(In-)Approximability Results for Interval, Resource Restricted, and Low Rank Scheduling

Marten Maack
Heinz Nixdorf Institute & Department of Computer Science, Universität Paderborn, Germany

Simon Pukrop
Heinz Nixdorf Institute & Department of Computer Science, Universität Paderborn, Germany

Anna Rodriguez Rasmussen
Department of Mathematics, Uppsala University, Sweden

Abstract

We consider variants of the restricted assignment problem where a set of jobs has to be assigned to a set of machines, for each job a size and a set of eligible machines is given, and the jobs may only be assigned to eligible machines with the goal of makespan minimization. For the variant with interval restrictions, where the machines can be arranged on a path such that each job is eligible on a subpath, we present the first better than 2-approximation and an improved inapproximability result. In particular, we give a $(2 - \frac{1}{24})$-approximation and show that no better than 9/8-approximation is possible, unless P=NP. Furthermore, we consider restricted assignment with R resource restrictions and rank D unrelated scheduling. In the former problem, a machine may process a job if it can meet its resource requirements regarding R (renewable) resources. In the latter, the size of a job is dependent on the machine it is assigned to and the corresponding processing time matrix has rank at most D. The problem with interval restrictions includes the 1 resource variant, is encompassed by the 2 resource variant, and regarding approximation the R resource variant is essentially a special case of the rank $R + 1$ problem. We show that no better than 3/2, 8/7, and 3/2-approximation is possible (unless P=NP) for the 3 resource, 2 resource, and rank 3 variant, respectively. Both the approximation result for the interval case and the inapproximability result for the rank 3 variant are solutions to open challenges stated in previous works. Lastly, we also consider the reverse objective, that is, maximizing the minimal load any machine receives, and achieve similar results.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Restricted Assignment, Approximation, Inapproximability

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.77

Related Version *Full Version*: `https://arxiv.org/abs/2203.06171` [13]

Funding This work was supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" under the project number 160364472 – SFB 901/3.

1 Introduction

Makespan minimization on unrelated parallel machines, or unrelated scheduling for short, is considered a fundamental problem in approximation and scheduling theory. In this problem, a set $\mathcal{J}$ of jobs has to be assigned to a set $\mathcal{M}$ of machines via a schedule $\sigma : \mathcal{J} \to \mathcal{M}$. Each job j has a processing time p_{ij} depending on the machine i it is assigned to and the goal is to minimize the makespan $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. In 1990, Lenstra, Shmoys, and Tardos [10] presented a 2-approximation for this problem and further showed that no better than 1.5-approximation can be achieved (unless P=NP) already for the restricted assignment problem, where each job j has a size p_j and $p_{ij} \in \{p_j, \infty\}$ for each machine i.

For each job j we denote its set of eligible machines by $\mathcal{M}(j) = \{i \in \mathcal{M} \mid p_{ij} = p_j\}$. Closing or narrowing the gap between 2-approximation and 1.5-inapproximability is a famous open problem in approximation [18] and scheduling theory [15]. The present paper deals with certain subproblems of both unrelated scheduling and restricted assignment.

Interval Restrictions. In the variant of restricted assignment with interval restrictions, denoted as RAI in the following, there is a total order of the machines and each job j is eligible on a discrete interval of machines, i.e., $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ and $\mathcal{M}(j) = \{M_\ell, M_{\ell+1}, \ldots M_r\}$ for some $\ell, r \in [m]$. There are several variants and special cases of this problem that are known to admit a polynomial time approximation scheme (PTAS), see [6, 8, 9, 11, 14, 16], the most prominent of which is probably the hierarchical case [11] in which each job is eligible on an interval of the form $\{M_1, M_2, \ldots M_r\}$, i.e., the first machine is eligible for each job. For RAI, on the other hand, there is an $(1+\delta)$-inapproximability result for some small but constant $\delta > 0$ [12]. Furthermore, Schwarz [16] designed a $(2 - 2/(\max_{j \in \mathcal{J}} p_j))$-approximation (assuming integral processing times); and Wang and Sitters [17] studied an LP formulation that provides an optimal solution for the special case with two distinct processing times and some additional assumption.

Resource Restrictions. In the restricted assignment problem with R resource restrictions, or RAR(R), a set $\mathcal{R}$ of R (renewable) resources is given, each machine i has a resource capacity $c_r(i)$ and each job j has a resource demand $d_r(j)$ for each $r \in \mathcal{R}$. The eligible machines are determined by the corresponding resource constraints, i.e., $\mathcal{M}(j) = \left\{i \in \mathcal{M} \,\middle|\, \forall r \in \mathcal{R} : d_r(j) \leq c_r(i)\right\}$ for each job j. It is easy to see, that RAR(1) corresponds to the mentioned hierarchical case which admits a PTAS [11]. On the other hand, there can be no approximation algorithm with ratios smaller than $48/47 \approx 1.02$ or 1.5 for RAR(2) and RAR(4), respectively, unless P=NP, see [12]. The same paper also points out that the case with one resource is a special case of the interval case which in turn is a special case of the two resource case, i.e., RAR(1) $\subset$ RAI $\subset$ RAR(2). While the hierarchical case, i.e. RAR(1), has been studied extensively before, RAR(R) was first introduced in a work by Bhaskara et al. [1], who mentioned it as a special case of the next problem that we consider.

Low Rank Scheduling. In the rank D version of unrelated scheduling, or LRS(D), the processing time matrix (p_{ij}) has a rank of at most D. Alternatively (see [3]), we can assume that each job j has a D dimensional size vector $s(j)$ and each machine i a D dimensional speed vector $v(i)$ such that $p_{ij} = \sum_{k=1}^{D} s_k(j) v_k(i)$. Now, LRS(1) is exactly makespan minimization on uniformly related parallel machines, which is well known to admit a PTAS [7]. Bhaskara et al. [1], who introduced LRS(D), presented a QPTAS for LRS(2) along with some initial inapproximability results for $D > 2$. Subsequently, Chen et al. [4] showed that there can be no better than 1.5-approximation for LRS(4) unless P=NP, and for LRS(3) the same authors together with Marx [3] ruled out a PTAS. On an intuitive level, resource restrictions can be seen as a restricted assignment version of low rank scheduling. However, there is a more direct relationship between the two problems: for each RAR(R) instance there exist LRS($R+1$) instances that are arbitrarily good approximations of the former (see [12]). Hence, any approximation algorithm for LRS($R+1$) can also be used for RAR(R), and any inapproximability result for RAR(R) carries over to LRS($R+1$). In fact many (but not all) inapproximability results for low rank scheduling essentially have this form.

