{1}{\ln(|M|)} g(S).$$

For completeness, we now provide a transformation due to Bhawalkar and Roughgarden (2011) that achieves the above guarantee:

Let $S \subseteq M$ be the queried subset. We set $T = \emptyset$ and $\beta(i) = 0$ for each $i \in M$. While $T \neq S$, we repeat:

1. Compute $A = \arg \min_{A' \subseteq S} \frac{g(A')}{|A'|}$.
2. For each $i \in A \setminus T$ (in lexicographic order), set $\beta(i) = \frac{g(A)}{|A|} \frac{1}{\ln(|M|)}$.
3. Set $T = T \cup A$. 
At the end of the above process, we return \( h(S) = \sum_{i \in S} \beta(i) \).

As proved by Bhawalkar and Roughgarden (2011), the function \( h \) which results from the above transformation is fractionally subadditive and satisfies the guarantees of Lemma 27.

**Corollary 3.** Assume that all processing speed functions are subadditive. For any assignment \( S \), there exists a well-structured assignment \( S' \) with \( L(S') \leq \frac{2e}{e-1} \ln(|M|) \cdot L(S) \).

**Proof.** Consider any assignment \( S \) with \( L(S) = C \). Consider a new instance of the Scheduling problem, in which the processing speed function \( g_j \) of each job \( j \) is replaced by its fractionally subadditive approximation \( h_j \) as given by Lemma 27. Note that for the assignment \( S \) the maximum machine load of any machine with respect to processing speeds \( h_j \) is at most \( \ln(|M|) \cdot C \). For this new instance, moreover, Theorem 1 guarantees the existence of a well-structured assignment \( S' \) of maximum machine load at most \( \frac{2e}{e-1} \ln(|M|) \cdot C \) with respect to \( h_j \). Because \( g_j(S) \leq h_j(S) \) for all \( j \in J \) and all \( S \subseteq M \), we conclude that \( L(S) \leq \frac{2e}{e-1} \ln(|M|) \cdot C \) when measuring machine loads for \( S' \) with respect to the original processing speeds \( g_j \) again. \( \square \)

**B.2 Existence of \( \Delta \) (Proof of Lemma 4)**

**Lemma 4.** There exists \( \Delta \in \mathbb{R}^{M \times J} \) such that

1. \( \sum_{i \in S_j} \Delta_{ij} = g_j(S_j) \) for all \( j \in J \),
2. \( \sum_{i \in T} \Delta_{ij} \leq g_j(T) \) for all \( j \in J \) and \( T \subseteq S_j \).

As noted in Section 2, the lemma follows from the Bondareva-Shapley theorem (Bondareva, 1963). For completeness, we give a direct proof using LP duality below.

**Proof.** Let \( j \in J \). Consider the following pair of primal and dual LP.

\[
\begin{align*}
\text{maximize:} & \quad \sum_{i \in S_j} \Delta_{ij} \\
\text{s.t.:} & \quad \sum_{i \in T} \Delta_{ij} \leq g_j(T) \quad \forall T \subseteq S_j \\
\end{align*}
\]

\[
\begin{align*}
\text{minimize:} & \quad \sum_{T \subseteq S_j} \alpha(T) \cdot g_j(T) \\
\text{s.t.:} & \quad \sum_{T \subseteq S_j; i \in T} \alpha(T) \geq 1 \quad \forall i \in S_j \\
\Delta & \geq 0 \\
\alpha & \geq 0
\end{align*}
\]

Note that both LPs have feasible solutions and let \( \Delta \) and \( \alpha \) be optimal solutions to the above LPs. By optimality and feasibility of \( \alpha \), it is not hard to verify that \( 0 \leq \alpha(T) \leq 1 \) for each \( T \subseteq S_j \). By strong duality and the fact that \( g_j \) is fractionally subadditive for each \( j \in J \), we get

\[
\sum_{i \in S_j} \Delta_{ij} = \sum_{T \subseteq S_j} \alpha(T)g_j(T) \geq g_j(S_j).
\]