Results. We present improved approximation and inapproximability results for this family of problems. In particular:

- An approximation algorithm for RAI with ratio $2 - \frac{1}{24} \approx 1.96$ presented in Section 2;
- a reduction that rules out a better than 1.5 approximation unless P=NP, i.e., a 1.5-inapproximability result, for RAR(3) presented in Section 3.1;
- a 8/7-inapproximability result for RAR(2) not included in this version of the paper;
- a 9/8-inapproximability result for RAI presented in Section 3.2;
- and a 1.5-inapproximability result for LRS(3) not included in this version of the paper.

All the missing proofs and results can be found in the long version of the paper. The positive result for RAI can be considered the first of the two main contributions of this paper. Finding a better than 2-approximation for RAI was posed as an open challenge in previous works [8, 16, 17]. When considering the respective results in [17] and [16], in particular, it seems highly probable that the actual goal of the research was to address exactly that challenge. The presented approximation algorithm follows the approach of solving and rounding a relaxed linear programming formulation of the problem, which has been used in the classical work by Lenstra et al. [10] and many of the results thereafter. In particular, we extend the so called assignment LP due to Lenstra et al. [10] and design a customized rounding approach. Both the linear programming extension and the rounding approach utilize extensions and refinements of ideas from [16] and [17]. Our result joins the relatively short list of special cases of the restricted assignment problem that do not allow a PTAS and for which an approximation algorithm with rate smaller than 2 is known. Other notable entries are the restricted assignment problem with only two processing times [2] and the so-called graph balancing case [5], where each job is eligible on at most two machines.

The inapproximability results directly build upon the results presented in the paper [12], which in turn utilizes many of the previously published ideas, e.g., from [1, 3–5, 10]. We use the satisfiability problem presented in [12] as the starting point for all of our reductions. For the RAI result in particular, we refine and restructure the respective results from [12] aiming for a significantly better ratio. The respective reduction involves a sorting process and curiously the main improvement in the reduction involves changing a sorting process resembling insertion sort into one resembling bubble sort. Due to this change, the construction becomes locally less complex enabling the use of smaller processing times and hence a stronger inapproximability result. Furthermore, the simplified construction in the result enables us to use the basic structure of the reduction as a starting point for the second main result of the paper, namely, the 1.5-inapproximability result for RAR(3). For this reduction several additional considerations and gadgets are needed, arguably making it the most elaborate of the presented results. The search for an inapproximability result with a reasonably big ratio for RAR(3) was stated as an open challenge in the long version of [3]. Adding the new result yields a very clear picture regarding the approximability of low rank makespan minimization: There is a PTAS for LRS(1), a QPTAS for LRS(2), and a 1.5-inapproximability result for LRS(D) with $D \geq 3$. The last two reductions regarding RAR(2) and RAR(3) yield much improved inapproximability results for the respective problems using comparatively simple and elegant reductions. The result regarding RAR(3), in particular, closes a gap in the results of [12] and also yields an (arguably) easier, alternative proof for the result of [4]. Finally, we note that all of the inapproximability results regarding restricted assignment with resource restrictions can be directly applied to the so called fair allocation or santa claus versions of the problems. In these problem variants, we maximize the minimum load received by the machines rather than minimization of the maximum load, i.e., the objective function is given by $C_{\min}(\sigma) = \min_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$ in this case.

Further Related Work. For a more detailed discussion of related work, we refer to [12] and the long version of this paper.

2 Approximation Algorithm for Makespan Minimization with Interval Restrictions

We establish the first approximation for RAI with an approximation factor better than 2:

▶ **Theorem 1.** *There is a* $(2-\gamma)$*-approximation for* RAI *with* $\gamma = \frac{1}{24}$.

The particular value of the parameter γ is justified in the end. To achieve this result, we first formulate a customized linear program based on the assignment LP due to Lenstra et al. [10] and develop a rounding approach that places different types of jobs in phases. Note that the placement of big jobs with size close to OPT (where OPT is the makespan of an optimal schedule) is often critical when aiming for an approximation ratio of smaller than 2 for a makespan minimization problem. For instance, the classical 2-approximation [10] for restricted assignment produces a schedule of length at most $\text{OPT} + \max_{j\in\mathcal{J}} p_j$ where OPT is the makespan of an optimal schedule and hence the approximation ratio is better if $\max_{j\in\mathcal{J}} p_j$ is strictly smaller than OPT. This is also the case with our approach – the main effort goes into the careful placement of such big jobs. In particular, we place the largest jobs in a first rounding step and the remaining big jobs in a second. All of these jobs have the property that each machine should receive at most one of them and they are placed accordingly. Moreover, the placement is designed to deviate not too much from the fractional placement due to the LP solution. In a last step, the remaining jobs are placed. Each rounding step is based on a simple heuristic approach that considers the machines from left to right and places the least flexible eligible jobs first, i.e., the jobs that have not been placed yet, are eligible on the current machine, and have a minimal last eligible machine in the ordering of the machines. Both the LP and the rounding approach reuse ideas from [16, 17]. Hence, the main novelty lies in the much more elaborate approach for placing the mentioned big jobs in two phases.

In the following, we first establish some preliminary considerations; then formulate the LP and argue that it is indeed a relaxation of the problem at hand; and then discuss and analyze the different phases of the rounding procedure step by step.

Preliminaries. For any integer k, we set $[k] = \{0, \dots, k-1\}$. We apply the standard technique (see [10]) of using a binary search framework to guess a candidate makespan T. The goal is then to either correctly decide that no schedule with makespan T exists, or to produce a schedule with makespan at most $(2-\gamma)T$. Given this guess T, we divide the jobs j into small ($p_j \leq 0.5T$), large ($0.5T < p_j \leq (0.5+\xi)T$) and huge ($(0.5+\xi)T < p_j$) jobs depending on some parameter $\xi = \frac{1}{24}$ which is justified later on. We denote the sets of small, large, and huge jobs as $\mathcal{S}$, $\mathcal{L}$, and $\mathcal{H}$, respectively. Furthermore, we fix the (total) order of the machines such that each job is eligible on consecutive machines. This is possible since we are considering RAI. For the sake of simplicity, we assume $\mathcal{M} = [m]$ with the ordering corresponding to the natural one and set $\mathcal{M}(\ell, r) = \{\ell, \dots, r\}$ for each $\ell, r \in \mathcal{M}$. When considering the machines, we use a left to right intuition with predecessor machines on the left and successor machines on the right. Note, that for each job j there exists a left-most and right-most eligible machine and we denote these by $\ell(j)$ and $r(j)$, respectively, i.e., $\mathcal{M}(j) = \mathcal{M}(\ell(j), r(j))$. For a set of jobs $J \subseteq \mathcal{J}$, we call a job $j \in J$ *least flexible* in J if $r(j)$ is minimal in $\{r(j') \mid j' \in J\}$, and a job j is called *less flexible* than a job j' if $r(j) \leq r(j')$. Lastly, we set $J(\ell, r) = \left\{ j \in J \,\middle|\, \mathcal{M}(j) \subseteq \mathcal{M}(\ell, r) \right\}$ for each set of jobs $J \subseteq \mathcal{J}$ and pair of machines $\ell, r \in \mathcal{M}$, and $p(J) = \sum_{j\in\mathcal{J}} p_j$.

Linear Program. The classical assignment LP (see [10]) is given by assignment variables $x_{ij} \in [0,1]$ for each $i \in \mathcal{M}$ and $j \in \mathcal{J}$ and the following constraints:

$$\sum_{i \in \mathcal{M}} x_{ij} = 1 \qquad \forall j \in \mathcal{J} \tag{1}$$

$$\sum_{j \in \mathcal{J}} p_j x_{ij} \leq T \qquad \forall i \in \mathcal{M} \tag{2}$$

$$x_{ij} = 0 \qquad \forall j \in \mathcal{J}, i \in \mathcal{M} \setminus \mathcal{M}(j) \tag{3}$$

Equation (1) guarantees that each job is (fractionally) placed exactly once; Equation (2) ensures that each machine receives at most a load of T; and due to Equation (3) jobs are only placed on eligible machines. We add additional constraints that have to be satisfied by any integral solution. In particular, we add the following constraints using parameters $UB(\ell, r)$ for each $\ell, r \in \mathcal{M}$ with $\ell \leq r$, which will be properly introduced shortly:

$$\sum_{j \in \mathcal{L} \cup \mathcal{H}} x_{ij} \leq 1 \qquad \forall i \in \mathcal{M} \tag{4}$$

$$\sum_{i \in \mathcal{M}(\ell,r)} \sum_{j \in \mathcal{H}} x_{ij} \leq UB(\ell, r) \qquad \forall \ell, r \in \mathcal{M}, \ell \leq r \tag{5}$$

Equation (4) captures the simple fact that no machine may receive more than one job of size larger than $0.5T$ and was used in [5] as well. The bound $UB(\ell, r)$, on the other hand, is defined in relation to the total load of small jobs that has to be scheduled in the respective interval $\mathcal{M}(\ell, r)$. In particular, we consider the overall load of small jobs that have to be placed in the interval together with the load due to huge jobs with their sizes rounded down to their minimum size. The respective load has to be bounded by T times the number of machines in the interval, i.e., $\sum_{i \in \mathcal{M}(l,r)} \sum_{j \in \mathcal{H}} (0.5 + \xi) T x_{ij} + p(\mathcal{S}(\ell, r)) \leq T|\mathcal{M}(l, r)|$. Since the number of huge jobs placed in an interval is integral for an integral solution, we can therefore set $UB(\ell, r) = \left\lfloor (T|\mathcal{M}(l, r)| - p(\mathcal{S}(\ell, r)))/((0.5 + \xi)T) \right\rfloor$. We note that a constraint similar to Equation (5) is also used in [16, 17]. Summing up, we try to solve the linear program given by Equations (1)–(5) which is indeed a relaxation for RAI. If this is not successful, we reject T and otherwise round the solution x using the procedure described in the following and yielding a rounded solution $\bar{x}$.

Placement of Huge Jobs. Starting with the first machine in the ordering, we place the huge jobs as follows:

- Let i^* be the current machine and H the set of huge jobs that have not been placed yet and are eligible on i^*.
- If $\left\lfloor \sum_{i \in \mathcal{M}(0,i^*)} \sum_{j \in \mathcal{H}} x_{ij} \right\rfloor > \left\lfloor \sum_{i \in \mathcal{M}(0,i^*-1)} \sum_{j \in \mathcal{H}} x_{ij} \right\rfloor$ and $H \neq \emptyset$, place a least flexible job $j \in H$ on i^*, i.e., we set $\bar{x}_{i^* j} = 1$.
- Consider the next machine in the ordering or stop if there is none.

This procedure indeed works and we preserve a connection to the original LP solution:

▶ **Lemma 2.** *All of the huge jobs are placed (on eligible machines) by the above procedure and, for each $\ell, r \in \mathcal{M}$ with $\ell \leq r$, we have $\sum_{i \in \mathcal{M}(\ell,r)} \sum_{j \in \mathcal{H}} \bar{x}_{ij} \leq \left\lceil \sum_{i \in \mathcal{M}(\ell,r)} \sum_{j \in \mathcal{H}} x_{ij} \right\rceil$.*

We note that this first rounding step is very similar to the first rounding step in [17].