Therefore, the values \( \Delta_{ij} \) obtained from optimal solutions to the primal LP above, for each \( j \in J \), fulfill the conditions of the lemma. \( \square \)

**B.3 Bounding the Maximum Load in the Reduced Assignment (Lemma 9)**

For convenience, we recall the definitions of \( J_i^1 := \{ j \in J : i_j = i, x'_{ij} > 1/2 \} \) as the set of jobs assigned to any machine \( i \in M \) via a Type 1 assignment and \( \ell_i := \sum_{j \in J_i^1} \frac{1}{x_{ij}} \) as the (fractional) load of \( i \) due to such Type 1 assignments in the extreme point solution \( x' \) to (LP1). We further recall that we defined \( T_j(\theta) := \{ i \in S_j : \ell_i \leq \theta \} \) for \( j \in J \) and that \( \theta_j^* \) was chosen a minimizer of \( \max_{i \in T_j(\theta)} 2\ell_i + f_j(T_j(\theta)) \).
Lemma 9. Let $T$ be the assignment defined by $T_j = T_j(\theta^*)$ if $S'_j = \bar{S}_j$ and $T_j = \{i_j\}$ otherwise. Then $T$ is well-structured and $L(T) \leq \frac{2e}{e-1}L(S)$.

Proof. The fact that $T$ is well-structured follows immediately from the fact that $S'$ is well-structured and that $T_j \subseteq S'_j$ for all $j \in J$. In particular, for every machine $i \in M$, the set of jobs processed by $i$ consists of the jobs $j \in J_1$ assigned to $i$ via a Type 1 assignment and potentially a single job $j' \in J$ with $i \in T_j(\theta^*_j)$ with $x'_{i,j} \leq 1/2$ assigned to $i$ via a Type 2 assignment. Recall that in the proof of Lemma 7, we showed that

$$\sum_{j \in J_1} f_j(\{i\}) \leq 2\ell_i \leq 2C. \quad (1)$$

Thus, if a machine $i \in M$ only processes jobs assigned to it via Type 1 assignments, then its load in $T$ is at most $2C \leq \frac{2e}{e-1}C$. It thus remains to bound the load of machines $i \in M$ for which there exist some Type 2 job $j \in J$ with $i \in T_j(\theta^*_j)$ and $x'_{i,j} \leq 1/2$.

Now fix any such Type 2 job $j \in J$ and note that the load of any machine $i \in T_j = T_j(\theta^*_j)$ is $f_j(T_j(\theta^*)) + \sum_{j \in J_1} f_j(\{i\}) \leq f_j(T_j(\theta^*)) + 2\ell_i$ by (1). The maximum load over all machines in $T_j$ is therefore bounded by

$$\max_{i \in T_j(\theta^*_j)} \{2\ell_i + f_j(T_j(\theta^*_j)) \leq 2\theta^*_j + f_j(T_j(\theta^*_j)),$$

where the inequality follows by definition of $T_j(\theta)$.

As $\theta^*_j$ was chosen to minimize $2\theta + f_j(T_j(\theta))$, it suffices to show that there exists some $\theta > 0$ such that $2\theta + f_j(T_j(\theta)) \leq \alpha C$, for $\alpha := \frac{2e}{e-1}$. Define $h_j(\theta) := \sum_{i \in T_j(\theta)} \Delta_{ij}$ and notice that

$$\int_{\theta=0}^{C} h_j(\theta)d\theta = \sum_{i \in S_j} \Delta_{ij} (C - \ell_i).$$

Note that $\ell_i + \frac{1}{\Delta_{ij}} x_{ij} \leq C$ for each $i \in \bar{S}_j$, by feasibility of the LP solution and the definition of $\ell_i$. By summing over all $i \in \bar{S}_j$ and using the fact that $\sum_{i \in \bar{S}_j} x_{ij} \geq \frac{1}{2}$, we obtain $\frac{1}{2} \leq \sum_{i \in \bar{S}_j} \Delta_{ij} (C - \ell_i)$ and, thus, $\frac{1}{2} \leq \int_{\theta=0}^{C} h_j(\theta)d\theta$.