Mapping out the Regions. In the next step, we divide the machines into regions, where each region did receive fractional large load of (roughly) one. To that end, we define a set of border machines $\mathcal{B}$ as the machines considered from left to right where the sum of

fractionally placed large jobs hits a new integer, i.e., $\mathcal{B} = \{i' \in \mathcal{M} \mid \lfloor \sum_{i \in \mathcal{M}(0,i')} \sum_{j \in \mathcal{L}} x_{ij} \rfloor > \lfloor \sum_{i \in \mathcal{M}(0,i'-1)} \sum_{j \in \mathcal{L}} x_{ij} \rfloor\}$. Moreover, let $\mathcal{B} = \{i_1, \dots, i_q\}$ with $i_1 < \cdots < i_q$ and i_0 the left-most machine with $\sum_{j \in \mathcal{L}} x_{i_0 j} > 0$. For each $s \in [q] = \{0, \dots, q-1\}$, we may initially define the s-th region as $R^s = \mathcal{M}(i_s, i_{s+1})$. At this point consecutive regions overlap by one machine. We want to change this, while guaranteeing that each region retains at least one *candidate machine* that may receive a large job in the following. In particular, a machine $i \in \mathcal{M}$ is a candidate if it did receive some fractional large or huge job in the LP solution, i.e., $\sum_{j \in \mathcal{H} \cup \mathcal{L}} x_{ij} > 0$, but no huge job afterwards, i.e., $\sum_{j \in \mathcal{H}} \bar{x}_{ij} = 0$. We denote the set of candidate machines as $\mathcal{C}$. For each $s \in [q-1]$, we apply the following procedure in incremental order:

- Check whether region R^s needs the last machine to have at least one candidate, i.e., $\mathcal{M}(i_s, i_{s+1} - 1) \cap \mathcal{C} = \emptyset$.
- If this is the case, we set $R^{s+1} = \mathcal{M}(i_{s+1}+1, i_{s+2})$ and otherwise set $R^s = \mathcal{M}(i_s, i_{s+1}-1)$.

After applying this procedure, we have:

▶ **Lemma 3.** *The regions are non-overlapping and each contain at least one candidate.*

Before proceeding with the placement of the large jobs, we note the following technical observation:

▶ **Lemma 4.** *Let $\ell, r \in \mathcal{M}$ with $\ell \leq r$, $\ell \in R^s$, $r \in R^t$, $k = |\{s, \dots, t\}|$, and* `fracLarge` $= \sum_{i \in \mathcal{M}(\ell,r)} \sum_{j \in \mathcal{L}} x_{ij}$. *Then we have $k - 2 <$* `fracLarge` $< k + 2$. *Furthermore,* `fracLarge` $< k + 1$ *if either $\ell > i_s$ or $r < i_{t+1}$ and* `fracLarge` $< k$ *if both of these conditions hold.*

Placement of Large Jobs. Using the regions, we place the large jobs via the following procedure starting with the first region:

- Let R^* be the current region and L the set of large jobs that have not been placed yet and are eligible on at least one candidate machine from R^*.
- Do the following *twice*: Pick a least flexible large job $j \in L$, place it on the leftmost eligible candidate machine $i \in R^*$, i.e. $\bar{x}_{ij} = 1$, and update L.
- Consider the next region in the ordering or stop if there is none.

Observe that the placement of both the large and huge jobs guarantees that only machines that did receive fractional large or huge load in the LP solution may receive any large or huge job and each such machine receives at most one such job. We argue that this procedure works and also retains some connection to the original LP solution x.

▶ **Lemma 5.** *All large jobs are placed (on eligible machines) by the described procedure and, for each $\ell, r \in \mathcal{M}$ with $\ell \leq r$, we have $\sum_{i \in \mathcal{M}(\ell,r)} \sum_{j \in \mathcal{L}} \bar{x}_{ij} < 2(\sum_{i \in \mathcal{M}(\ell,r)} \sum_{j \in \mathcal{L}} x_{ij} + 2)$.*

Proof. Regarding the second statement note that we place at most 2 jobs in each region and hence Lemma 4 directly yields the proof. As usual, we proof the first statement by contradiction. To that end, assume that there exists a large job j^* that is not placed by the procedure. First note, that there is at least one eligible candidate machine for j^*. To see this, consider the set M of eligible machines $i \in \mathcal{M}(j)$ that either received fractional load of j^* or some huge load, i.e., $x_{ij} > 0$ for $j \in \{j^*\} \cup \mathcal{H}$. Then Equation (4) implies $\sum_{i \in M} \sum_{j \in \mathcal{H}} x_{ij} \leq |M| - 1$. Hence, at most $|M| - 1$ many huge jobs are placed on machines from M due to Lemma 2 and therefore at least one of these machines is a candidate. There are two possibilities why j^* was not placed on such a machine: either a less flexible job got placed on the machine, or two other less flexible jobs were already placed in the same region. Let $r^* = r(j^*)$ and $\ell^* \leq \ell$ be minimal with the property that each large job that was placed

in $\mathcal{M}(\ell^*, r^*)$ is less flexible than j^* and each free candidate machine in the interval is free because two other machines in the same respective region already received a large job less flexible than j^*. Furthermore, let J^* be the set of large jobs placed in $\mathcal{M}(\ell^*, r^*)$ together with j^*. We argue that $\ell(j) \geq \ell^*$ for each $j \in J^*$. Otherwise, there exists a job $j \in J^*$ eligible on machine $\ell^* - 1$. Then there are three possibilities regarding this machine. It was not a candidate before the procedure; it was a candidate and received a job less flexible then j (and therefore also less flexible then j^*); or it was a candidate and did not receive a large job because two other machines in the same region received a job less flexible then j. Each yields a contradiction to the definition of ℓ^*. Let $\texttt{fracLarge} = \sum_{i \in \mathcal{M}(\ell^*, r^*)} \sum_{j \in \mathcal{L}} x_{ij}$ be the sum of fractional large jobs in $\mathcal{M}(\ell^*, r^*)$ according to x. Note that we did show $J^* \subseteq \mathcal{J}(\ell^*, r^*)$ and hence $\texttt{fracLarge} \geq |J^*|$.

Let $M^* \subseteq \mathcal{M}(\ell^*, r^*)$ be the set of machines that did receive a fraction of a job from $J^* \cup \mathcal{H}$. Then Equation (4) implies $\sum_{i \in M^*} \sum_{j \in \mathcal{H}} x_{ij} \leq |M^*| - |J^*|$, and furthermore Lemma 2 yields that at most $|M^*| - |J^*|$ huge jobs are placed on machines from $|M^*|$. Hence, there are at least $|J^*|$ candidate machines in $\mathcal{M}(\ell^*, r^*)$. Since not all of the jobs from J^* have been placed by the procedure, there is therefore at least one free machine in $i^* \in \mathcal{M}(\ell^*, r^*)$. The definition of ℓ^* yields, that two jobs less flexible then j^* have been placed in the same region as i^* and these jobs have to be included in J^* (and the machines they are placed on in $\mathcal{M}(\ell^*, r^*)$).

We now take a closer look at the regions (partially) included in $\mathcal{M}(\ell^*, r^*)$. Let $\ell^* \in R^s$, $r^* \in R^t$, and $k = |\{s, \dots, t\}|$. We consider three cases: If we have $\ell^* = i_s$ and $r^* = i_{t+1}$, i.e., the borders of the interval correspond to the (original) outer borders of their regions, then each of the regions $R^s, \dots, R^t$ did receive at least one job from J^* and one received at least two yielding $k \leq |J^*| - 2 \leq \texttt{fracLarge} - 2$. Moreover, if $\ell^* > i_s$ or $r^* < i_{t+1}$, then one of the regions $R^s, \dots, R^t$ may not have received a job from J^* changing the inequality to $k \leq \texttt{fracLarge} - 1$. Lastly, if both $\ell^* > i_s$ and $r^* < i_{t+1}$, then the two outer regions may have received no job from J^* yielding $k \leq \texttt{fracLarge}$. However, Lemma 4 considers the same three cases, yielding $\texttt{fracLarge} < k + 2$, $\texttt{fracLarge} < k + 1$, and $\texttt{fracLarge} < k$, respectively. ↯ ◀

Placement of Small Jobs. Lastly we place the small jobs. Starting with the first machine, we do the following:

- Let i^* be the current machine and J the set of jobs that have not been placed yet and are eligible on i^*.
- Successively place least flexible jobs j on i^*, i.e., set $\bar{x}_{i^*j} = 1$, until either $J = \emptyset$ or placing the next job would raise the load of i^* above $(2 - \gamma)T$.
- Consider the next machine in the ordering or stop if there is none.

We argue that this procedure works under certain conditions:

▶ **Lemma 6.** *All small jobs are placed (on eligible machines) by the described procedure if $\gamma \leq \xi$, $\gamma + \xi \leq \frac{1}{12}$, and $8\xi + 7\gamma \leq 0.75$ hold. In the resulting schedule, each machine has a load of at most $(2 - \gamma)T$.*

Proof (Idea). The main idea of the proof is to assume that some job cannot be placed and to use this to construct some interval in which each machine received some minimum amount of load and in which each placed job had to be placed in the respective interval. Then Lemma 2, Lemma 5, and the constraints of the LP are used to show a contradiction. Depending on the number of large jobs in the interval, either Equation (2) or Equation (5) are critical. ◀

Lastly, we choose values for ξ and γ which satisfy all the requirements of the above lemma and maximize γ. The biggest γ is achieved by setting $\gamma = \xi = \frac{1}{24}$. This concludes the proof of Theorem 1.

3 Complexity Results

Remember that we use the notation $[n] = \{0, \dots, n-1\}$ for each integer n. The complexity result in this work directly build upon the ones in [12]. In that work, a satisfiability problem denoted as 3-SAT* was introduced and shown to be NP-hard, and all reductions in the present work start from this problem. An instance of the problem 3-SAT* is a conjunction of clauses with exactly 3 literals each. Each of the clauses is either a 1-in-3-clause or a 2-in-3-clause, that is, they are satisfied if exactly one or two of their literals, respectively, evaluate to *true* in a given truth assignment. We denote a k-in-3-clause with literals x, y, and z as $(x, y, z)_k$ and the truth values true and false are denoted as $\top$ and $\bot$ in the following. There are as many 1-in-3-clauses in a 3-SAT* instance as there are 2-in-3-clauses, and, furthermore, each literal occurs exactly twice. Hence, a minimal example for a 3-SAT* instance is given by $(x_0, x_1, \neg x_2)_1 \wedge (\neg x_0, x_1, x_2)_1 \wedge (x_0, \neg x_1, \neg x_2)_2 \wedge (\neg x_0, \neg x_1, x_2)_2$: We have two 1-in-3-clauses and two 2-in-3-clauses, and two occurrences of x_i and $\neg x_i$ for each $i \in [3]$. The formula is satisfied if we map every variable to $\bot$.

In each reduction, we start with an instance I of 3-SAT* with m many 1-in-3-clauses $C_0, \dots, C_{m-1}$, m many 2-in-3-clauses $C_m, \dots, C_{2m-1}$ and n variables $x_0, \dots, x_{n-1}$. Since there are $2m$ clauses with 3 literals each and 4 occurrences for each variable, we have $6m = 4n$. In the following, the precise positions of the occurrences of the variables are important and we have to make them explicit. To this end, let for each $j \in [n]$ and $t \in [4]$ the pair (j, t) correspond to the first or second positive occurrence of variable x_j if $t = 0$ or $t = 1$, respectively, and to the first or second negative occurrence of variable x_j if $t = 2$ or $t = 3$. Furthermore, let $\kappa : [n] \times [4] \to [2m] \times [3]$ be the bijection that maps (j, t) to the corresponding clause index and position in that clause. For instance, in the above example we have $\kappa(0, 2) = (1, 0)$ and $\kappa(2, 1) = (3, 2)$.

Next, we construct an instance I' of the problem considered in the respective case. For the restricted assignment type problems, all job sizes are integral and upper bounded by some constant T such that the overall size of the jobs equals $|\mathcal{M}|T$. Hence, if a machine receives jobs with overall size more or less than T, the objective function value is worse than T for both the makespan and fair allocation case. The goal is to show, that there is a schedule with makespan T for I', if and only if I is a yes-instance. This rules out approximation algorithms with rate smaller than $(T + 1)/T$ for the makespan problem, and with rate smaller than $T/(T - 1)$ for the fair allocation variant since the overall load is $|\mathcal{M}|T$. For the low rank problem, we first design a restricted assignment reduction using the above approach and then show that there exist low rank scheduling instances that approximate the restricted assignment instance with arbitrary precision.

Simple Reduction. We start with a simple reduction for the general restricted assignment problem (with arbitrary restrictions) introducing several ideas and gadgets relevant for all of the following reductions. Note that the reduction is very similar to the one by Ebenlendr et al. [5] and to a reduction in [12].