Now assume by contradiction that $2\theta + f_j(T_j(\theta)) > \alpha C$ for all $\theta > 0$, Not that this is equivalent to $g_j(T_j(\theta)) < \frac{1}{\alpha C - 2\theta}$ and hence we obtain

$$\frac{1}{\alpha C - 2\theta} > g_j(T_j(\theta)) \geq \sum_{i \in T_j(\theta)} \Delta_{ij} = h_j(\theta),$$

where the last inequality follows by Lemma 4 applied to $T = T_j(\theta)$. By combining the above bounds, we obtain

$$\frac{1}{2} \leq \int_{\theta=0}^{C} h_j(\theta)d\theta < \int_{\theta=0}^{C} \frac{1}{\alpha C - 2\theta}d\theta.$$

Multiplying both sides by 2, resolving the integral and using that $\alpha = \frac{2e}{e-1}$ we obtain the contradiction

$$1 < \int_{\omega=0}^{\alpha} \frac{1}{\omega}d\omega = \ln \left( \frac{\alpha}{\alpha - 2} \right) = 1.$$

This shows that $2\theta^*_j + f_j(T_j(\theta^*_j)) \leq \frac{2e}{e-1}C$ and thus completes the proof of the lemma. \qed
C Appendix to Section 3

C.1 Orienting the Graph in Step 3b

Lemma 28 (Hakimi, 1965). Let $G = (V, E)$ be an undirected graph. For $v \in V$, let $d(v)$ be the degree of node $v$ in $G$. There is an orientation of $G$ such that the in-degree of each node $v \in V$ is at least $\lfloor d(v)/2 \rfloor$.

For completeness, we provide a simple proof of the lemma that also explains how such an orientation can be constructed in polynomial time.

Proof. Assume w.l.o.g. that $G$ is a connected graph. Clearly, if $G$ is Eulerian, the orientation can be computed simply by finding an Euler tour and orienting the edges of the cycle in the direction of the tour. If $G$ is not Eulerian, then by the handshaking lemma, we know that the number of odd-degree nodes must be even. In that case, we can compute an arbitrary matching between the odd-degree nodes (i.e., we group these nodes in pairs), and add artificial edges between matched nodes, so that the graph becomes Eulerian (since the degree of all nodes is now even when including the artificial edges). Now, if we compute an Euler tour in the above graph (including the artificial edges) and orient it in a consistent way, the lemma follows simply by dropping the artificial edges. \qed

D Appendix to Section 4

D.1 Hardness of Approximation for Fractionally Subadditive Processing Speeds (Theorem 15)

We prove the inapproximability result for ASSIGNMENT with fractionally subadditive processing speeds given by a value oracle. The proof follows from adapting a construction by Feige (2009) that shows a similar inapproximability for a welfare maximization problem.

Theorem 15. Any algorithm with approximation guarantee $o(n^{1/3})$ for monotone fractionally subadditive processing speeds requires an exponential number of queries to a value oracle.

Proof. We consider an instance of $n \geq 2$ jobs and $m = n^2$ machines, where the set of machines is partitioned into $n$ sets $T_1, \ldots, T_n$ of size $n$ each. For each job $j$ and set of machines $S$, let

$$f_j(S) := \max\{|S| + n^{5/3}, n \cdot |S \cap T_j|\}.$$ 

It is easy to verify that the processing speeds $g_j(S) := 1/f_j(S)$ are fractionally subadditive for every job $j$, and that the optimal makespan in the above instance is OPT = $1/n$ (i.e., when each job $j$ is processed just on its corresponding set $T_j$).