We have three types of basic jobs and machines, namely, truth assignment machines and jobs that are used to assign truth values to variables, clause machines and jobs that model clauses being satisfied, and variable jobs that connect the first two types:

Table 1 Resource demands and capacities of the jobs and machines, respectively, for the case with 3 resources.

Job/Mach.	Res. 1	Res. 2	Res. 3
$\texttt{TMach}(j,0)$	$4j+1$	$4n-4j$	1
$\texttt{TMach}(j,1)$	$4j+3$	$4n-4j$	0
$\texttt{TJob}(j)$	$4j$	$4n-4j$	0
$\texttt{CMach}(i,s)$, $\kappa^{-1}(i,s)=(j,t)$	$4j+t$	$4n-(4j+t)$	$2+i$
$\texttt{CJob}(i,s)$	0	0	$2+i$
$\texttt{VJob}(j,t)$	$4j+t$	$4n-(4j+t)$	$1-\lfloor\frac{t}{2}\rfloor$

- There are truth assignment machines $\texttt{TMach}(j,q)$ with $j\in[n]$ and $q\in[2]$ and one truth assignment job $\texttt{TJob}(j)$ with size 2 and eligible on $\{\texttt{TMach}(j,0),\texttt{TMach}(j,1)\}$.
- There are clause machines $\texttt{CMach}(i,s)$ for each $i\in[2m]$ and $s\in[3]$ and three clause jobs $\texttt{CJob}(i,s)$ each eligible on $\{\texttt{CMach}(i,s')\mid s'\in[3]\}$. The job $\texttt{CJob}(i,0)$ has size 1, $\texttt{CJob}(i,2)$ has size 2, and $\texttt{CJob}(i,1)$ has size 2 if clause C_i is a 1-in-3-clause and size 1 otherwise.
- Lastly, there are variable jobs $\texttt{VJob}(j,t)$ for each $j\in[n]$ and $t\in[4]$ each of size 1 and eligible on $\{\texttt{TMach}(j,\lfloor\frac{t}{2}\rfloor),\texttt{CMach}(\kappa(j,t))\}$.

First note:

▷ Claim 7. The overall job size $\sum_{j\in\mathcal{J}}p(j)$ is equal to $2|\mathcal{M}|$.

Consider the case that we have a satisfying truth assignment for instance I. If variable x_j is assigned to $\top$, we place $\texttt{TJob}(j)$ on $\texttt{TMach}(j,0)$, $\texttt{VJob}(j,0)$ and $\texttt{VJob}(j,1)$ on $\texttt{CMach}(\kappa(j,0))$ and $\texttt{CMach}(\kappa(j,1))$, respectively, together with local size 1 clause jobs. Furthermore, $\texttt{VJob}(j,2)$ and $\texttt{VJob}(j,3)$ are placed on $\texttt{TMach}(j,1)$ and $\texttt{CMach}(\kappa(j,2))$ and $\texttt{CMach}(\kappa(j,3))$ each receive a local size 2 clause job. If variable x_j is assigned to $\bot$, we place $\texttt{TJob}(j)$ on $\texttt{TMach}(j,1)$, and the placement strategy of the positive and negative variable jobs is reversed. Note that the placement of the clause jobs has to work out since the truth assignment is satisfying. This approach yields a schedule with makespan 2. However, it is also easy to see that a schedule with makespan 2 yields a satisfying truth assignment by basing the assignment of x_j on the placement of $\texttt{TJob}(j)$, and hence we have:

▶ **Lemma 8.** *There is a satisfying truth assignment for I, if and only if there is a schedule with makespan 2 for I'.*

This reduction can be considered the basis of all the other ones considered in this work.

3.1 Three Resources

In the RAR(3) case, we can use essentially the same construction as above. However, the sets of eligible machines are defined using the resources and are slightly different. The resource demands and capacities are specified in Table 1. The choice of resources implies:

▷ Claim 9. We have $\mathcal{M}(\texttt{TJob}(j))=\{\texttt{TMach}(j,0),\texttt{TMach}(j,1)\}$ for each $j\in[n]$ and $\mathcal{M}(\texttt{VJob}(j,t))=\{\texttt{TMach}(j,\lfloor\frac{t}{2}\rfloor),\texttt{CMach}(\kappa(j,t))\}$ for each $j\in[n]$ and $t\in[4]$.

Hence, the truth assignment and variable jobs have the same sets of eligible machines as before. For the clause jobs this is not true, however, a similar claim holds. We call a schedule that assigns a load of exactly T to each machine a T-schedule. Using the fact, that in a 2-schedule each clause machine has to receive at least one clause job, it is easy to show the following:

▷ Claim 10. In any 2-schedule each machine from $\{\texttt{CMach}(i,s) \mid s \in [3]\}$ receives exactly one job from $\{\texttt{CJob}(i,s) \mid s \in [3]\}$.

Hence, Lemma 8 works the same as before and we have:

▶ **Theorem 11.** *There is no better than* 1.5*-approximation for* RAR(3) *and no better than* 2*-approximation for the fair allocation version of this problem, unless P=NP.*

3.2 Interval Restrictions

In order to motivate the new ideas for the RAI reduction and to make them easier to understand, it is helpful to revisit the reduction from [12] first. One of the main ingredients in that result is a simple trick that we will also use extensively.

Pyramid Trick. Consider the following setting: We have 2ℓ consecutive machines and ℓ pairs of jobs. The i-th pair of jobs is eligible on the i-th machine and up to and including the $(2\ell+1-i)$-th machine. Furthermore, we assume that each machine has to receive at least one of the jobs. Then the first and last machine each have to receive one job from the first pair because there are no other eligible jobs that can be processed on these machines. Now, the same argument can be repeated for the second and second to last machine and so on. Hence, machine i and $(2\ell+1-i)$ each have to receive exactly one job from pair i.

Sorting. Next, consider that in the ordering of the machines the truth assignment machines are placed on the left and the clause machines on the right. We could use similar truth assignment and clause jobs as in the reduction in the beginning of this chapter. However, variable jobs each are eligible on one truth assignment and one clause machine. Hence they have to be eligible on all machines in between in a naive adaptation of the reduction to the interval case. If we want to use the pyramid trick to deal with this problem, then intuitively decisions regarding variables in later clauses have to be made before the decisions for variables in earlier clauses and this, of course, cannot be guaranteed regardless of the fixed order of the clauses or variables. The main work in [12] was to remedy this situation by – roughly speaking – sorting the information regarding the variables made in the truth assignment gadget to enable the use of the pyramid trick. To do so several gadgets have been introduced that were intertwined with the truth assignment gadget and carefully build up the ordered information using the pyramid trick and interlocking job sizes. The problem with this approach is that the job sizes can get big rather fast if too many different job types are eligible on the same machines resulting in a high value for T. Now, the main idea in the present work is to decouple the decision and the sorting process and make the sorting process as simple as possible to enable smaller job sizes and therefore a stronger result. Curiously, the sorting process in [12] could be interpreted as some variant of insertion sort, while the one used in the present reduction resembles bubble sort.

Machines and Order. Let $k \in \mathcal{O}(n^2)$ be a parameter to be specified later in this paragraph. In addition to the truth assignment and clause machines, we introduce the following ones:

- For each $j \in [n]$ and $t \in [4]$ there are two gateway machines: one forward $\texttt{FGMach}(j,t)$ and one backward gateway machine $\texttt{BGMach}(j,t)$.
- For each $\ell \in [k]$, $j \in [n]$, and $t \in [4]$ there are two sorting machines: one forward $\texttt{FSMach}(\ell,j,t)$ and one backward sorting machine $\texttt{BSMach}(\ell,j,t)$.

Let $\mathcal{T}$, $\mathcal{G}$, and $\mathcal{C}$ be the sets of truth assignment, gateway, and clause machines, respectively. Moreover, for each $\ell \in [k]$ let $\mathcal{S}_\ell = \{\texttt{BSMach}(\ell, j, t), \texttt{FSMach}(\ell, j, t) \mid j \in [n], t \in [4]\}$ be the ℓ-th set of sorting machines. We define the overall order of the machines by setting an internal order for each set of machines as well as an order of the machine sets. However, before we can do so, we need some additional concepts and notation. In particular, let φ_0 be the sequence of (j,t)-pairs with $j \in [n]$, $t \in [4]$ with increasing lexicographical order and $\psi_0 = \kappa(\varphi_0)$, i.e., $\varphi_0 = ((0,0), \ldots, (0,3), \ldots, (n-1,0), \ldots, (n-1,3))$ and $\psi_0 = (\kappa(0,0), \ldots, \kappa(0,3), \ldots, \kappa(n-1,0), \ldots, \kappa(n-1,3))$. Hence, ψ_0 is a permutation of the pairs (i,s) with $i \in [2m]$ and $s \in [3]$. We consider sorting ψ_0 with the goal of reaching the increasing lexicographical order. Let k be the number of transpositions performed by bubble sort if we do this. Furthermore, let $\psi_{\ell+1}$ for $\ell \in [k]$ be the sequence we get after the first $(\ell+1)$-transpositions and $\varphi_{\ell+1} = \kappa^{-1}(\psi_{\ell+1})$. The use of bubble sort guarantees that $k \in \mathcal{O}(n^2)$ and that two consecutive sequences φ_ℓ, $\varphi_{\ell+1}$ differ only by two consecutive entries that are transposed. For any finite sequence χ, we denote the reversed sequence as $\bar{\chi}$. Now, the ordering is specified as follows:

- The sets are ordered as follows: $\mathcal{T}, \mathcal{G}, \mathcal{S}_0, \ldots, \mathcal{S}_{k-1}, \mathcal{C}$.
- The truth assignment machines are ordered in *increasing* lexicographical order of the indices (j,q).
- The clause machines are ordered in *decreasing* lexicographical order of the indices (i,s).
- The backward gateway machines are placed before the forward gateway machines and for each $\ell \in [k]$ the backward sorting machines are placed before the forward sorting machines from $\mathcal{S}_\ell$ as well.
- The forward and backward gateway machines are ordered in increasing and decreasing lexicographical order of the indices (j,t), i.e., φ_0 and $\bar{\varphi}_0$, respectively.
- For each $\ell \in [k]$, the backward sorting machines are sorted according to the placement of the (j,t) indices in $\bar{\varphi}_\ell$.
- For each $\ell \in [k]$, the forward sorting machines are sorted according to the placement of the (j,t) indices in $\varphi_{\ell+1}$.

The peculiar ordering of the machines is designed to enable the use of the pyramid trick.

Jobs, Sizes, and Eligibilities. We give a full list of all jobs together with their sizes and define the sets of eligible machines by stating the respective first and last eligible machine for each job. We will need one more definition: Let $\xi : [k] \to [n] \times [4]$ be the function that maps ℓ to the distinct pair (j,t) that has a higher index in $\varphi_{\ell+1}$ than in φ_ℓ.

- Truth assignment jobs: For each $j \in [n]$ there is a job $\texttt{TJob}(j)$ with size 2, first machine $\texttt{TMach}(j,0)$ and last machine $\texttt{TMach}(j,1)$.
- Variable jobs: For each $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \bot\}$ there is a job $\texttt{VJob}(j,t,\circ)$ with size 2 if $\circ = \bot$ and 3 otherwise, first machine $\texttt{TMach}(j, \lfloor t/2 \rfloor)$ and last machine $\texttt{BGMach}(j,t)$.
- Gateway jobs: For each $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \bot\}$ there is a job $\texttt{GJob}(j,t,\circ)$ with size 5 if $\circ = \bot$ and 4 otherwise, first machine $\texttt{BGMach}(j,t)$ and last machine $\texttt{FGMach}(j,t)$.
- Bridge jobs: For each $\ell \in [k+1]$, $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \bot\}$ there is a bridge job $\texttt{BJob}(\ell,j,t,\circ)$ with size 1 if $\circ = \bot$ and 2 otherwise, first machine either $\texttt{FGMach}(j,t)$ if $\ell = 0$ or $\texttt{FSMach}(\ell-1,j,t)$ otherwise, and last machine either $\texttt{BSMach}(\ell,j,t)$ if $\ell < k$ or $\texttt{CMach}(\kappa(j,t))$ otherwise.
- Sorting jobs: For each $\ell \in [k]$, $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \bot\}$ there is a job $\texttt{SJob}(\ell,j,t,\circ)$. If $\xi(\ell) = (j,t)$, it has size 4 if $\circ = \bot$ and 3 otherwise, and, if $\xi(\ell) \neq (j,t)$, it has size 7 if $\circ = \bot$ and 6 otherwise. The first machine of $\texttt{SJob}(\ell,j,t,\circ)$ is $\texttt{BSMach}(\ell,j,t)$ and the last machine is $\texttt{FSMach}(\ell,j,t)$.