The partition $T_1, \ldots, T_n$ is unknown to the decision maker, who can only access the values of $f_j$ via a value oracle. By the argumentation of Feige (2009, Section 1.2), however, the decision maker cannot find any set $S$ with $|S \cap T_j| \geq \frac{|S|}{n} + n^{2/3}$ within a polynomial number of value queries, since the probability of choosing any such set $S$ when selecting a subset of $M$ uniformly at random is smaller than the inverse of any polynomial in $n$. Hence, we can assume that any algorithm using only a polynomial number of value queries will return an assignment $S$ with $f_j(S_j) := \frac{n}{|S_j| + n^{5/3}}$ for all $j \in J$.

Now, if there is even a single job $j$ with $|S_j| \leq n^{5/3}$, then $f_j(S_j) \geq \frac{1}{2} \cdot n^{-2/3} = \Omega(n^{1/3}) \cdot \text{OPT}$. On the other hand, if $|S_j| \geq n^{5/3}$ for all $j \in J$, then by a volume argument there will be at least one machine running at least $n^{2/3}$ jobs, and each of these jobs takes at least time $\Omega(1/n)$. Thus our makespan is off from OPT by at least a factor of $\Omega(n^{2/3})$. \qed
D.2 Proof of Lemma 24

Lemma 24. Assume there exists an assignment $S$ for some set of jobs $J_1 \subseteq J$ with maximum load $C'$ such that $|S_j| = 1$ for all $j \in J_1$. Then we can construct in polynomial-time an assignment that assigns at least $|J_1|$ jobs (possibly different from $J_1$) with a maximum load at most $2C'$.

Proof. Let $J_1 \subseteq J$ be the subset of $|J_1|$ jobs that are assigned without splitting in the given partial assignment and let $i_j \in M$ be the machine where each job $j \in J_1$ is assigned to. We consider the following LP:

\[
\begin{align*}
\text{maximize:} & \quad \sum_{i \in M} \sum_{j \in J} x_{ij} \\
\text{s.t.:} & \quad \sum_{i \in M} x_{ij} \leq 1 \quad \forall j \in J \\
& \quad \frac{1}{g_j({i_j})} x_{ij} \leq C' \quad \forall i \in M \\
& \quad x_{ij} \equiv 0 \quad \forall i \in M, j \in J \text{ with } f_j({i_j}) > C' \\
& \quad x \geq 0
\end{align*}
\]

By setting $x_{ij} = 1$ for each $j \in J_1$ and $i = i_j$, and $x_{ij} = 0$, otherwise, we can easily verify that all the constraints of (LP-H) are trivially satisfied. Further, for the same assignment, the objective becomes $\sum_{i \in M} \sum_{j \in J} x_{ij} = |J_1|$. Hence, we conclude that (LP-H) is feasible and its optimal value is at least $|J_1|$. Let $x$ be an optimal extreme point solution of (LP-H), which satisfies $\sum_{i \in M} \sum_{j \in J} x_{ij} \geq |J_1|$. Given that for each job $j$ it holds $\sum_{i \in M} x_{ij} \leq 1$, it follows that at least $|J_1|$ jobs appear in the support of $x$ (that is, $\sum_{i \in M} x_{ij} > 0$ for at least $|J_1|$ jobs). Thus, it suffices to prove that we can round $x$ into a feasible schedule in a way such that any job that appears in its support is integrally assigned to a machine.

By working along the lines of Lenstra et al. (1990) (see Appendix A.3), we can construct an assignment graph based on the support of $x$. It can be verified that, if $x$ is an extreme point solution of (LP-H), then the graph is a pseudoforest (Shmoys and Tardos, 1993). Therefore, by using the rounding algorithm of Lenstra et al. (1990) we can schedule all the jobs in the support of $x$ within makespan at most $2C'$ and, thus, the desired lemma follows. \hfill \square