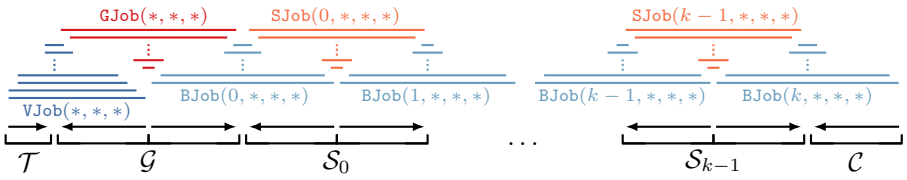

Figure 1 A visualization of the job and machine structure for the reduction regarding RAI. The lower part corresponds to sets of machines with arrows corresponding to the direction of their ordering; and the upper part to different job sets with lines corresponding to intervals of eligible machines for pairs of jobs. Truth assignment and clause jobs as well as private loads are not depicted.

- Clause jobs: For each $i \in [2m]$ and $s \in [3]$ there is a job $\texttt{CJob}(i, s)$ with size 7 if $s = 1$, $8 - k$ if $s = 2$ and C_i is a k-in-3-clause, and 6 if $s = 3$, first machine $\texttt{CMach}(i, 2)$ and last machine $\texttt{CMach}(i, 0)$.
- Private loads: Each truth assignment machine has a private load (a job eligible only one one machine) of 2, each backward or forward gateway machine a load of 1 or 2, respectively, and for each $\ell \in [k]$ the sorting machines $\texttt{BSMach}(\ell, \xi(\ell))$ and $\texttt{FSMach}(\ell, \xi(\ell))$ have a private load of 3.

Using this reduction, we can show:

▶ **Theorem 12.** *There is no better than $\frac{9}{8}$-approximation for* RAI *and no better than $\frac{8}{7}$-approximation for the fair allocation version of this problem, unless P=NP.*

While the formal proof of this result exceeds the scope of this short version of the paper, we briefly discuss the main idea: We have a truth assignment gadget that determines the truth values of the variables and is followed by the gateway gadget, whose sole purpose is to decouple the used job sizes in the truth assignment gadget and the sorting gadget. Next there is the sorting gadget that slowly reorders the information about the decisions in the truth assignment gadget, and lastly there is the clause gadget in which the truth assignment is evaluated. The connection between the truth assignment and gateway gadget is provided by the variable jobs and all other connections are realized via bridge jobs. A sketch of the overall structure of the reduction is provided in Figure 1.

4 Conclusion

We conclude this work with a brief discussion of possible future research directions. There are some obvious questions that can be pursued directly building upon the presented results, i.e., a better approximation ration for RAI or even stronger inapproximability results for RAI or RAR(2). Of course, an improved approximation ratio for any problem of the family would be interesting to develop. We would like to highlight LRS(2), in particular, as the in some sense easiest problem in the family without a known polynomial time approximation with ratio better than 2. Lastly, only very little is known regarding fixed-parameter tractable algorithms for this family of problems. For instance, it is open whether RAR(1) is fixed-parameter tractable with respect to the objective value.

References

1 Aditya Bhaskara, Ravishankar Krishnaswamy, Kunal Talwar, and Udi Wieder. Minimum makespan scheduling with low rank processing times. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 937–947. SIAM, 2013. doi:10.1137/1.9781611973105.67.

2 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On (1, ε)-restricted assignment makespan minimization. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1087–1101. SIAM, 2015. doi:10.1137/1.9781611973730.73.

3 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.22.

4 Lin Chen, Deshi Ye, and Guochuan Zhang. An improved lower bound for rank four scheduling. *Oper. Res. Lett.*, 42(5):348–350, 2014. doi:10.1016/j.orl.2014.06.003.

5 Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.

6 Leah Epstein and Asaf Levin. Scheduling with processing set restrictions: Ptas results for several variants. *International Journal of Production Economics*, 133(2):586–595, 2011. doi:10.1016/j.ijpe.2011.04.024.

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

8 Klaus Jansen, Marten Maack, and Roberto Solis-Oba. Structural parameters for scheduling with assignment restrictions. *Theor. Comput. Sci.*, 844:154–170, 2020. doi:10.1016/j.tcs.2020.08.015.

9 Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. PTAS for ordered instances of resource allocation problems with restrictions on inclusions. *CoRR*, abs/1610.00082, 2016. arXiv:1610.00082.

10 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. doi:10.1007/BF01585745.

11 Chung-Lun Li and Xiuli Wang. Scheduling parallel machines with inclusive processing set restrictions and job release times. *Eur. J. Oper. Res.*, 200(3):702–710, 2010. doi:10.1016/j.ejor.2009.02.011.

12 Marten Maack and Klaus Jansen. Inapproximability results for scheduling with interval and resource restrictions. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.5.

13 Marten Maack, Simon Pukrop, and Anna Rodriguez Rasmussen. (in-)approximability results for interval, resource restricted, and low rank scheduling. *CoRR*, abs/2203.06171, 2022. doi:10.48550/arXiv.2203.06171.

14 Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Oper. Res. Lett.*, 38(1):47–50, 2010. doi:10.1016/j.orl.2009.09.010.

15 Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999. doi:10.1002/(SICI)1099-1425(199909/10)2:5<203::AID-JOS26>3.0.CO;2-5.

16 Ulrich M. Schwarz. *Approximation algorithms for scheduling and two-dimensional packing problems*. PhD thesis, University of Kiel, 2010. URL: http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_00005147.

17 Chao Wang and René Sitters. On some special cases of the restricted assignment problem. *Inf. Process. Lett.*, 116(11):723–728, 2016. doi:10.1016/j.ipl.2016.06.007.

18 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.